Avtonomni komobilni lesistemi snavozila

Gregor.Klančar
Andrej..Zdešar
Sašo....Blažič
Igor...Škrjanc

UL
FE

Univerza v Ljubljani
Fakulteta za elektrotehniko

Avtonomni mobilni sistemi: kolesna vozila

Gregor Klančar
Andrej Zdešar
Sašo Blažič
Igor Škrjanc

Ljubljana, 2021

CIP – Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

KLANČAR, Gregor
Avtonomni mobilni sistemi: kolesna vozila [Elektronski vir] / Gregor Klančar, Andrej Zdešar, Sašo Blažič, Igor Škrjanc; izdajatelj Fakulteta za elektrotehniko. - 1. izd. El. knjiga. - Ljubljana: Fakulteta za elektrotehniko, 2021

Način dostopa (URL):
http://msc.fe.uni-lj.si/ams-kv/ams-kv.pdf
http://msc.fe.uni-lj.si/ams-kv (dodatni material)

Naslovnica je bila oblikovana na podlagi slike, ki je objavljena na spletni strani https://mars.nasa.gov/mars2020/multimedia/raw-images/RLF_0093_0675177851_534EBY_N0040136RHAZ00699_00_0LLJ (dostopano 14. 9. 2021). Slika prikazuje sled kolesa, ki jo je Nasin robotski rover Perseverance pustil na Marsu med tem, ko je obšel oviro. Posnetek je bil zajel 25. maja 2021 z zadnjo levo kamero za izogibanje oviram.

Založnik: Založba FE, Ljubljana

Izdajatelj: UL Fakulteta za elektrotehniko, Ljubljana

Recenzenta: prof. dr. Borut Zupančič, prof. dr. Rihard Karba

1. izdaja

Predgovor

V knjigi so zbrani bistveni algoritmi, ki so potrebni za izvedbo avtonomnih mobilnih sistemov. Knjiga vsebuje pregled obstoječe (raziskovalno aktualne) teorije kot tudi številne avtorske raziskovalne prispevke s področja avtonomnih mobilnih sistemov. Čeprav se knjiga osredotoča na obravnavo algoritmov za kolesna vozila, se lahko z ustreznimi prilagoditvami velik delež predstavljenega materiala uporabi tudi za druge vrste mobilnih sistemov in tudi na mnogih drugih področjih.

Knjiga je po tematikah razdeljana na sedem poglavij, ki jih bralec lahko bere linearno ali pa se osredotoči le na izbrana poglavja s tematikami, ki ga zanimajo. Po uvodnem poglavju 1, ki podaja nekaj osnovnih pojmov in kratek zgodovinski pregled, sledi poglavje 2, ki obravnava modeliranje kinematike gibanja raznovrstnih kolesnih mehanizmov in tudi dinamični model mobilnega sistema z omejitvami. Nato so v poglavju 3 predstavljeni različni pristopi vodenja kolesnih mobilnih sistemov in pristopi načrtovanja poti v poglavju 4. V poglavju 5 je podan pregled senzorjev, ki se uporabljajo v mobilnih sistemih, pri čemer so predstavljene tudi transformacije koordinatnih sistemov. Poglavje 6 pokriva stohastičnost v mobilnih sistemih in obravnava ocenjevanje pošumljenih stanj z Bayesovim in Kalmanovim filtrom ter s filtrom delcev. Na koncu se poglavje 7 dotakne še agentov in večagentnih sistemov, z opisi nekaj praktičnih primerov uporabe.

Teorija je podprta z mnogimi primeri z rešitvami, ki so opremljeni z izvlečki programov v Matlabu. Le-te lahko bralec tudi preizkusi in uporabi pri praktičnem delu. Na spletni strani `http://msc.fe.uni-lj.si/ams-kv` je na voljo elektronska izdaja knjige in dodatni material, ki knjigo dopolnjuje. Poleg elektronske knjige lahko bralec v mapo `src` s spletne strani prenese vse programe, ki so predstavljeni v knjigi. Ime m-datoteke pri izvlečku programa nato deluje kot povezava do celotnega programa.

Knjiga je namenjena vsakomur, ki ga zanima področje avtonomnih mobilnih sistemov z raziskovalnega in/ali praktičnega stališča. Zaželeno je vsaj nekaj osnovnega znanja iz matematičnega modeliranja in simulacij dinamičnih sistemov, teorije regulacij, digitalnega vodenja sistemov, optimizacije, statistike in verjetnosti ter programiranja. Knjiga se lahko uporabi tudi kot gradivo pri predmetih na

dodiplomskem ali podiplomskem študiju, ki obravnavajo mobilno robotiko. Tako je primerno gradivo za študente 2. letnika na podiplomskem študiju 2. stopnje Elektrotehnike na Univerzi v Ljubljani, Fakulteti za elektrotehniko.

Za nastanek knjige so zaslužni sodelavci Laboratorija za avtomatiko in kibernetiko (vključno z bivšimi sodelavci, tudi pod prejšnjimi imeni laboratorija) na Univerzi v Ljubljani, Fakulteti za elektrotehniko. Velika zahvala gre prof. dr. Rihardu Karbi in prof. dr. Borutu Zupančiču za temeljit pregled dela in številne koristne komentarje. Posebna zahvala gre prof. dr. Dragu Matku, ki je področje mobilnih sistemov vpeljal v laboratorij. Hvala as. dr. Matevžu Bošnaku za njegov prispevek na področju avtonomnih mobilnih sistemov. Hvala vsem študentom, raziskovalnim partnerjem in tehničnemu osebju na Univerzi v Ljubjani, Fakulteti za elektrotehniko. Hvala študentki Valentini Stanić za pomoč pri pripravi knjige v slovenščini. Hvala raziskovalcem z institucij po vsem svetu, s katerimi smo sodelovali pri najrazličnejših projektih, ki so vplivali na pripravo te knjige. Hvala tudi Javni agenciji za raziskovalno dejavnost Republike Slovenije, ki je podprla izvedbo mnogih raziskovalnih in aplikativnih projektov.

Ljubljana
September, 2021

G. Klančar, A. Zdešar, S. Blažič, I. Škrjanc

Kazalo

1

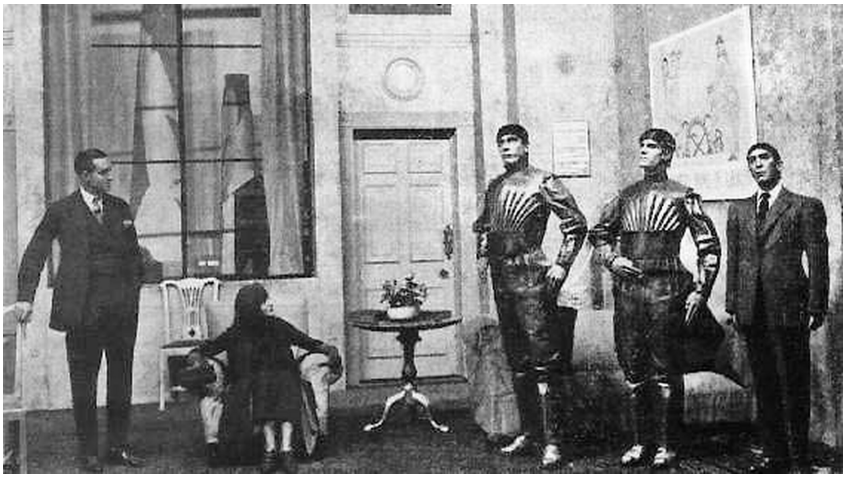
Uvod v mobilne sisteme

1.1 Roboti

Beseda *robot* izvira iz družine slovanskih jezikov. V poljščini beseda "*robota*" pomeni delo, medtem ko je v češčini ali slovenščini bolj arhaična in pomeni "*suženjsko delo*" ali *corvée*. Znani češki pisatelj Karel Čapek je sestavil in uporabil besedo "*robot*" v svoji drami *R.U.R. - Rossumovi Univerzalni Roboti*. Z njo je opisal umetnega človeka, ki bi ga danes lahko poimenovali kiborg ali android (slika 1.1). Zaradi velikega uspeha drame, je besedo prevzela večina svetovnih jezikov. Medtem ko je beseda *robot* nastala pred manj kot sto leti, je sama ideja o mehanskem bitju veliko starejša.

V Grški mitologiji najdemo mnogo bitij, kjer ima vsako svoj namen. *Spartoi* so oboroženi mitološki vojaki, ki so se razvili iz zmajevih zob, katere je posejal Kadmos. Vojaki so Kadmosu pomagali pri izgradnji Kadmeje, tj. trdnjave v Tebah. Talos, ki ga je ustvaril Hefajst, je bil ogromen bronast avtomat, namenjen zaščiti Evrope na Kreti pred pirati in napadalci. Grški bog kovačev in obrtnikov Hefajst je zaslužen tudi za nekatere druge mehanske strukture, ki so bile realizirane. Avtomate lahko najdemo tudi v starodavnih judovskih, kitajskih in indijskih legendah. Skozi celotno zgodovino je prisotna ideja o mehanskem avtomatu, ki je podoben ljudem ali živalim; v 19. in 20. stoletju pa je postala res priljubljena. V 20. stoletju se je pojavil priljubljen medij, ki je upodobil in oživel robote – film. Nekatere ideje v literaturi in filmu so bile v času nastanka označene kot znanstvena fantastika, kasneje pa je ta fikcija postala resničnost.

Vendar pa roboti niso samo stvar fikcije. Zelo zgodnji iznajditelji so skušali ustvari mehanski avtomat. Grški matematik Arhitas naj bi v 4. stoletju p. n. št.



Slika 1.1: Scena iz dramske predstave *R.U.R.*, ki prikazuje tri robote [Fotografija v javni domeni (`https://commons.wikimedia.org/wiki/File%3ACapek_play.jpg`)]

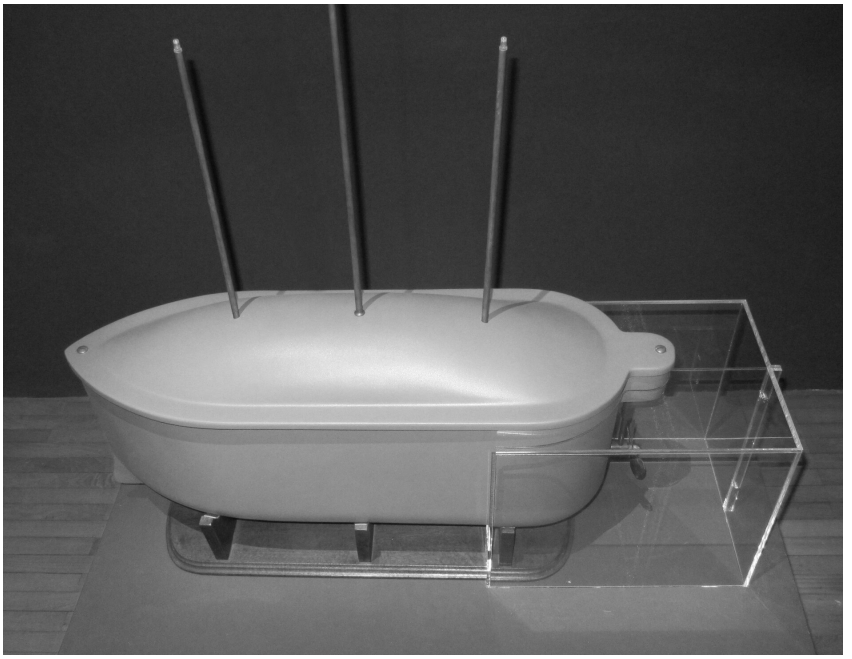
oblikoval in zgradil prvo umetno letečo napravo na lasten pogon. Ta mehanska ptica na parni pogon naj bi bila zmožna preleteti okoli 200 metrov. Leonardo da Vinci je v svoji obširni dediščini zapustil mnogo mehanskih načrtov. S pomočjo grobih skic iz Leonardovih zapiskov je Rosheim [1] rekonstruiral programirljiv voziček (slika 1.2), ki je služil kot podlaga Leonardovim izumom, med katerimi sta bila tudi robotski lev in vitez. Z razmahom industrijske revolucije je tehnološki napredek pripeljal do razcveta avtomatizacije, ki je postopoma vodila do današnje mobilne robotike.

1.2 Mobilnost

Beseda *mobilnost* izhaja iz latinske besede z istim pomenom "*mōbilis*". Večina živalskih vrst ima sposobnost lokomocije, tj. premikanja organizma iz enega mesta na drugo. Medtem ko nekatere živali za premikanje uporabljajo pasivne sisteme (npr. s pomočjo gibanja vode ali zraka), so druge razvile bolj ali manj napredne mehanizme za aktivno gibanje. Ene živali se gibajo v tridimenzionalnem prostoru (plavanje v vodi, letenje po zraku, premikanje po tleh), druge bolj ali manj sledijo dvodimenzionalni površini vode ali tal, tretje pa so zmožne združiti različne načine gibanja. V okviru mobilnih robotov nas zanimajo sistemi, ki se lahko premikajo z uporabo svojega sistema za lokomocijo. Slednji pogosto posnemajo gibanje človeka ali določene živali. Posnemanje bioloških sistemov običajno uspešno rešuje nekatere tehnične težave, ki se pojavijo med načrtovanjem gibanja umetnega sistema.

Drug pomemben vidik mobilnega sistema je avtonomija, saj lahko pri gibanju pride do prevelike oddaljenosti od človeškega operaterja. Tako mora imeti sistem določeno stopnjo avtonomnosti, da se lahko premika po prostoru brez pomoči

Slika 1.2: Model kolesa in programirljivega vozička, zgrajen na podlagi zapiskov Leonarda da Vincija



Slika 1.3: Radijsko krmiljen električni čoln Nikole Tesle

(a) (b)

Slika 1.4: 5200 let staro leseno kolo z osjo, ki so ju našli na Ljubljanskem barju, je glede na starost kot tudi tehnološko dovršenost eno izmed najpomembnejših predmetov svetovne kulturne dediščine (premer kolesa meri 70 cm, dolžina osi pa 120 cm) [Muzej in galerije mesta Ljubljane, avtor M. Paternoster]

operaterja oz. da na daljavo sprejema njegove ukaze. Nikola Tesla je konec 19. stoletja prvi oblikoval in sestavil radijsko voden električni čoln (slika 1.3). Od 20. stoletja nivo avtonomnosti nenehno narašča, vendar človek še vedno na določenem nivoju upravlja obstoječe mobilne sisteme.

1.3 Kolesa

Čeprav se lahko gibljejo tudi zelo primitivne živalske vrste, ni samoumevno razviti umetni sistem, ki je sposoben posnemati gibanje živali. Medtem ko koles in podobnih struktur ni mogoče najti v živalskem svetu, vozila s kolesi omogočajo energetsko učinkovito gibanje po tleh. Površina tal mora biti dovolj gladka, čeprav se lahko ustrezno zgrajena kolesna vozila premikajo tudi po neravnem terenu, stopnicah ipd. Ni znano, kje in kdaj so izumili kolo; uveljavljeno prepričanje je, da so prva kolesa uporabili približno 4000 let p. n. št. v Mezopotamiji in so se od tam razširila po celem svetu. Nekateri strokovnjaki ocenjujejo, da so kolo izumili v prazgodovinski Evropi. Najstarejše ohranjeno leseno kolo z osjo je staro 5200 let in je bilo odkrito v Sloveniji na Ljubljanskem barju (slika 1.4).

1.4 Avtonomni mobilni sistemi

Mobilni sistemi niso (fizično) povezani oziroma vpeti v okolico in se lahko poljubno premikajo v določenem območju. Glede na okolje, v katerem se premikajo, jih lahko razvrstimo v tri skupine:

- **Kopenski mobilni sistemi.** Med njimi najdemo različne vrste mobilnih platform, kot so mobilna vozila s kolesi ali gosenicami, roboti z nogami (humanoidi in roboti, ki posnemajo hojo živali) ter roboti, ki posnemajo druge načine živalskega premikanja (npr. kačje lezenje). Kopenske mobilne sisteme s kolesi ali gosenicami brez operaterja imenujemo kopenska brezpilotna vozila (UGV, angl. *unmanned ground vehicles*).
- **Zračni mobilni sistemi.** Ta skupina je sestavljena iz mobilnih sistemov, ki letijo v določenem zračnem prostoru (letala, helikopterji, droni, rakete in leteči sistemi, ki posnemajo letenje živali). Če letijo brez pilota jih imenujemo brezpilotna zračna vozila (UAV, angl. *unmanned aerial vehicles*).
- **Vodni in podvodni mobilni sistemi.** V tej skupino uvrščamo različne vrste ladij, čolnov, podmornic, avtonomnih podvodnih vozil (AUV, angl. *autonomous underwater vehicles*) ipd.

V knjigi bomo obravnavali samo kolesna mobilna vozila, čeprav se lahko z ustreznimi prilagoditvami velik delež predstavljenega materiala uporabi tudi za druge vrste mobilnih sistemov.

Mobilne sisteme obravnavamo kot avtonomne, če so sposobni avtonomnega gibanja v svoji okolici. Avtonomija mora biti zagotovljena

- z energijskega vidika – robot nosi vir energije,
- z vidika odločanja – robot se je sposoben odločati in izvajati ustrezne akcije.

V praksi to pomeni, da mobilni sistem sprejema ukaze človeškega operaterja glede na stopnjo avtonomnosti, ki mu je vgrajena. Sistem nato poskuša izvesti ukazane naloge in ustrezne "podnaloge" na nižjih nivojih. V primeru predvidljivih okoliščin se naloga izvede v določenem časovnem intervalu. Glede na stopnjo avtonomnosti robota lahko operater izvede naslednje tipične ukaze:

Želene hitrosti koles. Robot sprejema ukaze, ki predstavljajo želene hitrosti koles. Osnovni algoritmi vodenja z ustreznimi senzorji (običajno so to rotacijski dajalniki) omogočajo želeno vrtenje koles glede na ukaz.

Želene translacijske in kotne hitrosti robota. Računalniški program, ki deluje na robotu, pozna njegov kinematični model in lahko izračuna ustrezne hitrosti koles, da doseže želene hitrosti robota.

Želena pot ali trajektorija robota. Robot lahko v svoji okolici ugotovi in vodi svojo lego, ki je običajno določena kot skupna informacija o poziciji in orientaciji glede na izbrani koordinatni sistem. Na tej ravni najdemo lokalizacijo robota s pomočjo različnih senzorjev, nameščenih na robotu ali v okolju, s katerimi poiščemo najboljši približek lege robota. Pri vodenju se pojavijo tudi težave zaradi nelinearnosti sistema, napačnih informacij senzorjev, zdrsa koles, slabih modelov, zakasnitev itd.

Želeno delovanje v znanem okolju z morebitnimi ovirami. Robot mora izvesti opravilo v znanem okolju z nekaj (statičnimi ali dinamičnimi) ovirami. Na tej ravni je robot sposoben načrtovati svojo pot, oz. jo ponovno zasnovati v primeru pojava ovir, ki preprečujejo izpolnitev operacije.

Želena operacija v neznanem okolju. Robot ne pozna svoje okolice, zato mora sočasno izvajati algoritme za določanje položaja in graditi zemljevid svoje okolice – tovrsten pristop je znan kot SLAM (angl. *simultaneous localisation and mapping*).

Želena naloga. Robot prevzame nalogo, ki jo mora izpolniti v okolju, kjer lahko sodeluje z drugimi roboti ali agenti. Robot potrebuje določeno stopnjo razumevanja svojih nalog, poznati pa mora tudi njihove prioritete, da lahko prekine neko nalogo in/ali prevzame nalogo z višjo prioriteto. Številna pravila odločanja morajo biti vgrajena v robota. Recimo robot preveri stanje energije v svojih akumulatorjih in jih po potrebi napolni.

Roboti na prvih dveh (zgoraj opisanih) nivojih niso avtonomni. Obstajajo tudi druge razvrstitve, pri čemer je pomembno, da razumemo naloge in inteligenco robota, ki jih ima na določeni ravni.

Glavni mehanski in elektronski sestavni deli avtonomnega mobilnega robota so:

- **mehanska konstrukcija**: togi in gibljivi sestavni deli (telo, kolesa, gosenice, noge itd.),
- **aktuatorski pogon**: električni motorji (DC, koračni motor, servomotor itd.),
- **senzorji**: rotacijski dajalniki, senzorji bližine in razdalje, inercialna navigacijska enota, globalni navigacijski satelitski sistem (GNSS, angl. *Global navigation satellite system*) itd.,
- **računalniki**: mikrokrmilniki, prenosni osebni računalnik, vgrajeni sistemi itd.,
- **napajalna enota**: baterije, sončne celice itd.,
- **elektronika**: elektronika za pogon motorjev, meritve senzorjev, distribucija moči in telekomunikacijska elektronika.

Knjiga obravnava algoritme, potrebne za obdelavo podatkov senzorjev in pogon motorjev s ciljem zagotavljana avtonomnosti mobilnega robota.

Kolesni mobilni roboti imajo več dobrih lastnosti, zaradi katerih so privlačni za uporabo. Mobilni roboti omogočajo dostop do okolij, nevarnih za ljudi (npr. minska polja, radioaktivna okolja, globokomorske raziskave itd.), in oddaljenih ali nedostopnih okolij (npr. raziskovanje zunajzemeljskih planetov, "nano roboti" v medicini itd.). Hkrati lahko namesto človeka izvajajo naloge, ki so zanj fizično

zahtevne. Uvedba avtomatizacije, robotike in mobilnih sistemov omogoča tudi večjo produktivnost, boljšo kakovost izdelka ali storitve ter zmanjša stroške dela.

Dandanes se mobilni sistemi uporabljajo v številnih aplikacijah na različnih področjih, ki se zaradi hitrega tehnološkega razvoja nenehno širijo. Nepopoln seznam aplikacij kolesnih mobilnih robotov vključuje:

- medicinske storitve, kot je pomoč pri operacijah, opravljanje laboratorijskih analiz (npr. v situacijah, kjer je nevarnost okužbe),
- aplikacije čiščenja (sesanje tal, pometanje in pomivanje v domovih ali velikih zgradbah, čiščenje oken),
- aplikacije v kmetijstvu, kot je avtomatizirano obiranje sadja, sajenje, košnja trave,
- gozdna dela, čiščenje gozdov,
- prodaja blaga široke potrošnje,
- pregled in nadzor nevarnih področij (detekcija in deaktivacija min na minskih poljih, pregled jedrskih reaktorjev, čiščenje kanalizacijskih cevi),
- vesoljske aplikacije (sateliti, pregled in servisiranje satelitov, raziskovanje planetov),
- pomorske aplikacije (roboti za postavljanje kablov in pregledovanje morskega dna),
- roboti za nakladanje in razkladanje blaga ali materiala iz letal, ladij ter tovornjakov,
- vojaški roboti (izvidniški roboti, letala in razni avtopilotski izstrelki),
- varnostni roboti (za nadzor skladišč in stavb),
- pomoč starejšim in invalidnim osebam (avtonomni invalidski vozički, roboti za rehabilitacijo),
- potrošniške aplikacije (robotski hišni ljubljenčki, robotski nogomet),
- sistemi v raziskovalnih ustanovah, namenjeni učenju in razvoju novih algoritmov.

Nekaj od predhodno naštetih področij uporablja tudi že avtonomne mobilne sisteme (AMS), torej robote, ki se lahko samostojno gibljejo v okolici med opravljanjem različnih opravil. Med temi sistemi prevladujejo kolesni avtonomni sistemi kot tudi letalni avtonomni sistemi. Število praktičnih aplikacij AMS in tudi komercialno dostopnih sistemov se povečuje. Uporaba avtonomni sesalnikov, kosilnic in podobnih sistemov je že vsakdanjost. Pogosta je tudi uporaba AMS v tovarnah, bolnišnicah in distribucijskih centrih za dostavo. Obetajoče so tudi

bodoče aplikacije v kmetijstvu in javnem transportu, kjer številni raziskovalni centri razvijajo kmetijske robote in samovozeča vozila, ki bodo kmalu zaživela v uporabi. Najdemo lahko tudi številne druge aplikacije na področjih kot so: vojska za izvidniške namene, misije v vesolju za planetarna raziskovanja, pri naravnih nesrečah za iskanje in reševanje in na področju varovanja. Mobilni avtonomni sistemi predstavljajo hitro razvijajoče področje raziskav in razvoja, zato se bo v bližnji prihodnosti pojavilo še veliko novih aplikacij, ki nam trenutno niso tako očitne. Da dosežemo želeno avtonomijo in novo funkcionalnost, morajo ti sistemi združevati številne tehnologije in opremo. Ključne tehnologije, ki so predstavljene v nadaljevanju te knjige so: modeliranje, vodenje, planiranje, senzorika. lokalizacija in sistemi odločanja.

Predvidevanje prihodnosti je že od nekdaj zahtevna naloga. Današnje tehnologije in aplikacije so bile še pred desetletjem težko predstavljive. Predvideva se, da bodo v bližnji prihodnosti avtonomni kolesni mobilni roboti postali še bolj nepogrešljivi v vsakdanjem življenju: v tovarnah prihodnosti bodo sodelovali z ljudmi, nam pomagali pri domačih opravilih, nas peljali po cesti, reševali življenja (v reševalnih misijah) in še veliko več. V naslednjem poglavju je prikazan kratek pregled zgodovine, oz. kako nas je tehnološki razvoj pripeljal do trenutne točke.

1.5 Kratka zgodovina

Poglavje predstavlja nekaj pomembnih mejnikov v zgodovini kolesnih mobilnih robotov [2]. Poudarek je na aplikacijah, vendar so omenjeni tudi nekateri tehnološki dosežki, ki so pomembno vplivali na področje mobilne robotike.

1898 Nikola Tesla je na sejmu elektronike v dvorani Madison Square Garden v New Yorku demonstriral brezžično radijsko vodeno plovilo [3], ki je eden izmed njegovih patentiranih izumov [4].

1939–1945 Med drugo svetovno vojno so v Nemčiji razvili avtopilotski raketi V-1 [5] in V-2 [6]. Hkrati je Američan Norbert Wiener razvijal sistem za avtomatsko ciljanje protiletalskega orožja [7].

1948–1949 W. Grey Walter je ustvaril avtonomna robota imenovana *Elmer* in *Elsie* [8], ki sta bila podobna želvam in zmožna slediti svetlobnemu viru (tj. fotodioda), zaznavati ovire (kontaktno stikalo) in se izogibati oviram.

1961–1963 Univerza Johns Hopkins je razvila mobilnega robota *Beast* [9], ki je lahko taval po belih hodnikih in iskal črne stenske vtičnice za polnjenje svojih baterij.

1966–1972 Raziskovalni inštitut Stanford je razvijal robota *Shakey* [10], ki je vseboval kamero, sonar, senzorje za zaznavanje trka in brezžično

povezavo. To je bil prvi robot za splošno rabo, ki je znal načrtovati svoje akcije. Rezultati projekta vključujejo razvoj iskalnega algoritma A*, Houghovo transformacijo in graf vidljivosti.

1969 Predstavljena in patentirana prva robotska kosilnica *MowBot* [11].

1970 Sovjetska zveza je na Luni uspešno izkrcala prvi lunarni rover *Lunokhod 1*, ki je bil daljinsko voden z Zemlje ter nosil več kamer in drugih senzorjev. V 301 dneh delovanja je rover prevozil približno 10 km, posnel več kot 25 000 slik in naredil več analiz tal [12].

1973 Sovjetska zveza je na Luni izkrcala drugi lunarni rover *Lunokhod 2*. Med štirimesečno misijo je rover prepotoval 39 km, kar je do leta 2014 veljalo za najdaljšo prepotovano razdaljo izven Zemlje [13].

1976 Nasini vesoljski plovili brez posadke *Viking 1* in *Viking 2* (vsako sestavljeno iz vesoljskega plovila in pristajalnika) sta vstopili v Marsovo obrito, nekoliko dni kasneje pa so pristajalniki mehko pristali na površini Marsa [14].

1977 Francoski laboratorij za analizo in arhitekturo sistemov (LAAS) je začel z razvojem mobilnega robota *Hilare 1* [15], ki je bil opremljen z ultrazvočnimi in laserskimi pregledovalniki razdalj ter kamero na robotski roki.

1979 Vozilo Stanford (angl. *Stanford cart*) (začetni model predstavljen leta 1962) je bilo zmožno vizualne navigacije po progi z ovirami [16].

1982 Na voljo je bil prvi model iz serije komercialnih robotov *HERO*, ki so bili namenjeni predvsem za domačo in izobraževalno rabo [17].

1986 Ekipa pod vodstvom Ernsta Dietera Dickmannsa [18] je razvila robotski avto *VaMoRs*, ki se je lahko sam vozil po ulicah brez prometa s hitrostjo do 90 km/h.

1995 Na tržišču se je pojavil cenovno ugoden mobilni robot *Pioneer* za izobraževalne in raziskovalne namene [19].

1996 Organiziran je bil prvi robotski nogometni turnir, leto kasneje pa je bila ustanovljena FIRA (angl. *Federation of international robot-soccer association*) [20].

1996–1997 NASA je v okviru projekta *Mars Pathfinder* na Mars poslala rover *Sojourner* [21], ki je sprejemal ukaze iz Zemlje ter se je lahko samostojno peljal po vnaprej določeni poti in se pri tem izogibal nevarnim situacijam.

2002 Na tržišču se je pojavil prvi model robotskega sesalnika *Roomba* za domačo rabo [22].

2004 Marsova roverja dvojčka *Spirit* in *Opportunity* sta pristala na Marsu [23]. Rover *Spirit* se je leta 2009 zagozdil, rover *Opportunity* pa je še vedno aktiven in je leta 2014 podrl rekord za najdaljšo zunajzemeljsko prepotovano razdaljo, ki ga je postavil *Lunokhod 2*.

2004 Prvo tekmovanje *DARPA Grand Challenge* je potekalo v puščavi Mojave (ZDA). Nobeno avtonomno vozilo ni dokončalo 240 km dolge proge [24].

2005 Na drugem tekmovanju *DARPA Grand Challenge* je avtonomno vozilo *Stanley* iz Univerze v Stanfordu prvo dokončalo progo. Še štiri druga vozila (od 23) so uspešno opravila nalogo [25].

2007 Organizirano je bilo tekmovanje *DARPA Urban Grand Challenge*, kjer je šest avtonomnih vozil uspešno prevozilo progo v urbanem okolju. Zahtevano je bilo upoštevanje vseh prometnih pravil ter uspešno vključevanje v promet [26].

2009 Izšla je prvotna različica *robotskega operacijskega sistema* ROS 0.4 (angl. *Robot operating system*) [27].

2009 Google je začel (na kalifornijskih avtocestah) preizkušati svojo tehnologijo avtonomne vožnje s predelanim avtom Toyota Prius [28].

2010 V izzivu *VisLab Intercontinental Autonomous Challenge* [29] so štiri avtonomna vozila brez pomoči človeka opravila skoraj 6000 km dolgo potovanje od Parme v Italiji do Šanghaja na Kitajskem.

2012 Na Marsu je uspešno pristal Nasin robotski rover *Curiosity* [30], ki je še vedno aktiven.

2014 Google je razkril nov prototip avtonomnega vozila brez volana in pedalov [28].

2015 Podjetje Tesla v določenih modelih svojih električnih vozil omogoči sisteme za avtonomno vožnjo. V petih letih skupno število kilometrov, ki jih prevozijo lastniki vozil v avtonomnem načinu delovanja (druga stopnja avtonomnosti), preseže 5 milijard [31].

2016 Od junija je flota samovoznih vozil podjetja Google v avtonomnem načinu skupaj prevozila 2 777 585 km [32].

2016 Zagonsko podjetje *comma.ai*, ki ga je ustanovil George Hotz, izda prvo delujočo različico odprtokodne programske opreme za razvoj avtonomne vožnje, ki v osnovi temelji na uporabi kamere za zaznavanje okolja.

2017 Izide prva verzija odprtokodnega simulacijskega okolja CARLA za razvoj algoritmov za avtonomno vožnjo v urbanem okolju [33].

2018 Podjetje Waymo vzpostavi storitev avtonomnega prevoza oseb za končne uporabnike v kraju Phoenix (Arizona) [34].

2019 Podjetje Waymo je s svojimi avtonomnimi vozili prevozilo več kot 10 milijonov kilometrov v resničnem svetu (kar je več kot 200-krat okoli Zemlje oz. 10-krat do Lune in nazaj) in več kot 10 milijard kilometrov v simulacijskem okolju (kar je več kot pot, ki jo Zemlja napravi okoli Sonca v 10 letih) [35].

2021 Na Marsu je marca v kraterju Jezero (v bližini dolin Neretva vallis in Sava vallis) pristal Nasin rover *Perserverance* s helikopterjem *Ingenuity* na krovu [36]. Rover je opremljen s 23 kamerami, od tega se jih 9 uporablja za navigacijo, detekcijo ovir in planiranje poti pri avtonomni vožnji. Dva meseca kasneje je na Marsu pristal še kitajski rover Žurong [37].

Literatura

[1] M. Rosheim. *Leonardo's Lost Robots*. Springer Berlin Heidelberg, 2006.

[2] Mobile robot. `https://en.wikipedia.org/wiki/Mobile_robot`, 2020. Dostopano: 5. 10. 2020.

[3] Nikola Tesla. `https://en.wikipedia.org/wiki/Nikola_Tesla`, 2020. Dostopano: 5. 10. 2020.

[4] N. Tesla. Method of and apparatus for controlling mechanism of moving vessels or vehicles, 1898. US Patent 613,809.

[5] V-1 flying bomb. `https://en.wikipedia.org/wiki/V-1_flying_bomb`, 2020. Dostopano: 5. 10. 2020.

[6] V-2 rocket. `https://en.wikipedia.org/wiki/V-2_rocket`, 2020. Dostopano: 5. 10. 2020.

[7] D. Jerison in D. Stroock. Norbert Wiener. *Notices of the AMS*, zv. 42, št. 4, str. 430–438, 1995.

[8] William Grey Walter. `https://en.wikipedia.org/wiki/William_Grey_Walter`, 2020. Dostopano: 5. 10. 2020.

[9] D. P. Watson in D. H. Scheidt. Autonomous systems. *Johns Hopkins APL technical digest*, zv. 26, št. 4, str. 368–376, 2005.

[10] N. J. Nilsson. Shakey the robot. Tehn. por., SRI International, 1984.

[11] S. L. Bellinger. Self-propelled random motion lawnmower, 1972. US Patent 3,698,523.

[12] I. Karachevtseva, J. Oberst in sod. Cartography of the Lunokhod-1 landing site and traverse from LRO image and stereo-topographic data. *Planetary and Space Science*, zv. 85, str. 175–187, 2013.

[13] Lunokhod 2. `https://en.wikipedia.org/wiki/Lunokhod_2`, 2020. Dostopano: 5. 10. 2020.

[14] Viking project information. `http://nssdc.gsfc.nasa.gov/planetary/viking.html`, 2020. Dostopano: 5. 10. 2020.

[15] G. Giralt, R. Chatila in M. Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. V *Autonomous robot vehicles*, str. 420–443. Springer, 1990.

[16] H. P. Moravec. The Stanford cart and the CMU rover. *Proceedings of the IEEE*, zv. 71, str. 872–884, 1983.

[17] HERO (robot). `https://en.wikipedia.org/wiki/HERO_(robot)`, 2020. Dostopano: 5. 10. 2020.

[18] E. D. Dickmanns. *Dynamic vision for perception and control of motion*. Springer Science & Business Media, 2007.

[19] Pioneer 3 - Robots: Your guide to the world of robotics. `https://robots.ieee.org/robots/pioneer`, 2020. Dostopano: 5. 10. 2020.

[20] FIRA RoboworldCup Official Website. `https://www.firaworldcup.org`, 2020. Dostopano: 5. 10. 2020.

[21] NASA – Mars Pathfinder and Sojourner. `http://www.nasa.gov/mission_pages/mars-pathfinder/`, 2019. Dostopano: 5. 10. 2020.

[22] iRobot history. `http://www.irobot.com/About-iRobot/Company-Information/History.aspx`, 2020. Dostopano: 5. 10. 2020.

[23] NASA Mars Exploration Rovers – Spirit and Opportunity. http://www.nasa.gov/mission_pages/mer/index.html, 2018. Dostopano: 5. 10. 2020.

[24] DARPA Grand Challenge 2004. https://archive.darpa.mil/grandchallenge04, 2014. Dostopano: 5. 10. 2020.

[25] DARPA Grand Challenge 2005. https://archive.darpa.mil/grandchallenge05, 2014. Dostopano: 5. 10. 2020.

[26] DARPA Urban Challenge. https://archive.darpa.mil/grandchallenge, 2014. Dostopano: 5. 10. 2020.

[27] ROS.org, history. http://www.ros.org/history/, 2020. Dostopano: 5. 10. 2020.

[28] Waymo. https://waymo.com, 2020. Dostopano: 5. 10. 2020.

[29] The VisLab Intercontinental Autonomous Challenge. https://vislab.it/viac, 2020. Dostopano: 5. 10. 2019.

[30] NASA Mars Science Laboratory – Curiosity. https://www.nasa.gov/mission_pages/msl/index.html, 2020. Dostopano: 5. 10. 2020.

[31] Lex Fridman Tesla Vehicle Deliveries and Autopilot Mileage Statistics. https://lexfridman.com/tesla-autopilot-miles-and-vehicles, 2020. Dostopano: 1. 3. 2021.

[32] Google Self-Driving Car Project: Monthly Report, June 2016. https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar/files/reports/report-0616.pdf, 2016. Dostopano: 18. 7. 2016.

[33] A. Dosovitskiy, G. Ros in sod. CARLA: An open urban driving simulator. V *Proceedings of the 1st Annual Conference on Robot Learning*, str. 1–16. 2017.

[34] Waymo Story. https://waymo.com/company/#story, 2021. Dostopano: 1. 3. 2021.

[35] TechCrunch Waymo has now driven 10 billion autonomous miles in simulation. https://techcrunch.com/2019/07/10/waymo-has-now-driven-10-billion-autonomous-miles-in-simulation, 2019. Dostopano: 1. 3. 2021.

[36] NASA Mars Perseverance Rover. https://www.nasa.gov/perseverance, 2021. Dostopano: 1. 3. 2021.

[37] CNSA Probe makes historic landing on Mars. http://www.cnsa.gov.cn/english/n6465652/n6465653/c6812005/content.html, 2021. Dostopano: 25. 5. 2021.

2

Modeliranje gibanja mobilnih sistemov

2.1 Uvod

Človek že več tisoč let izkorišča prednosti kolesnega pogona. Osnovna zgradba prazgodovinskega dvokolesnega vozička (slika 2.1) je enaka tisti v modernih avtomobilih in kolesnih robotih. V tem poglavju je predstavljeno modeliranje gibanja različnih kolesnih mobilnih sistemov. Dobljeni model se lahko uporabi v različne namene. V knjigi ga bomo večinoma uporabljali za načrtovanje strategij lokomocije sistema. **Lokomocija** je proces gibanja avtonomnega sistem z enega mesta na drugo.

Modeli gibanja lahko opisujejo kinematiko robota, kjer nas zanima matematičen zapis gibanja brez upoštevanja sil in navorov, ki v splošnem tako gibanje povzročijo. **Kinematični model** opisuje geometrijske relacije v sistemu, to so relacije med vhodnimi parametri in vedenjem sistema, ki jih podajajo stanja sistema. Kinematični model opisuje hitrosti sistema in je predstavljen z množico diferencialnih enačb prvega reda.

Dinamični model pa opisuje gibanje sistema zaradi sil, ki delujejo nanj. Tovrstni model vključuje fizikalne veličine, kot so sile, energije, masa sistema, vztrajnost in hitrosti. Opisi dinamičnih modelov so podani z diferencialnimi enačbami drugega reda.

Pri načrtovanju gibanja kolesnih mobilnih robotov običajno uporabimo kinematične modele, medtem ko za druge (bolj kompleksne) sisteme, kot so zračna

Slika 2.1: Dvokolesni voziček [Muzej in galerije mesta Ljubljane, slikar: I. Rehar]

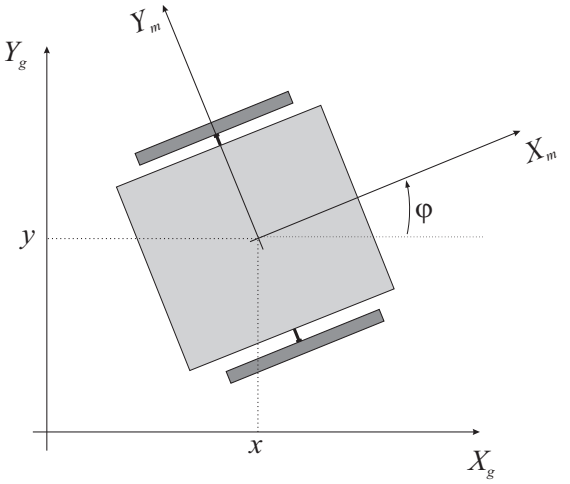
plovila, zračni in nožni roboti, hitra kolesna vozila ipd., uporabljamo dinamične modele gibanja.

2.2 Kinematika kolesnih mobilnih sistemov

Obstaja več različnih kinematičnih modelov:

- **Notranja kinematika** pojasnjuje relacije med notranjimi spremenljivkami sistema (npr. kako vrtenje koles vpliva na gibanje vozila).
- **Zunanja kinematika** opisuje pozicijo in orientacijo vozila glede na referenčni koordinatni sistem.
- **Direktna kinematika** modelira stanja sistema kot funkcijo vhodov (hitrosti koles, gibanje sklepov, zasuk krmilnega kolesa itd.), **inverzna kinematika** pa se uporablja za načrtovanje gibanja, torej podaja vhode v sistem, ki so potrebni za doseg želenega stanja.
- **Omejitve gibanja** se tipično pojavijo, ko ima sistem manj vhodnih spremenljivk kot prostostnih stopenj (neholonomične omejitve). Holonomične omejitve omejujejo dosegljivost določenih stanj sistema, medtem ko neholonomične omejitve omejijo smeri možnih premikov sistema (kolesa robota se lahko vrtijo le v smeri njihove orientacije). Število prostostnih stopenj je minimalno število stanj s katerimi lahko opišemo konfiguracijo sistema.

V nadaljevanju sledi nekaj primerov določitve notranje kinematike kolesnih



Slika 2.2: Vozilo v ravnini

mobilnih robotov. Lega robota v ravnini je podana z vektorjem stanj

$$\boldsymbol{q}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \varphi(t) \end{bmatrix}$$

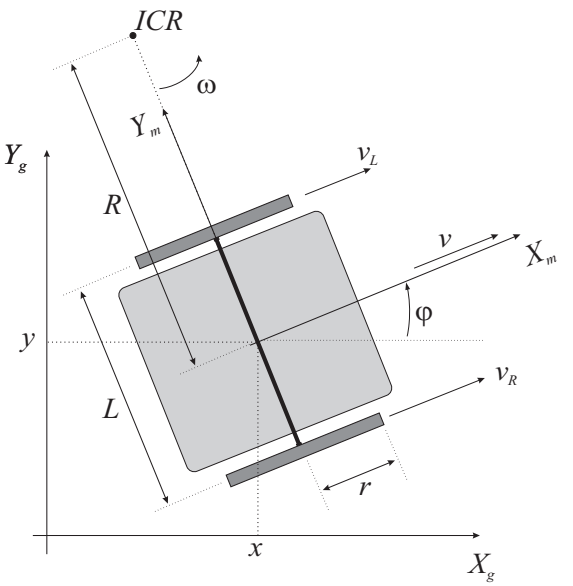
v globalnih koordinatah $(X_g,\ Y_g)$, kot je prikazano na sliki 2.2. Premični koordinatni sistem $(X_m,\ Y_m)$ je pripet na mobilnega robota. Relacija med globalnim in premičnim koordinatnim sistemom (zunanja kinemtika) je podana z vektorjem translacije $[x,\ y]^T$ in rotacijsko matriko

$$\boldsymbol{R}(\varphi) = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Kolesni mobilni robot se giblje s pomočjo koles, ki se vrtijo zaradi trenja med njimi in podlago. Pri zmernih hitrostih običajno predpostavimo model **idealnega kotaljenja koles**, kjer se lahko kolo premika le zaradi rotacije (kotaljenja), brez zdrsov v smeri kotaljenja ali pravokotno na smer kotaljenja. Vsako kolo se lahko prosto vrti okoli lastne osi, torej obstaja točka, ki leži na presečišču vseh osi koles. Ta točka se imenuje **trenutni center rotacije** (ICR, angl. *instantaneous center of rotation*) ali trenutni center ukrivljenosti (ICC, angl. *instantaneous center of curvature*) in določa točko, okoli katere vsa kolesa krožijo z enako krožno hitrostjo ω glede na ICR. Za nadaljnje branje si lahko pogledate [1–3].

2.2.1 Diferencialni pogon

Diferencialni pogon je zelo preprost in zato precej pogosto uporabljen mehanizem pogona, predvsem pri manjših vozilih ali mobilnih robotih. Vozilo s takim



Slika 2.3: Kinematika diferencialnega pogona

pogonom ima ponavadi eno ali dve dodatni podporni kolesi (angl. *castor*), ki podpirata vozilo in preprečujeta njegovo prevračanje. Kolesi diferencialnega pogona sta vpeti na skupno os, hitrost vrtenja vsakega kolesa pa je poljubna in gnana s svojim motorjem. Glede na sliko 2.3 sta vhodni (regulirni) spremenljivki hitrost desnega kolesa $v_R(t)$ in hitrost levega kolesa $v_L(t)$. Ostale spremenljivke na sliki 2.3 so: r – radij kolesa, L – razdalja med kolesoma in $R(t)$ – trenutni radij trajektorije vožnje vozila oz. razdalja med središčem vozila (središčna točka med kolesoma) in točko ICR. V vsakem časovnem trenutku imata obe kolesi enako kotno hitrost $\omega(t)$ okrog ICR

$$\omega(t) = \frac{v_L(t)}{R(t) - \frac{L}{2}}$$
$$\omega(t) = \frac{v_R(t)}{R(t) + \frac{L}{2}}$$

od koder izrazimo $\omega(t)$ in $R(t)$ kot

$$\omega(t) = \frac{v_R(t) - v_L(t)}{L}$$
$$R(t) = \frac{L}{2}\frac{v_R(t) + v_L(t)}{v_R(t) - v_L(t)}$$

Tangencialna hitrost vozila je

$$v(t) = \omega(t)R(t) = \frac{v_R(t) + v_L(t)}{2}$$

Obodni hitrosti koles sta $v_L(t) = r\omega_L(t)$ in $v_R(t) = r\omega_R(t)$, kjer sta $\omega_L(t)$ in $\omega_R(t)$ kotni hitrosti levega in desnega kolesa okoli njune osi. Upoštevajoč navedene

relacije lahko zapišemo notranjo kinematiko (v lokalnih koordinatah) kot

$$\begin{bmatrix} \dot{x}_m(t) \\ \dot{y}_m(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} v_{X_m}(t) \\ v_{Y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{L} & \frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix} \tag{2.1}$$

Zunanja kinematika robota (v globalnih koordinatah) pa je

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \cos\varphi(t) & 0 \\ \sin\varphi(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{2.2}$$

kjer sta $v(t)$ in $\omega(t)$ vhodni (regulirni) spremenljivki. Model (2.2) lahko s pomočjo Eulerjeve integracijske metode zapišemo v diskretni obliki (2.3), ki je veljavna za diskretne čase vzorčenja $t = kT_s$, $k = 0, 1, 2, \ldots$, kjer je T_s čas vzorčenja

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_s\cos\varphi(k) \\ y(k+1) &= y(k) + v(k)T_s\sin\varphi(k) \\ \varphi(k+1) &= \varphi(k) + \omega(k)T_s \end{aligned} \tag{2.3}$$

Direktna in inverzna kinematika

Lego robota v trenutku t dobimo z integracijo kinematičnega modela, kar imenujemo **odometrija** (angl. *odometry, dead reckoning*). Določitev lege robota s podanimi vhodnimi spremenljivkami imenujemo direktna kinematika

$$\begin{aligned} x(t) &= \int_0^t v(t)\cos\varphi(t)\,\mathrm{d}t \\ y(t) &= \int_0^t v(t)\sin\varphi(t)\,\mathrm{d}t \\ \varphi(t) &= \int_0^t \omega(t)\,\mathrm{d}t \end{aligned} \tag{2.4}$$

Če med časi vzorčenja predpostavimo konstantni hitrosti v in ω, lahko integracijo v enačbah (2.4) izračunamo numerično z uporabo Eulerjeve metode. Dobimo direktno kinematiko

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_s\cos\varphi(k) \\ y(k+1) &= y(k) + v(k)T_s\sin\varphi(k) \\ \varphi(k+1) &= \varphi(k) + \omega(k)T_s \end{aligned}$$

Z uporabo trapezne integracijske metode dobimo bolj točen rezultat numerične integracije

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_s \cos\left(\varphi(k) + \frac{\omega(k)T_s}{2}\right) \\ y(k+1) &= y(k) + v(k)T_s \sin\left(\varphi(k) + \frac{\omega(k)T_s}{2}\right) \\ \varphi(k+1) &= \varphi(k) + \omega(k)T_s \end{aligned}$$

V primeru uporabe eksaktne integracije pa je direktna kinematika

$$\begin{aligned} x(k+1) &= x(k) + \frac{v(k)}{\omega(k)} \left(\sin\left(\varphi(k) + \omega(k)T_s\right) - \sin\varphi(k)\right) \\ y(k+1) &= y(k) - \frac{v(k)}{\omega(k)} \left(\cos\left(\varphi(k) + \omega(k)T_s\right) - \cos\varphi(k)\right) \\ \varphi(k+1) &= \varphi(k) + \omega(k)T_s \end{aligned}$$

kjer integriramo znotraj intervala vzorčenja in za hitrosti v in ω predvidimo sledeče spremembe stanj

$$\Delta x(k) = v(k) \int_{kT_s}^{(k+1)T_s} \cos\varphi(t)\,\mathrm{d}t = v(k) \int_{kT_s}^{(k+1)T_s} \cos\left(\varphi(k) + \omega(k)(t - kT_s)\right)\,\mathrm{d}t$$

$$\Delta y(k) = v(k) \int_{kT_s}^{(k+1)T_s} \sin\varphi(t)\,\mathrm{d}t = v(k) \int_{kT_s}^{(k+1)T_s} \sin\left(\varphi(k) + \omega(k)(t - kT_s)\right)\,\mathrm{d}t$$

Zapis inverzne kinematike je bolj zahtevna naloga, saj moramo določiti ustrezne vhode, da se bo robot peljal v želeno lego ali po želeni trajektoriji. Mobilni roboti so običajno izpostavljeni neholonomičnim omejitvam (poglavje 2.3), ki onemogočajo poljubne smeri vožnje. Obstaja tudi več možnih rešitev (poti) za doseg želene lege.

Preprosta rešitev inverzne kinematike je možna, če dovolimo le premo gibanje vozila ($v_R(t) = v_L(t) = v_R \implies \omega(t) = 0$, $v(t) = v_R$) ali le kroženje na mestu ($v_R(t) = -v_L(t) = v_R \implies \omega(t) = \frac{2v_R}{L}$, $v(t) = 0$) s konstantnimi hitrostmi. Za kroženje na mestu se enačbe gibanja (2.4) poenostavijo v

$$\begin{aligned} x(t) &= x(0) \\ y(t) &= y(0) \\ \varphi(t) &= \varphi(0) + \frac{2v_R t}{L} \end{aligned} \tag{2.5}$$

za premo gibanje pa se enačbe gibanja (2.4) poenostavijo v

$$\begin{aligned} x(t) &= x(0) + v_R t \cos\varphi(0) \\ y(t) &= y(0) + v_R t \sin\varphi(0) \\ \varphi(t) &= \varphi(0) \end{aligned} \tag{2.6}$$

Možna strategija gibanja je usmeritev vozila proti ciljni legi z rotacijo, nato sledi prema vožnja proti cilju, na koncu pa poravnava dejanske orientacije vozila

z želeno (ciljno) orientacijo. Zahtevane vhodne spremenljivke za vsako fazo (rotacija, premo gibanje, rotacija) se lahko enostavno izrazijo iz (2.5) in (2.6).

Če predpostavimo diskretno notacijo, kjer sta hitrosti $v_R(k)$ in $v_L(k)$ konstantni znotraj intervala vzorčenja T_s in se lahko spreminjata le v časovnih trenutkih $t = kT_s$, lahko zapišemo enačbe gibanja robota. Za kroženje na mestu ($v_R(k) = -v_L(k)$) imamo

$$\begin{aligned} x(k+1) &= x(k) \\ y(k+1) &= y(k) \\ \varphi(k+1) &= \varphi(k) + \frac{2v_R(k)T_s}{L} \end{aligned} \tag{2.7}$$

in za premo gibanje ($v_R(k) = v_L(k)$)

$$\begin{aligned} x(k+1) &= x(k) + v_R(k)T_s \cos\varphi(k) \\ y(k+1) &= y(k) + v_R(k)T_s \sin\varphi(k) \\ \varphi(k+1) &= \varphi(k) \end{aligned} \tag{2.8}$$

Za želeno gibanje vozila znotraj intervala vzorčenja $t \in [kT_s, (k+1)T_s)$ lahko za vsak vzorec časa izračunamo inverzno kinematiko tako, da izrazimo vhodne spremenljivke iz (2.7) in (2.8).

Kot smo že omenili, obstaja več različnih gladkih poti, ki pripeljejo vozilo v želeno lego, kar otežuje izvedbo inverzne kinematike. Inverzna kinematika pa je enostavna, če imamo predpisano želeno gladko trajektorijo $(x(t), y(t))$, ki ji mora vozilo slediti tako, da je njegova orientacija vedno tangentna na trajektorijo. Trajektorija je definirana v časovnem intervalu $t \in [0, T]$. Ob predpostavki, da je začetna lega vozila na želeni trajektoriji ter imamo idealen kinematični model, lahko izračunamo potrebne regulirne veličine (vhode) v kot

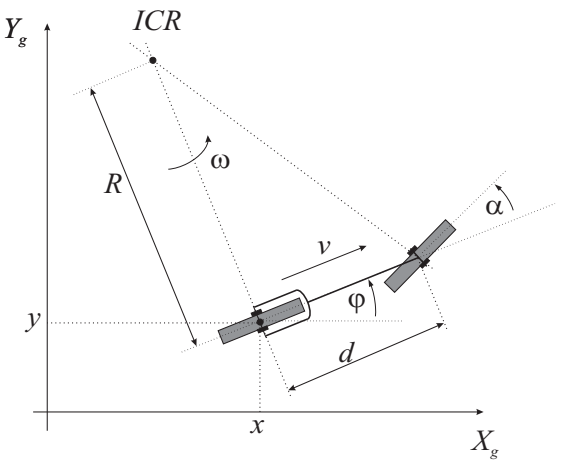
$$v(t) = \pm\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{2.9}$$

kjer predznak določa želeno smer vožnje (+ za vožnjo naprej, – za vzvratno vožnjo). Kot tangente v vsaki točki na trajektoriji je določen z

$$\varphi(t) = \operatorname{atan2}\left(\dot{y}(t), \dot{x}(t)\right) + l\pi \tag{2.10}$$

kjer $l \in \{0, 1\}$ definira želeno smer vožnje (0 za vožnjo naprej in 1 za vzvratno vožnjo). Funkcija $\operatorname{atan2}(y, x)$ je štirikvadrantna razširitev funkcije $\arctan\frac{y}{x}$

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\frac{y}{x} & ; \quad x > 0 \\ \arctan\frac{y}{x} + \pi & ; \quad x < 0 \text{ in } y \geq 0 \\ \arctan\frac{y}{x} - \pi & ; \quad x < 0 \text{ in } y < 0 \\ \frac{\pi}{2} & ; \quad x = 0 \text{ in } y > 0 \\ -\frac{\pi}{2} & ; \quad x = 0 \text{ in } y < 0 \\ \text{nedoločeno} & ; \quad x = 0 \text{ in } y = 0 \end{cases} \tag{2.11}$$



Slika 2.4: Kinematika kolesnega pogona

Z odvajanjem (2.10) po času dobimo kotno hitrost vozila $\omega(t)$

$$\omega(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{\dot{x}^2(t) + \dot{y}^2(t)} = v(t)\kappa(t) \tag{2.12}$$

kjer je $\kappa(t)$ ukrivljenost trajektorije. Z uporabo relacij (2.9) in (2.12) ter predpisane referenčne poti vozila ($x(t)$, $y(t)$) lahko izračunamo potrebni regulirni veličini $v(t)$ in $\omega(t)$. Potrebna pogoja pri načrtovanju poti sta, da je pot dvakrat odvedljiva in da je tangencialna hitrost različna od nič ($v(t) \neq 0$). Če je pri nekem času t tangencialna hitrost $v(t) = 0$, se robot vrti na mestu s krožno hitrostjo $\omega(t)$. Kota $\varphi(t)$ ne moremo določiti iz enačbe (2.9), torej mora biti podan eksplicitno. Prikazano inverzno kinematiko za znano trajektorijo lahko uporabimo pri vodenju kot predkrmiljenje, ki je dodatek povratnozančnemu vodenju za odpravo motenj, vplivov zaradi netočnega modela kinematike in začetnih pogreškov lege vozila [4].

2.2.2 Kolesni pogon

Kolesni pogon, prikazan na sliki 2.4, ima krmilno kolo s kotom krmiljenja α in se kotali s kotno hitrostjo ω_s (pogon na prednje kolo). Točka ICR je določena s presečiščem osi prednjega in zadnjega kolesa. V danem trenutku kolo kroži okoli ICR s kotno hitrostjo ω, radijem R in razdaljo med kolesoma d

$$R(t) = d \tan\left(\frac{\pi}{2} - \alpha(t)\right) = \frac{d}{\tan\alpha(t)}$$

Krmilno kolo kroži okoli ICR s kotno hitrostjo ω, zato lahko zapišemo

$$\omega(t) = \dot{\varphi}(t) = \frac{v_s(t)}{\sqrt{d^2 + R^2(t)}} = \frac{v_s(t)}{d}\sin\alpha(t)$$

kjer je $v_s(t) = \omega_s(t)r$ obodna hitrost in r radij krmilnega kolesa.

Notranja kinematika vozila (v koordinatnem sistemu robota) je določena z

$$\begin{aligned} \dot{x}_m(t) &= v_s(t)\cos\alpha(t) \\ \dot{y}_m(t) &= 0 \\ \dot{\varphi}(t) &= \frac{v_s(t)}{d}\sin\alpha(t) \end{aligned} \tag{2.13}$$

zunanja kinematika pa z

$$\begin{aligned} \dot{x}(t) &= v_s(t)\cos\alpha(t)\cos\varphi(t) \\ \dot{y}(t) &= v_s(t)\cos\alpha(t)\sin\varphi(t) \\ \dot{\varphi}(t) &= \frac{v_s(t)}{d}\sin\alpha(t) \end{aligned}$$

oziroma v matrični obliki

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \cos\varphi(t) & 0 \\ \sin\varphi(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{2.14}$$

kjer je $v(t) = v_s(t)\cos\alpha(t)$ in $\omega(t) = \frac{v_s(t)}{d}\sin\alpha(t)$.

Kolesni pogon na zadnje kolo

Običajno imajo vozila (kolo, tricikel in nekateri avtomobili) pogon na zadnja kolesa. V tem primeru sta regulirni veličini hitrost zadnjega kolesa $v_r(t)$ in kot krmiljenja sprednjega (krmilnega) kolesa $\alpha(t)$. Notranjo kinematiko lahko enostavno izpeljemo iz (2.13), kjer upoštevamo $v_r(t) = v_s(t)\cos\alpha(t)$

$$\begin{aligned} \dot{x}_m(t) &= v_r(t) \\ \dot{y}_m(t) &= 0 \\ \omega(t) = \dot{\varphi}(t) &= \frac{v_r(t)}{d}\tan\alpha(t) \end{aligned}$$

in zunanja kinematika je

$$\begin{aligned} \dot{x}(t) &= v_r(t)\cos\varphi(t) \\ \dot{y}(t) &= v_r(t)\sin\varphi(t) \\ \dot{\varphi}(t) &= \frac{v_r(t)}{d}\tan\alpha(t) \end{aligned} \tag{2.15}$$

oziroma v matrični obliki

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \cos\varphi(t) & 0 \\ \sin\varphi(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_r(t) \\ \omega(t) \end{bmatrix}$$

kjer je $\omega(t) = \frac{v_r(t)}{d}\tan\alpha(t)$.

Direktna in inverzna kinematika

Z upoštevanjem (2.14) lahko zapišemo direktno kinematiko kolesa s sprednjim pogonom z (2.4), podobno kot smo zapisali pri diferencialnem pogonu.

V splošnem je inverzno kinematiko zelo težko rešiti, lahko pa problem precej poenostavimo z vpeljavo strategije gibanja z dvema osnovnima načinoma premika. Prvi način predstavlja *premo gibanje v smeri naprej* ($\alpha(t) = 0$), drugi način pa *kroženje na mestu* ($\alpha(t) = \pm\frac{\pi}{2}$). Pri premem gibanju se hitrosti vozila poenostavijo v $v(t) = v_s(t)$ in $\omega(t) = 0$. Z vstavitvijo teh hitrosti v (2.14) in diskretizacijo dobimo sledeče enačbe gibanja

$$\begin{aligned} x(k+1) &= x(k) + v_s(k)T_s\cos\varphi(k) \\ y(k+1) &= y(k) + v_s(k)T_s\sin\varphi(k) \\ \varphi(k+1) &= \varphi(k) \end{aligned} \tag{2.16}$$

V primeru kroženja na mestu pa se hitrosti vozila poenostavijo v $v(t) = 0$ in $\omega(t) = \frac{v_s(t)}{d}$. Z vstavitvijo teh hitrosti v (2.14) in diskretizacijo dobimo sledeči model gibanja

$$\begin{aligned} x(k+1) &= x(k) \\ y(k+1) &= y(k) \\ \varphi(k+1) &= \varphi(k) + \frac{v_s(t)}{d}T_s \end{aligned} \tag{2.17}$$

Regulirni veličini (vhoda v sistem) lahko določimo iz (2.16) in (2.17) za želeno gibanje med časi vzorčenja.

2.2.3 Trikolesni pogon

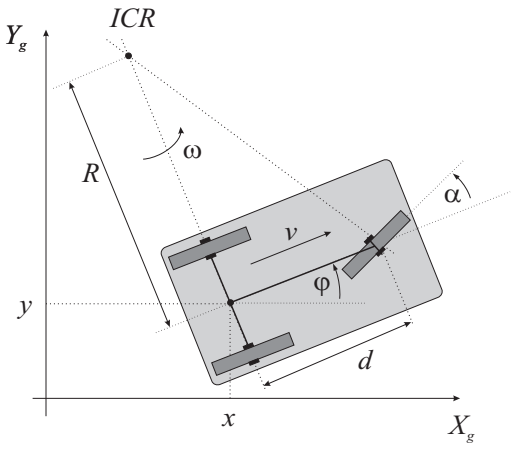
Trikolesni pogon, prikazan na sliki 2.5, ima enako kinematiko kot kolesni pogon

$$\begin{aligned} \dot{x}(t) &= v_s(t)\cos\alpha(t)\cos\varphi(t) \\ \dot{y}(t) &= v_s(t)\cos\alpha(t)\sin\varphi(t) \\ \dot{\varphi}(t) &= \frac{v_s(t)}{d}\sin\alpha(t) \end{aligned} \tag{2.18}$$

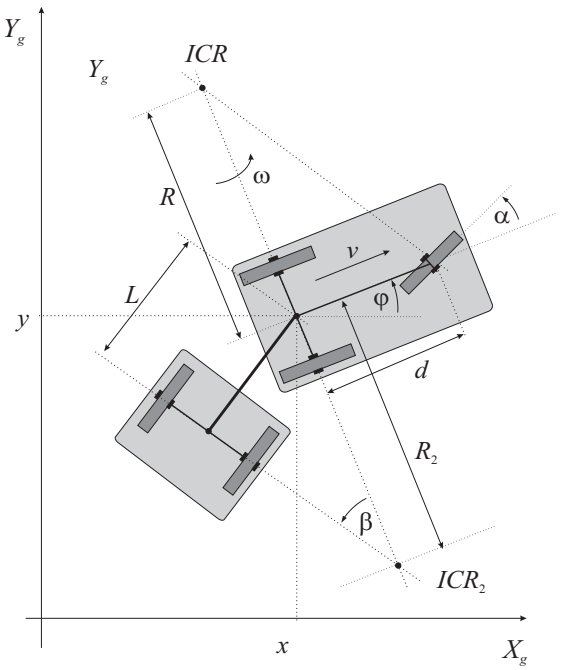
kjer velja $v(t) = v_s(t)\cos\alpha(t)$, $\omega(t) = \frac{v_s(t)}{d}\sin\alpha(t)$ in v_s je obodna hitrost krmilnega kolesa. Trikolesni pogon je pogosto uporabljen v mobilni robotiki, ker tri kolesa zagotavljajo stabilnost vozila v vertikalni smeri in tako pomožna podporna kolesa niso potrebna.

2.2.4 Tricikel s priklopnikom

Kinematika tricikla je opisana v poglavju 2.2.3. Za priklopnik določimo točko



Slika 2.5: Kinematika trikolesnega pogona



Slika 2.6: Kinematika tricikla s priklopnikom

ICR_2, ki leži na presečišču zadnje osi tricikla in osi priklopnika. Kotna hitrost, s katero kolesa priklopnika krožijo okoli točke ICR_2, je

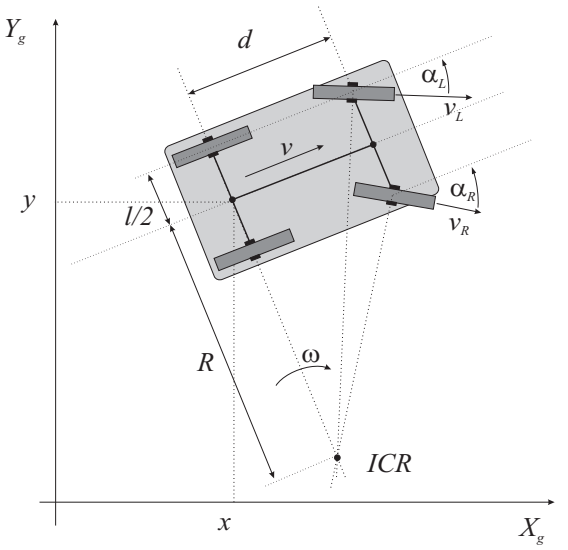
$$\omega_2(t) = \frac{v(t)}{R_2(t)} = \frac{v_s(t)\cos\alpha(t)}{R_2(t)} = \frac{v_s(t)\cos\alpha(t)\sin\beta(t)}{L} = \dot{\beta}(t)$$

in končna kinematika vozila na sliki 2.6 je določena z

$$\begin{aligned}
\dot{x}(t) &= v_s(t)\cos\alpha(t)\cos\varphi(t) \\
\dot{y}(t) &= v_s(t)\cos\alpha(t)\sin\varphi(t) \\
\dot{\varphi}(t) &= \frac{v_s(t)}{d}\sin\alpha(t) \\
\dot{\beta}(t) &= \frac{v_s(t)\cos\alpha(t)\sin\beta(t)}{L}
\end{aligned}$$

2.2.5 Avtomobilski (Ackermannov) pogon

Avtomobilski pogon uporablja Ackermannov princip krmiljenja, čigar osnovna ideja je, da ima notranje kolo (tisto, ki je bližje točki ICR) večji zasuk krmiljenja kot zunanje. To omogoča vozilu, da se vrti okoli središčne točke na osi zadnjih koles. Posledično ima notranje kolo manjšo obodno hitrost kot zunanje. Ackermannovo krmiljenje omogoča vrtenje zadnjih koles brez zdrsov, zato točka ICR leži na premici, ki gre skozi zadnjo os. Ta krmilni mehanizem omogoča manjšo obrabo pnevmatik. Na sliki 2.7 je levo kolo na zunanji strani, desno pa



Slika 2.7: Shema Ackermannovega pogona

na notranji. Orientacijo prednjih krmilnih koles lahko določimo iz

$$\begin{aligned}
\tan\left(\frac{\pi}{2}-\alpha_L\right) &= \frac{R+\frac{l}{2}}{d} \\
\tan\left(\frac{\pi}{2}-\alpha_R\right) &= \frac{R-\frac{l}{2}}{d}
\end{aligned}$$

od koder izrazimo kote krmiljenja

$$\alpha_L = \frac{\pi}{2} - \arctan\frac{R+\frac{l}{2}}{d}$$
$$\alpha_R = \frac{\pi}{2} - \arctan\frac{R-\frac{l}{2}}{d}$$

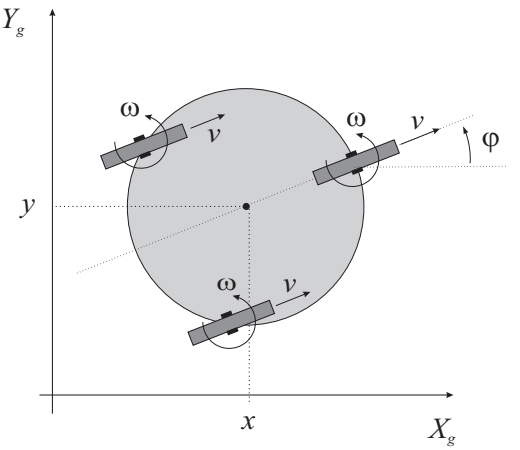
Notranje in zunanje zadnje kolo krožita okoli točke ICR z enako kotno hitrostjo ω, torej sta njuni obodni hitrosti

$$v_L = \omega\left(R+\frac{l}{2}\right)$$
$$v_R = \omega\left(R-\frac{l}{2}\right)$$

Ackermannov kinematični pogon je primeren za modeliranje gibanja večjih vozil. Model gibanja lahko opišemo tudi z uporabo kinematike tricikla (2.18), kjer uporabimo povprečen Ackermannov kot krmiljenja $\alpha = \frac{\pi}{2} - \arctan\frac{R}{d}$. Inverzna kinematika Ackermannovega pogona je zahtevna in presega namen tega dela.

2.2.6 Sinhroni pogon

Vozilo s sinhronim pogonom lahko vsa svoja kolesa sinhrono krmili okoli vertikalne osi (v danem trenutku imajo vsa kolesa enako orientacijo). Tipično ima vozilo s sinhronim pogonom tri kolesa, ki so razporejena simetrično (v enakostraničnem trikotniku) okoli središča vozila, kot je prikazano na sliki 2.8. Vsa kolesa so



Slika 2.8: Sinhroni pogon

krmiljena sinhrono, torej so njihove osi vrtenja zmeraj vzporedne in zato se točka ICR nahaja v neskončnosti. Vozilo lahko neposredno spreminja orientacijo koles, kar predstavlja tretje stanje v vektorju stanj (2.19). Regulirne veličine so hitrost krmiljenja koles ω in njihova obodna hitrost v.

Kinemtika vozila s sinhronim pogonom je podobna kinematiki difencialnega pogona

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \cos\varphi(t) & 0 \\ \sin\varphi(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{2.19}$$

kjer sta $v(t)$ in $\omega(t)$ regulirni spremenljivki ali vhoda, ki ju lahko neodvisno spreminjamo (to pri diferencialnem pogonu ni možno).

Direktna in inverzna kinematika

Direktno kinematiko dobimo z integracijo kinematičnega modela (2.19).

$$x(t) = \int_0^t v(t)\cos\varphi(t)\,\mathrm{d}t$$

$$y(t) = \int_0^t v(t)\sin\varphi(t\,\mathrm{d}t$$

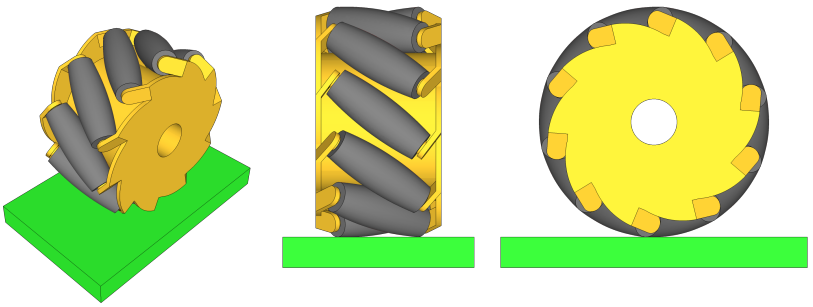
$$\varphi(t) = \int_0^t \omega(t)\,\mathrm{d}t$$

Splošna rešitev inverzne kinematike ni možna, ker obstaja več rešitev za doseg želene lege. Inverzna kinematika je enostavno rešljiva v posebnem primeru, kjer se vozilo vrti na mestu ali pa premo giblje v smeri trenutne orientacije (brez rotacije). Ko se robot določen časovni interval Δt vrti na mestu s konstantno krožno hitrostjo ω, se njegova orientacija spremeni za $\omega\Delta t$. V primeru premega gibanja s konstantno hitrostjo v, ki traja Δt, se vozilo premakne za $v\Delta t$ v smeri trenutne orientacije.

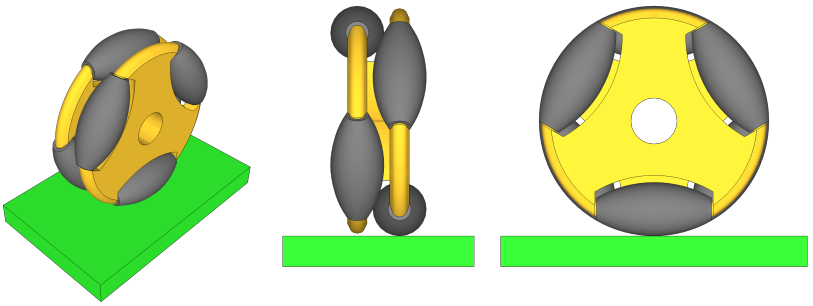
2.2.7 Večsmerni pogon

V predhodno opisanih kinematičnih modelih so bila uporabljena preprosta kolesa, ki se lahko vrtijo (kotalijo) le v smeri njihove orientacije (npr. diferencialni pogon). Tovrstna preprosta kolesa imajo samo eno možno smer kotaljenja. Da omogočimo večsmerno kotaljenje, potrebujemo bolj kompleksno konstrukcijo koles. Primer takega kolesa je kolo *Mecanum* ali *švedsko kolo* (slika 2.9), ki ima po obodu razvrščenih več valjčkov. Osi pasivnih valjčkov niso vzporedne z osjo glavnega kolesa, ampak so običajno pod kotom $\gamma = 45°$. To omogoča različne smeri gibanja, ki izhajajo iz poljubne kombinacije smeri vrtenja glavnega kolesa in pasivnih valjčkov.

Drug primer kompleksnega kolesa je kolo *omni*, ki omogoča večsmerno gibanje, podobno kot kolo Mecanum. Kolo omni (slika 2.10) ima na svojem obodu nameščene pasivne valjčke, katerih os je pravokotna glede na os kolesa.



Slika 2.9: Kolo Mecanum z valjčki, nameščenimi po obodu kolesa. Vsak valjček ima os vrtenja pod kotom 45° glede na ravnino koles in pod kotom 45° glede na linijo, vzporedno z osjo kolesa.

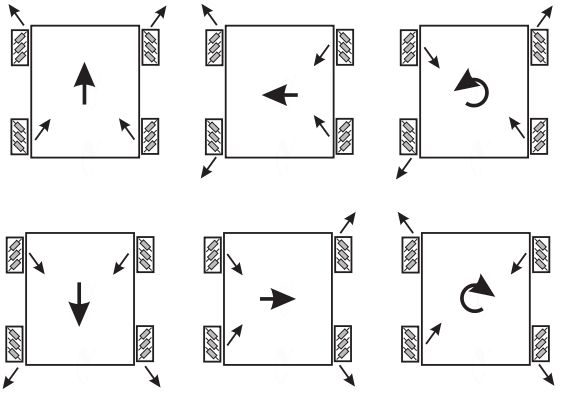


Slika 2.10: Kolo omni s šestimi "prostovrtečimi" se valjčki, ki so razporejeni po obodu kolesa. Kolo se lahko vrti in bočno drsi (vrtijo se valjčki).

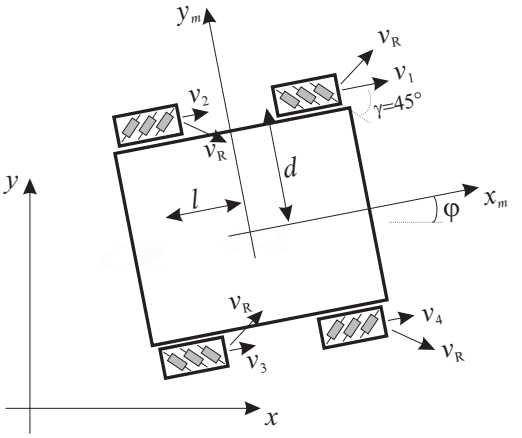
Kinematika štirikolesnega večsmernega pogona

Priljubljena štirikolesna *platforma Mecanum*, prikazana na sliki 2.11, ima kolesa z levo in desnosučnimi valjčki, kjer sta diagonalni kolesi istega tipa. To omogoča vozilu, da se premika v poljubni smeri s poljubno rotacijo, kar dosežemo s spreminjanjem hitrosti in smeri vrtenja glavnih koles. Če se vsa štiri kolesa vrtijo v isti smeri (z isto hitrostjo), se vozilo giblje naprej ali vzvratno. Ko pa se glavni kolesi na eni strani platforme vrtijo v nasprotni smeri kot kolesi na drugi strani platforme, bo le-ta krožila. Bočno gibanje platforme dosežemo tako, da se kolesi na eni diagonali vrtijo v nasprotni smeri kot kolesi na drugi diagonali. Kombinacija opisanih gibanj omogoča gibanje platforme v poljubni smeri s poljubno rotacijo.

Inverzno notranjo kinematiko štirikolesnega pogona Mecanum na sliki 2.12 lahko zapišemo na naslednji način. Hitrost sprednjega kolesa (v koordinatnem sistemu robota) pridobimo iz hitrosti glavnega kolesa $v_1(t)$ in hitrosti pasivnih valjčkov $v_R(t)$. V nadaljevanju bomo izpustili zapis s časovno odvisnostjo, da dobimo bolj kompaktne in enostavne enačbe (npr. $v_1(t) = v_1$). Skupna hitrost koles v smereh



Slika 2.11: Osnovne smeri gibanja večsmernega pogona s štirimi kolesi Mecanum, ki se lahko vrtijo naprej ali nazaj



Slika 2.12: Štirikolesna platforma Mecanum

x_m in y_m v koordinatnem sistemu robota je $v_{m1x} = v_1 + v_R \cos\frac{\pi}{4} = v_1 + \frac{v_R}{\sqrt{2}}$ in $v_{m1y} = v_R \sin\frac{\pi}{4} = \frac{v_R}{\sqrt{2}}$, od koder pridobimo hitrost (sprednjega) glavnega kolesa $v_1 = v_{m1x} - v_{m1y}$. Hitrost sprednjega kolesa v koordinatnem sistemu robota lahko izrazimo tudi s translacijsko hitrostjo robota $v_m = \sqrt{\dot{x}_m^2 + \dot{y}_m^2}$ in njegovo kotno hitrostjo $\dot{\varphi}$ kot $v_{m1x} = \dot{x}_m - \dot{\varphi}d$ in $v_{m1y} = \dot{y}_m + \dot{\varphi}l$ (pomen razdalj d in l je mogoče razbrati iz slike 2.12). Iz slednjih relacij lahko izrazimo hitrost glavnega kolesa s hitrostjo robota kot $v_1 = \dot{x}_m - \dot{y}_m - (l+d)\dot{\varphi}$. Podobne enačbe lahko zapišemo za v_2, v_3 in v_4. Torej je inverzna kinematika v lokalnih koordinatah

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -l-d \\ 1 & 1 & -l-d \\ 1 & -1 & l+d \\ 1 & 1 & l+d \end{bmatrix} \begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\varphi} \end{bmatrix} \tag{2.20}$$

ki jo lahko zapišemo v matrični obliki kot $\boldsymbol{v} = \boldsymbol{J}\dot{\boldsymbol{q}}_m$, kjer je $\boldsymbol{v}^T = [v_1,\ v_2,\ v_3,\ v_4]^T$ in $\boldsymbol{q}_m^T = [x_m,\ y_m,\ \varphi]^T$.

Za izračun inverzne kinematike v globalnih koordinatah moramo obravnavati

rotacijsko matriko $\boldsymbol{R}_G^L$, ki predstavlja orientacijo lokalnih koordinat glede na globalne ($\boldsymbol{q}_m = \boldsymbol{R}_G^L \boldsymbol{q}$)

$$\boldsymbol{R}_G^L = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

in jo upoštevamo kot $\boldsymbol{v} = \boldsymbol{J}\boldsymbol{R}_G^L\dot{\boldsymbol{q}}$.

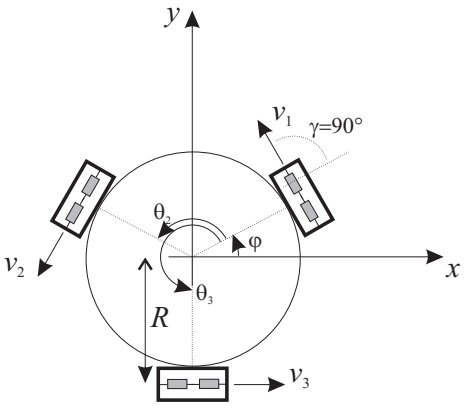
Iz notranje inverzne kinematike $\boldsymbol{v} = \boldsymbol{J}\dot{\boldsymbol{q}}_m$ (2.20) dobimo direktno notranjo kinematiko $\dot{\boldsymbol{q}}_m = \boldsymbol{J}^+\boldsymbol{v}$, kjer je $\boldsymbol{J}^+ = \left(\boldsymbol{J}^T\boldsymbol{J}\right)^{-1}\boldsymbol{J}^T$ psevdoinverz matrike $\boldsymbol{J}$. Direktno notranjo kinematiko platforme Mecanum s štirimi kolesi zapišemo kot

$$\begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\varphi} \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ \frac{-1}{(l+d)} & \frac{-1}{(l+d)} & \frac{1}{(l+d)} & \frac{1}{(l+d)} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

Direktno kinematko v globalnih koordinatah pa dobimo z $\dot{\boldsymbol{q}} = \left(\boldsymbol{R}_G^L\right)^T\boldsymbol{J}^+\boldsymbol{v}$.

Kinematika trikolesnega večsmernega pogona

Na sliki 2.13 je prikazana priljubljena večsmerna konfiguracija za trikolesni pogon. Njegovo inverzno kinematiko (v globalnih koordinatah) dobimo z upoštevanjem



Slika 2.13: Trikolesni večsmerni pogon ($\theta_2 = 120°$, $\theta_3 = 240°$)

translacijske hitrosti robota $v = \sqrt{\dot{x}^2 + \dot{y}^2}$ in njegove kotne hitrosti $\dot{\varphi}$. Hitrost prvega kolesa $v_1 = v_{1t} + v_{1r}$ je sestavljena iz translacije $v_{1t} = -\dot{x}\sin\varphi + \dot{y}\cos\varphi$ in orientacije $v_{1r} = R\dot{\varphi}$. Torej je skupna hitrost prvega kolesa enaka $v_1 = -\dot{x}\sin\varphi + \dot{y}\cos\varphi + R\dot{\varphi}$. Podobno je ob upoštevanju kota (v globalnih koordinatah) drugega kolesa $\varphi+\theta_2$ njegova hitrost enaka $v_2 = -\dot{x}\sin(\varphi+\theta_2) + \dot{y}\cos(\varphi+\theta_2) + R\dot{\varphi}$; hitrost tretjega kolesa pa je $v_3 = -\dot{x}\sin(\varphi+\theta_3) + \dot{y}\cos(\varphi+\theta_3) + R\dot{\varphi}$. Inverzna

kinematika trikolesnega pogona v globalnih koordinatah je

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -\sin\varphi & \cos\varphi & R \\ -\sin(\varphi+\theta_2) & \cos(\varphi+\theta_2) & R \\ -\sin(\varphi+\theta_3) & \cos(\varphi+\theta_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} \tag{2.21}$$

kar zapišemo v matrični obliki kot $\boldsymbol{v} = \boldsymbol{J}\dot{\boldsymbol{q}}$. Včasih je bolj priročno voditi robota v njegovih lokalnih koordinatah, ki jih pridobimo z upoštevanjem transformacije rotacije $\boldsymbol{v} = \boldsymbol{J}\left(\boldsymbol{R}_G^L\right)^T \dot{\boldsymbol{q}}_m$

Direktno kinematiko v globalnih koordinatah dobimo s pomočjo inverzne kinematike (2.21) kot $\dot{\boldsymbol{q}} = \boldsymbol{S}\boldsymbol{v}$, kjer je $\boldsymbol{S} = \boldsymbol{J}^{-1}$

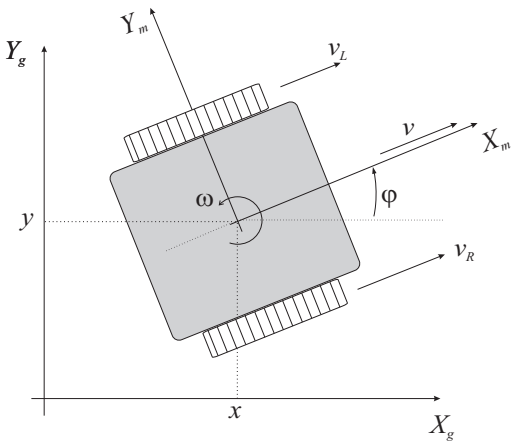
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} -\sin\theta_1 & -\sin(\theta_1+\theta_2) & -\sin(\theta_1+\theta_3) \\ \cos\theta_1 & \cos(\theta_1+\theta_2) & \cos(\theta_1+\theta_3) \\ \frac{1}{2R} & \frac{1}{2R} & \frac{1}{2R} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

2.2.8 Gosenični pogon

Gibanje goseničnega pogona (slika 2.14) lahko približno opišemo s kinematiko diferencialnega pogona

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \cos\varphi(t) & 0 \\ \sin\varphi(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

Diferencialni pogon predpostavlja idealno kotaljenje koles s točkastim dotikom kolesa in podlage, kar pa ne drži v primeru goseničnega pogona. Gosenični pogon ima večjo kontaktno površino med kolesi in tlemi, torej morajo za spremembo smeri gibanja gosenice (oz. kolesa) drseti. To jim omogoča premikanje po zahtevnejšem terenu, kjer so ostala kolesna vozila manj uspešna. Koeficient zdrsa



Slika 2.14: Gosenični pogon

med gosenicami in podlago ni konstanten, saj je odvisen od kontakta s tlemi oz.

vrste podlage. Zato je odometrija (direktna kinematika) še manj zanesljiva za ocenjevanje lege robota v primerjavi z diferencialnim pogonom.

2.3 Omejitve gibanja

Pri gibanju kolesnega mobilnega robota se srečujemo z dinamičnimi in kinematičnimi omejitvami. **Dinamične omejitve** izvirajo iz dinamičnega modela sistema, ki ima omejeno odzivnost (pospeševanje) zaradi svoje vztrajnosti (mase) in omejitev motornega pogona (npr. omejen navor motorja zaradi njegovih zmogljivosti ali preprečevanja podrsavanja koles). **Kinematične omejitve** pa izvirajo iz konstrukcije robota in njegovega kinematičnega modela. Zanimive so predvsem kinematične omejitve, ki jih ločimo na holonomične in neholonomične omejitve. **Neholonomične omejitve** omejujejo možne smeri premika mobilnega robota [5]. **Holonomične omejitve** pa se nanašajo le na dimenzijo opisa sistema s posplošenimi koordinatami, zato lahko z njihovo pomočjo odstranimo odvečne posplošene koordinate, ki so odvisne od drugih.

Nek sistem je holonomičen, če nima kinematičnih omejitev ali pa vsebuje samo holonomične omejitve, zato nima omejitev v smeri gibanja. Neholonomični sistem pa vsebuje neholonomične omejitve, torej se ne more premikati v poljubni smeri (npr. avtomobil se lahko premika le v smeri vrtenja koles, ne more pa se premikati bočno). Za holonomične sisteme lahko določimo nabor neodvisnih posplošenih koordinat, ki določajo prostor, v katerem so možne poljubne smeri gibanja. V neholonomičnih sistemih temu ni tako, saj gibanje v vsakem trenutku ni poljubno, temveč je dovoljeno le gibanje, ki ustreza neholonomičnim omejitvam. Za holonomične sisteme torej velja, da so njihova stanja neposredno odvisna od konfiguracije notranjih spremenljivk (zasuki koles, koti sklepov). V primeru neholonomičnih sistemov to ne drži, saj vrnitev notranjih spremenljivk v začetno konfiguracijo ne zagotavlja tudi vrnitve sistema v začetno stanje (pozicijo in orientacijo). Posplošeno lahko rečemo, da je izhodno stanje neholonomičnih sistemov odvisno od opravljene poti (zaporedje notranjih spremenljivk).

V nadaljevanju bomo obravnavali mehanske sisteme, katerih konfiguracijo (lega sistema v okolju in odnosi med deli sistema) lahko opišemo z vektorjem posplošenih koordinat $\boldsymbol{q}$. Pri podani trajektoriji $\boldsymbol{q}(t)$ določimo vektor posplošenih hitrosti $\dot{\boldsymbol{q}}(t)$.

Holonomične omejitve izrazimo v obliki enačb, ki povezujejo posplošene koordinate. Te enačbe lahko uporabimo za izločitev nekaterih posplošenih spremenljivk, da dobimo manjši prostor posplošenih spremenljivk, potrebnih za opis sistema. Neholonomične omejitve pa ne zmanjšujejo dimenzije prostora posplošenih spremenljivk temveč samo dimenzije prostora možnih posplošenih hitrosti. Neholonomične omejitve torej vplivajo na problem načrtovanja poti. V zvezi z njimi se pojavljajo naslednja vprašanja:

- *Kako ugotoviti, ali je kinematična omejitev neholonomična?* Če je omejitev integrabilna, se lahko enačba, ki vsebuje hitrostne parametre (odvode posplošenih koordinat), prevede v holonomično omejitev.
- *Ali neholonomična omejitev omejuje množico dostopnih konfiguracij (tj. lege sistema)?* Z uporabo orodij teorije vodenja lahko pridemo do preprostih pogojev, pod katerimi neholonomične omejitve ne vplivajo na območje dosegljivih leg.
- *Kako zgraditi generator izvedljivih oz. možnih poti za robota z neholonomičnimi omejitvami?*

2.3.1 Holonomične omejitve

Holonomične omejitve so vezane na posplošene koordinate (stanja) sistema. Za sistem z n posplošenimi koordinatami $\boldsymbol{q} = [q_1, \ldots, q_n]^T$ je holonomična omejitev izražena kot

$$f(\boldsymbol{q}) = f(q_1, \ldots, q_n) = 0 \tag{2.22}$$

kjer je f gladka funkcija z zveznimi odvodi. Ta omejitev določa podmnožico vseh možnih konfiguracij v posplošenih koordinatah (delovni prostor), ki zadostujejo omejitvi (2.22) (zmanjša število prostostnih stopenj sistema). Z upoštevanjem (2.22) lahko namreč izločimo določeno posplošeno koordinato (izrazimo jo lahko z $n-1$ ostalimi koordinatami).

V splošnem imamo lahko m holonomičnih omejitev ($m < n$). Če so omejitve linearno neodvisne, določajo $(n-m)$–dimenzionalni "podprostor", ki je dejanski delovni prostor sistema (sistem ima $n-m$ prostostnih stopenj).

2.3.2 Neholonomične omejitve

Neholonomične omejitve omejujejo možne hitrosti ali smeri gibanja sistema. Zapišemo jih v obliki

$$f(\boldsymbol{q}, \dot{\boldsymbol{q}}) = f(q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n) = 0 \tag{2.23}$$

kjer je f gladka funkcija z zveznimi odvodi in $\dot{\boldsymbol{q}}$ vektor hitrosti sistema v posplošenih koordinatah. V primeru, da sistem nima omejitev (2.23), se lahko giblje v poljubnih smereh.

Kinematična omejitev (2.23) je holonomična, če je integrabilna, kar pomeni, da lahko hitrosti $\dot{q}_1, \ldots, \dot{q}_n$ izločimo iz enačbe (2.23) in zapišemo omejitev v obliki (2.22). Če omejitev (2.23) ni integrabilna, je neholonomična.

Če obstaja m linearno neodvisnih neholonomičnih omejitev v obliki (2.23), je prostor dostopnih hitrosti $(n-m)$–dimenzionalen. Neholonomična omejitev torej omeji dovoljene hitrosti sistema. Za primer lahko vzamemo dvokolesnega robota

(invalidski voziček), ki se lahko premika v smeri trenutne orientacije koles, v bočni smeri (pravokotno na kolesa) pa ne.

Predpostavimo, da so omejitve linearne v odvisnosti od $\dot{\boldsymbol{q}} = [\dot{q}_1, \ldots, \dot{q}_n)]^T$. Potem lahko (2.23) zapišemo kot

$$f(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{a}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = \begin{bmatrix} a_1(\boldsymbol{q}) & \ldots & a_n(\boldsymbol{q}) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix} = 0$$

kjer je $\boldsymbol{a}(\boldsymbol{q})$ vektor členov omejitve (nedovoljena smer pomika). V kolikor imamo m neholonomičnih omejitev, lahko njihove člene zapišemo v matriko omejitev

$$\boldsymbol{A}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{a}_1^T(\boldsymbol{q}) \\ \vdots \\ \boldsymbol{a}_m^T(\boldsymbol{q}) \end{bmatrix}$$

in vse neholonomične omejitve sistema podamo v matrični obliki

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

Nadalje določimo matriko dosegljivih smeri gibanja sistema (m omejitev določa $n - m$ dosegljivih smeri)

$$\boldsymbol{S}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{s}_1(\boldsymbol{q}) & \boldsymbol{s}_2(\boldsymbol{q}) & \ldots & \boldsymbol{s}_{n-m}(\boldsymbol{q}) \end{bmatrix}$$

Ta matrika podaja kinematični model sistema, za katerega velja

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{v}(t) \tag{2.24}$$

kjer je $\boldsymbol{v}(t)$ regulirni vektor (glejte kinamatični model (2.2)). Produkt matrike omejitev $\boldsymbol{A}$ in kinematične matrike $\boldsymbol{S}$ je ničelna matrika

$$\boldsymbol{AS} = \boldsymbol{0}$$

2.3.3 Integrabilnost omejitev

Da ugotovimo, ali je določena omejitev neholonomična (oz. hitrostna), moramo preveriti, če jo je mogoče integrirati in s tem prevesti v holonomično omejitev. V kolikor to ni mogoče, je omejitev neholonomična.

2.3.4 Vektorska polja, porazdelitev, Liejevi oklepaji

V trenutnem času t in stanju $\boldsymbol{q}$ dobimo možne smeri premikov iz določene točke prostora $\boldsymbol{q}$ z linearno kombinacijo vektorskih polj v matriki dosegljivih smeri $\boldsymbol{S}$.

Porazdelitev tako podaja dosegljiv "podprostor" iz določene točke prostora $\boldsymbol{q}$ s premiki, ki predstavljajo linearno kombinacijo vektorskih polj (stolpci matrike $\boldsymbol{S}$). **Vektorska polja** so odvodi posplošenih koordinat, torej predstavljajo hitrosti ali smeri možnih premikov v prostoru. Vektorsko polje je zvezno odvedljiva preslikava, ki vsaki točki prostora priredi natanko določen vektor. Prikaz določitve dosegljivih vektorskih polj je podan v primeru 2.1.

Primer 2.1

Za robota z diferencialnim pogonom s kinematičnim modelom (2.2) določite dosegljive hitrosti (smeri premikov) in omejitve gibanja.

Rešitev

Vektorski polji dosegljivih hitrosti (smeri premikov) sta

$$\boldsymbol{s}_1(\boldsymbol{q}) = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} \qquad \boldsymbol{s}_2(\boldsymbol{q}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.25}$$

kar pomeni, da so možne smeri premika v danem trenutku, ko se nahajamo v legi $\boldsymbol{q}$, podane z linearno kombinacijo

$$\dot{\boldsymbol{q}} = u_1\boldsymbol{s}_1(\boldsymbol{q}) + u_2\boldsymbol{s}_2(\boldsymbol{q}) \tag{2.26}$$

kjer sta u_1 in u_2 poljubni realni števili, ki predstavljata regulirni veličini. Enačba (2.26) je le preurejen zapis kinematičnega modela (2.2).

V kolikor vektorskih polj $\boldsymbol{s}_i$ nimamo podanih, jih lahko določimo iz znanih omejitev $\boldsymbol{a}_j$, kjer upoštevamo ortogonalnost smeri omejitev in smeri gibanja, torej $\boldsymbol{s}_i \perp \boldsymbol{a}_j$. Iz slike 2.3 lahko določimo omejitev s smerjo, v kateri se robot ne more premikati, to je bočno na kolesa. Edina omejitev je

$$\boldsymbol{a}(\boldsymbol{q}) = \begin{bmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{bmatrix}$$

kar je ravno pravokotno na vektor možnega premika $\boldsymbol{s}_1(\boldsymbol{q})$ (translacijski premik v smeri kotaljenja koles) in na vektor $\boldsymbol{s}_2(\boldsymbol{q})$ (rotacija okoli osi pravokotne na ravnino gibanja).

V kolikor porazdelitev določenih vektorskih polj definira celoten prostor, je **vsebovana** (angl. *involutive*). Če osnovna porazdelitev ni vsebovana, lahko določimo množico novih vektorskih polj, ki so linearno neodvisna od vektorskih polj osnovne porazdelitve. Nova vektorska polja (smeri pomika) lahko dobimo s končnim številom preklopov osnovnih smeri pomika, kjer dolžino pomikov

limitiramo proti 0. Nove smeri lahko določimo s pomočjo *Liejevih oklepajev*, ki predstavljajo operacijo nad dvema vektorskima poljema, kot bo kasneje podano z enačbo (2.28). Praktičen primer uporabe je paralelno parkiranje avtomobila (tudi vozila z diferencialnim pogonom), kjer ni možen neposreden bočni premik na parkirno mesto zaradi neholonomičnih omejitev sistema (kolesa ne drsijo bočno). Kljub temu pa lahko dosežemo premik v stran z zaporedno kombinacijo gibanja naprej in nazaj ter zasukov, kar prikazuje primer 2.2.

Primer 2.2

Predstavite manever za paralelno parkiranje vozila z diferencialnim pogonom.

Rešitev

Z uporabo osnovnih smeri pomika, definiranih z vektorskimi polji $\boldsymbol{s}_1(\boldsymbol{q})$ in $\boldsymbol{s}_2(\boldsymbol{q})$ (glejte enačbo (2.25)), in začetnim stanjem (lega vozila) $\boldsymbol{q}_0 = \boldsymbol{q}(0)$ določimo novo stanje sistema. Začnemo v stanju $\boldsymbol{q}_0 = \boldsymbol{q}(0)$ in se za kratek čas ε gibljemo v smeri $\boldsymbol{s}_1$, nato v smeri $\boldsymbol{s}_2$ za čas ε, nato v smeri $-\boldsymbol{s}_1$ za čas ε in na koncu v smeri $-\boldsymbol{s}_2$ za čas ε, kjer je dosežena končna lega vozila. To lahko matematično zapišemo kot

$$\boldsymbol{q}(4\varepsilon) = \phi_{\varepsilon}^{-\boldsymbol{s}_2}\left(\phi_{\varepsilon}^{-\boldsymbol{s}_1}\left(\phi_{\varepsilon}^{\boldsymbol{s}_2}\left(\phi_{\varepsilon}^{\boldsymbol{s}_1}\left(\boldsymbol{q}_0\right)\right)\right)\right)$$

kar predstavlja nelinearno diferencialno enačbo, katere rešitev (integracija kinematičnega modela) lahko aproksimiramo z razvojem v Taylerjevo vrsto (za podrobnosti glejte [1]) kot

$$\boldsymbol{q}(4\varepsilon) = \boldsymbol{q}_0 + \varepsilon^2\left(\frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{q}}\boldsymbol{s}_1(\boldsymbol{q}_0) - \frac{\partial \boldsymbol{s}_1}{\partial \boldsymbol{q}}\boldsymbol{s}_2(\boldsymbol{q}_0)\right) + O(\varepsilon^3) \tag{2.27}$$

kjer so parcialni odvodi ovrednoteni v $\boldsymbol{q}_0$, $O(\varepsilon^3)$ pa predstavlja prispevek višjih odvodov, ki je za kratke čase ε zanemarljiv. Dobljeni končni premik manevra parkiranja je tako dolžine ε^2 v smeri vektorja, ki je podan v oklepaju in predstavlja operacijo Liejev oklepaj, definiran v enačbi (2.28).

Manever paralelnega parkiranja lahko predstavimo z eksperimentom. Predpostavimo, da je začetna lega robota $\boldsymbol{q}_0 = [0,\ 0,\ 0]^T$. V prvem koraku izvajanja manevra pridemo v točko

$$\boldsymbol{q}_1 = \boldsymbol{q}_0 + \varepsilon\boldsymbol{s}_1 = \begin{bmatrix}0\\0\\0\end{bmatrix} + \varepsilon\begin{bmatrix}\cos 0\\ \sin 0\\ 0\end{bmatrix} = \begin{bmatrix}\varepsilon\\0\\0\end{bmatrix}$$

v drugem koraku izvedemo rotacijo

$$\boldsymbol{q}_2 = \boldsymbol{q}_1 + \varepsilon\boldsymbol{s}_2 = \begin{bmatrix}\varepsilon\\0\\0\end{bmatrix} + \varepsilon\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}\varepsilon\\0\\\varepsilon\end{bmatrix}$$

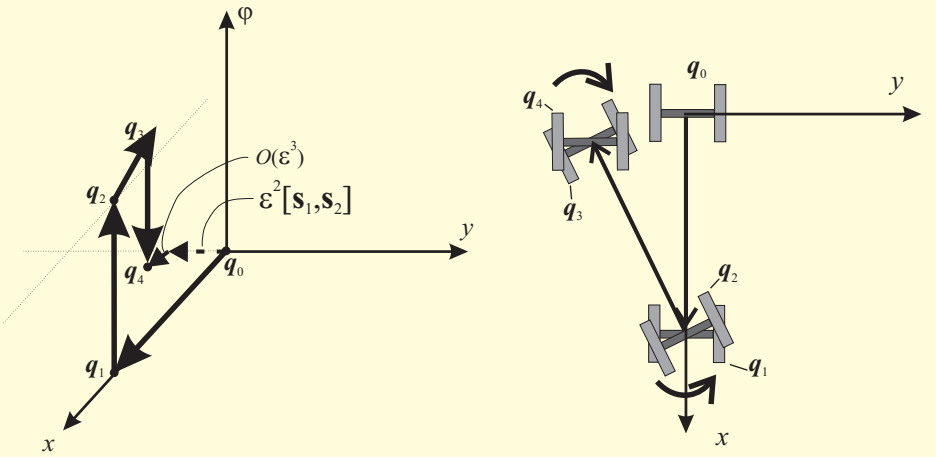
v tretjem koraku imamo translacijo v negativni smeri

$$\boldsymbol{q}_3 = \boldsymbol{q}_2 - \varepsilon \boldsymbol{s}_1 = \begin{bmatrix} \varepsilon \\ 0 \\ \varepsilon \end{bmatrix} - \varepsilon \begin{bmatrix} \cos\varepsilon \\ \sin\varepsilon \\ 0 \end{bmatrix} = \begin{bmatrix} \varepsilon - \varepsilon\cos\varepsilon \\ -\varepsilon\sin\varepsilon \\ \varepsilon \end{bmatrix}$$

in v zadnjem koraku rotacijo v negativni smeri

$$\boldsymbol{q}_4 = \boldsymbol{q}_3 - \varepsilon \boldsymbol{s}_2 = \begin{bmatrix} \varepsilon - \varepsilon\cos\varepsilon \\ -\varepsilon\sin\varepsilon \\ \varepsilon \end{bmatrix} - \varepsilon \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \varepsilon - \varepsilon\cos\varepsilon \\ -\varepsilon\sin\varepsilon \\ 0 \end{bmatrix}$$

Na sliki 2.15 so prikazani izračunani premiki in vmesne točke.



Slika 2.15: Manever paralelnega parkiranja v delovnem prostoru (levo), kjer je orientacija predstavljena z osjo z, in pogled na manever od zgoraj (desno)

Opazimo lahko, da končni premik v stran ni neposredno izvedljiv (v enem koraku) z možnimi smermi gibanja $\boldsymbol{s}_1$ in $\boldsymbol{s}_2$, je pa izvedljiv v več korakih z njuno kombinacijo. Dobljen končni premik ni točno v bočni smeri zaradi končnega časa ε in člena $O(\varepsilon^3)$ v relaciji (2.27). V kolikor so premiki majhni s kratkim časom trajanja $\varepsilon \to 0$, je dobljen premik točno v stran, torej je končna lega

$$\boldsymbol{q}_4 = \begin{bmatrix} 0 \\ -\varepsilon^2 \\ 0 \end{bmatrix}$$

Kot smo videli v primeru 2.2, lahko s pomočjo Liejevih oklepajev določimo nove smeri gibanja, ki jih osnovna porazdelitev ne dovoljuje. Te nove smeri lahko dosežemo s končnim številom neskončno kratkih premikov v smereh vektorskih

polj v osnovni porazdelitvi. Liejevi oklepaji so operacija nad vektorskima poljema $\boldsymbol{i}(\boldsymbol{q})$ in $\boldsymbol{j}(\boldsymbol{q})$, katere rezultat je novo vektorsko polje $[\boldsymbol{i}, \boldsymbol{j}]$. Definiramo ga kot

$$[\boldsymbol{i}, \boldsymbol{j}] = \frac{\partial \boldsymbol{j}}{\partial \boldsymbol{q}} \boldsymbol{i} - \frac{\partial \boldsymbol{i}}{\partial \boldsymbol{q}} \boldsymbol{j} \tag{2.28}$$

kjer sta

$$\frac{\partial \boldsymbol{i}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial i_1}{\partial \boldsymbol{q}_1} & \frac{\partial i_1}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial i_1}{\partial \boldsymbol{q}_n} \\ \frac{\partial i_2}{\partial \boldsymbol{q}_1} & \frac{\partial i_2}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial i_2}{\partial \boldsymbol{q}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial i_n}{\partial \boldsymbol{q}_1} & \frac{\partial i_n}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial i_n}{\partial \boldsymbol{q}_n} \end{bmatrix}$$

$$\frac{\partial \boldsymbol{j}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial j_1}{\partial \boldsymbol{q}_1} & \frac{\partial j_1}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial j_1}{\partial \boldsymbol{q}_n} \\ \frac{\partial j_2}{\partial \boldsymbol{q}_1} & \frac{\partial j_2}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial j_2}{\partial \boldsymbol{q}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial j_n}{\partial \boldsymbol{q}_1} & \frac{\partial j_n}{\partial \boldsymbol{q}_2} & \cdots & \frac{\partial j_n}{\partial \boldsymbol{q}_n} \end{bmatrix}$$

Porazdelitev je **vsebovana**, če z Liejevimi oklepaji ne moremo pridobiti novih linearno neodvisnih vektorskih polj. Vsebovana porazdelitev je zaprta znotraj Liejevih oklepajev [6] in sistem je popolnoma holonomičen, torej nima omejitev gibanja ali pa so vse omejitve holonomične. Ta ugotovitev izhaja iz Frobeniusovega teorema [7]: *Če je osnovna porazdelitev vsebovana, je sistem holonomičen in vse morebitne omejitve sistema so integrabilne.* Predpostavimo, da je iz m omejitev k omejitev holonomičnih in $(m-k)$ neholonomičnih. Glede na Frobeniusov teorem obstajajo tri možnosti za k [8]:

- $k = m$: dimenzija vsebovane porazdelitve (število linearno neodvisnih vektorskih polj) je enaka dimenziji osnovne porazdelitve $(n-m)$.
- $0 < k < m$: imamo k integrabilnih omejitev, torej lahko iz opisa sistema izločimo k posplošenih koordinat. V tem primeru je dimenzija vsebovane porazdelitve enaka $n-k$.
- $k = 0$: dimenzija vsebovane porazdelitve je n (dimenzija prostora) in vse omejitve so neholonomične.

Primer 2.3

Določite omejitve gibanja in smeri možnih premikov za primer enojnega kolesa, ki se kotali po podlagi. Določite število prostostnih stopenj sistema ter število in vrsto njegovih omejitev.

Rešitev

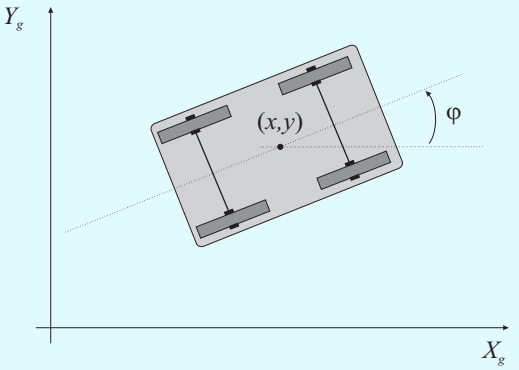
Kinematika enojnega kolesa je enaka kot pri diferencialnem pogonu (kolesi damo skupaj), prikazan na sliki 2.3 s kinematičnim modelom 2.2. Omejene in možne smeri gibanja smo že določili v primeru 2.1. Imamo samo eno hitrostno omejitev, ki ne omejuje dosegljivosti leg $\boldsymbol{q}$ v prostoru, zato ima sistem tri prostostne stopnje. Da je omejitev res hitrostna (neholonomična), lahko pokažemo z uporabo Liejevih oklepajev in Frobeniusovega teorema. Najprej določimo Liejev oklepaj za dosegljivi vektorski polji $\boldsymbol{s}_1$ in $\boldsymbol{s}_2$ (2.25)

$$\begin{aligned}\boldsymbol{s}_3 = [\boldsymbol{s}_1, \boldsymbol{s}_2] &= \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{q}} \boldsymbol{s}_1 - \frac{\partial \boldsymbol{s}_1}{\partial \boldsymbol{q}} \boldsymbol{s}_2 \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & -\sin\varphi \\ 0 & 0 & \cos\varphi \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \sin\varphi \\ -\cos\varphi \\ 0 \end{bmatrix}\end{aligned}$$

Dobimo novo smer možnega pomika, ki je linearno neodvisna od vektorskih polj $\boldsymbol{s}_1$ in $\boldsymbol{s}_2$ ter je zato ni v osnovni porazdelitvi. Posledično lahko sklepamo, da je omejitev neholonomična, dimenzija vsebovane porazdelitve pa je 3 (sistem ima 3 prostostne stopnje), kar je enako dimenziji sistema. Hkrati z dodatnimi Liejevimi oklepaji ($[\boldsymbol{s}_1, \boldsymbol{s}_3]$, $[\boldsymbol{s}_2, \boldsymbol{s}_3]$) ni možno določiti novih linearno neodvisnih vektorskih polj.

Primer 2.4

Določite omejitve gibanja in smeri možnih premikov za primer avtomobila brez krmilnega mehanizma (kolesa so fiksno vpeta), prikazan na sliki 2.16. Določite število in vrsto omejitev ter število prostostnih stopenj sistema.



Slika 2.16: Avtomobil brez krmilnega mehanizma

Rešitev

Vozilo se ne more premikati bočno niti vrteti (njegova orientacija φ se ne spreminja), torej imamo naslednje smeri omejitev gibanja

$$\boldsymbol{a}_1(\boldsymbol{q}) = \begin{bmatrix} \sin\varphi \\ -\cos\varphi \\ 0 \end{bmatrix} \qquad \boldsymbol{a}_2(\boldsymbol{q}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

in omejitvi gibanja

$$\begin{aligned} \boldsymbol{a}_1^T(\boldsymbol{q})\ \dot{\boldsymbol{q}} &= \dot{x}\sin\varphi - \dot{y}\cos\varphi = 0 \\ \boldsymbol{a}_2^T(\boldsymbol{q})\ \dot{\boldsymbol{q}} &= \dot{\varphi} = 0 \end{aligned}$$

ki sta integrabilni, kar lahko preprosto pokažemo s tem, da ju integriramo

$$\begin{aligned} \varphi &= \varphi_0 \\ (x - x_0)\sin\varphi - (y - y_0)\cos\varphi &= 0 \end{aligned}$$

Vozilo se lahko premika le v smeri vektorskega polja

$$\boldsymbol{s}_1 = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix}$$

Ker sta obe omejitvi holonomični, je porazdelitev vektorskega polja $\boldsymbol{s}_1$ vsebovana, kar pomeni, da je sistem holonomičen in ima eno prostostno stopnjo. Z Liejevimi oklepaji ne moremo določiti novih vektorskih polj (smeri premikov), saj imamo le eno vektorsko polje možnega premika.

Primer 2.5

V programskem okolju Matlab izvedite simulacijo platforme z diferencialnim pogonom. Parametri vozila so: računski korak $T_s = 0{,}033\,\mathrm{s}$, čas simulacije $10\,\mathrm{s}$, polmer kolesa $r = 0{,}04\,\mathrm{m}$, razdalja med kolesoma $L = 0{,}08\,\mathrm{m}$.

Naloge:

1. Analitično izračunajte in izvedite simulacijo poti, ki jo opravi robot, če je začetno stanje $\boldsymbol{q}(0) = [x, y, \varphi]^T = [0,\ -0{,}5,\ 0]^T$ in so vhodi robota (v treh primerih):
   - $v(t) = 0{,}5\,\mathrm{m/s}$, $\omega(t) = 0\,\mathrm{rad/s}$;

- $v(t) = 1\,\mathrm{m/s}$, $\omega(t) = 2\,\mathrm{rad/s}$;
- kotni hitrosti koles $\omega_L(t) = 24\,\mathrm{rad/s}$, $\omega_R(t) = 16\ \frac{\mathrm{rad}}{\mathrm{s}}$.

Kakšni sta kotni hitrosti koles? Ali sta izračunana in simulirana pot enaki? Zakaj ne?

2. Z uporabo odometrije izvedite lokalizacijo vozila pri konstantnih vhodnih hitrostih $v(t) = 0{,}5\,\mathrm{m/s}$, $\omega(t) = 1\,\mathrm{rad/s}$. Primerjajte ocenjeno in simulirano pot robota. Dobljene rezultate ovrednotite tudi v primeru idealne situacije, tj. brez šuma in pogreška modeliranja.

3. Izvedite lokalizacijo iz drugega vprašanja z upoštevanjem začetnega pogreška stanja. Predpostavite, da je pravo začetno stanje neznano, in za lokalizacijo uporabite drugačna začetna stanja kot pri simulaciji.

4. Izberite si pot, sestavljeno iz dveh ali treh daljic, kjer začetna točka sovpada z začetno lego robota. Izračunajte potrebno zaporedje vhodov za vožnjo robota po tej poti (čas simulacije je 10 sekund).

5. Izberite poljubno trajektorijo, definirano kot časovna parametrična funkcija ($x(t) = f(t)$ in $y(t) = g(t)$, f in g sta gladki funkciji). Izračunajte potrebne vhode za sledenje predpisani trajektoriji. Kaj se zgodi, če na začetku robot ni na tej poti?

6. Simulirajte manever paralelnega parkiranja, opisan v primeru 2.2. Izberite ustrezen ε ter konstantni hitrosti v in ω.

7. Za dano začetno stanje $\boldsymbol{q}(0) = [0,\ -0{,}5,\ 0]^T$ izračunajte kinematično matriko $\boldsymbol{S}$ in matriko omejitev $\boldsymbol{A}$.

Rešitev

Osnovna koda za simulacijo diferencialnega pogona je podana v programu 2.1. Kodo lahko priredimo in dobimo želene rešitve.

Program 2.1

./src/mdl/example_diff_drive.m

```
r = 0.04; % Radij kolesa
L = 0.08; % Dolžina osi med kolesoma
Ts = 0.03; % Računski korak
t = 0:Ts:10; % Čas simulacije
q = [4; 0.5; pi/6]; % Začetna lega

for k = 1:length(t)
    wL = 12; % Hitrost levega kolesa
    wR = 12.5; % Hitrost desnega kolesa
```

```
    v = r/2*(wR+wL); % Hitrost robota
    w = r/L*(wR-wL); % Kotna hitrost robota
    dq = [v*cos(q(3)+Ts*w/2); v*sin(q(3)+Ts*w/2); w];
    q = q + Ts*dq; % Integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območje [-pi, pi]
end
```

Primer 2.6

V programskem okolju Matlab izvedite simulacijo platforme s trikolesnim pogonom na zadnjih kolesih. Parametri vozila so: računski korak $T_s = 0{,}033\,\mathrm{s}$, čas simulacije $10\,\mathrm{s}$, polmer kolesa $r = 0{,}2\,\mathrm{m}$, razdalja med zadnjima kolesoma $L = 0{,}08\,\mathrm{m}$ ter razdalja med sprednjim in zadnjima kolesoma $D = 0{,}07\,\mathrm{m}$.

Naloge:

1. Analitično izračunajte in izvedite simulacijo poti, ki jo opravi robot, če je začetno stanje $\boldsymbol{q}(0) = [x,\ y,\ \varphi]^T = [0,\ -0{,}5,\ 0]^T$ in so vhodi robota (v dveh primerih):

   - $v(t) = 0{,}5\,\mathrm{m/s}$, $\alpha(t) = 0$
   - $v(t) = 1\,\mathrm{m/s}$, $\alpha(t) = \frac{\pi}{6}$

   Kakšne so kotne hitrosti koles? Ali sta izračunana in simulirana pot enaki? Zakaj ne?

2. Z uporabo odometrije izvedite lokalizacijo vozila pri konstantnih vhodih $v(t) = 0{,}5\,\mathrm{m/s}$, $\alpha(t) = \frac{\pi}{6}$. Primerjajte ocenjeno in simulirano pot robota. Dobljene rezultate ovrednotite tudi v primeru idealne situacije, tj. brez šuma in pogreška modeliranja.

3. Izvedite lokalizacijo iz druge naloge z upoštevanjem začetnega pogreška stanja. Predpostavite, da je pravo začetno stanje neznano, in za lokalizacijo uporabite drugačna začetna stanja kot pri simulaciji.

4. Izberite si pot, sestavljeno iz dveh ali treh daljic, kjer začetna točka sovpada z začetno lego robota. Izračunajte potrebno zaporedje vhodov za vožnjo robota po tej poti (čas simulacije je 10 sekund).

5. Izberite poljubno trajektorijo, definirano kot časovno parametrična funkcija ($x(t) = f(t)$ in $y(t) = g(t)$, f in g sta gladki funkciji). Izračunajte potrebne vhode za sledenje predpisani trajektoriji. Kaj se zgodi, če na začetku robot ni na tej poti?

Rešitev

Osnovna koda za simulacijo trikolesnega pogona je podana v programu 2.2. Kodo lahko priredimo in dobimo želene rešitve.

Program 2.2

./src/mdl/example_tricycle_drive.m

```
D = 0.07; % Razdalja med prednjim kolesom in zadnjo osjo
Ts = 0.03; % Računski korak
t = 0:Ts:10; % Čas simulacije
q = [4; 0.5; pi/6]; % Začetna lega

for k = 1:length(t)
    v = 0.5; % Hitrost robota
    alpha = 0.04*(1+sin(k*Ts*pi/2)); % Orientacija prednjega kolesa
    w = v/D*tan(alpha); % Kotna hitrost robota
    dq = [v*cos(q(3)+Ts*w/2); v*sin(q(3)+Ts*w/2); w];
    q = q + Ts*dq; % Integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območje [-pi, pi]
end
```

2.3.5 Vodljivost kolesnih mobilnih robotov

Pred načrtovanjem vodljivosti sistema se moramo vprašati: *Ali lahko robot doseže katerokoli točko* $\boldsymbol{q}$ *v delovnem prostoru z izvajanjem svojih možnih manevrov?* Odgovor na vprašanje je povezan z **vodljivostjo** sistema. Če je sistem vodljiv, lahko doseže poljubno konfiguracijo $\boldsymbol{q}$ s kombiniranjem razpoložljivih manevrov gibanja.

Če ima robot kinematične omejitve, jih moramo analizirati in ugotoviti, ali vplivajo na vodljivost sistema. Robot z neholonomičnimi omejitvami se lahko premika le v svojih osnovnih smereh gibanja (osnovni manevri, kot sta vožnja naravnost in kroženje). Vendar lahko tak robot z združevanjem osnovnih manevrov še vedno doseže želeno konfiguracijo $\boldsymbol{q}$. Nova smer gibanja se razlikuje od vsake osnovne smeri ali linearne kombinacije osnovnih smeri, kot je prikazano v (2.24). Vodljivost robota določimo z analizo vsebovane porazdelitve, pridobljene z zaporednimi operacijami Liejevih oklepajev v osnovnih smereh gibanja ($\boldsymbol{s}_1$, $\boldsymbol{s}_2$, $\boldsymbol{s}_3$ itd.) na naslednji način ([7, 9–12])

$$\{\boldsymbol{s}_1,\ \boldsymbol{s}_2,\ \boldsymbol{s}_3,\ [\boldsymbol{s}_1,\boldsymbol{s}_2],\ [\boldsymbol{s}_1,\boldsymbol{s}_3],\ [\boldsymbol{s}_2,\boldsymbol{s}_3],\ldots,\ [\boldsymbol{s}_1,[\boldsymbol{s}_1,\boldsymbol{s}_2]],\ [\boldsymbol{s}_1,[\boldsymbol{s}_1,\boldsymbol{s}_3]],\ldots\}$$

Če ima vsebovana porazdelitev rang n (n je dimenzija vektorja $\boldsymbol{q}$), lahko robot v delovnem prostoru doseže katerokoli točko $\boldsymbol{q}$ in je zato vodljiv. Sledeča izjava je znana kot Chowov izrek: *Sistem je vodljiv, če je rang njegove vsebovane porazdelitve glede na Liejeve oklepaje enak dimenziji delovnega prostora.* Torej je robot vodljiv, če so vse njegove omejitve neholonomične in je rang njegove vsebovane porazdelitve enak n. Ta test vodljivosti je namenjen nelinearnim sistemom, vendar ga je možno uporabiti tudi za linearne sisteme $\dot{\boldsymbol{q}} = \boldsymbol{A}\boldsymbol{q} + \boldsymbol{B}u$,

kjer sta osnovni vektorski polji $\boldsymbol{s}_1 = \boldsymbol{Aq}$ in $\boldsymbol{s}_2 = \boldsymbol{B}$. Izračun njegove vsebovane porazdelitve vodi v Kalmanov test za vodljivost linearnih sistemov

$$\text{rang}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} & \dots\end{bmatrix} = n$$

Vsebovano porazdelitev pridobimo z zaporednim izračunavanjem Liejevih oklepajev nad osnovnimi in predhodno določenimi novimi smermi gibanja. Število zaporednih stopenj (nivojev izračunov) določa indeks težavnosti manevriranja kolesnega mobilnega robota [13]. Višji kot je ta indeks, več osnovnih manevrov je potrebnih za doseg želene smeri gibanja.

Primer 2.7

Pokažite, da je diferencialni pogon vodljiv.

Rešitev

Diferencialni pogon ima tri ($n = 3$) stanja $\boldsymbol{q} = [x, y, \varphi]$ ter dve osnovni smeri gibanja $\boldsymbol{s}_1$ in $\boldsymbol{s}_2$

$$\boldsymbol{s}_1(\boldsymbol{q}) = \begin{bmatrix}\cos\varphi \\ \sin\varphi \\ 0\end{bmatrix} \qquad \boldsymbol{s}_2(\boldsymbol{q}) = \begin{bmatrix}0 \\ 0 \\ 1\end{bmatrix}$$

Nova smer gibanja, pridobljena z Liejevim oklepajem, je

$$\begin{aligned}\boldsymbol{s}_3 = [\boldsymbol{s}_1, \boldsymbol{s}_2] &= \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{q}}\boldsymbol{s}_1 - \frac{\partial \boldsymbol{s}_1}{\partial \boldsymbol{q}}\boldsymbol{s}_2 \\ &= \begin{bmatrix}\sin\varphi \\ -\cos\varphi \\ 0\end{bmatrix}\end{aligned}$$

Z izračunom Liejevih oklepajev iz poljubne kombinacije vektorskih polj $\boldsymbol{s}_1$, $\boldsymbol{s}_2$ in $\boldsymbol{s}_3$ ne dobimo nove linearno neodvisne smeri gibanja, zato je porazdelitev $[\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3]$ vsebovana (zaprta z Liejevimi oklepaji) in ima rang

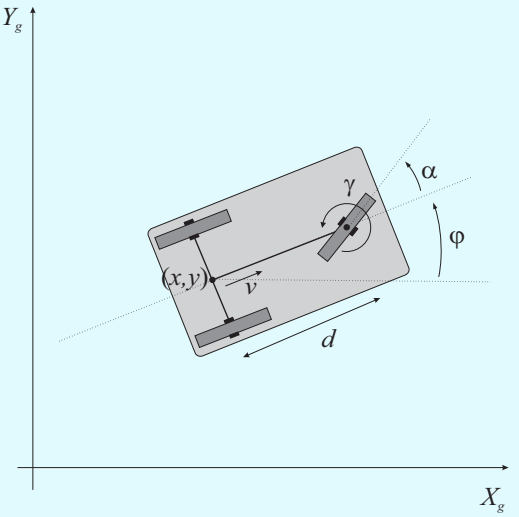
$$\text{rang}\begin{bmatrix}s_1 & \boldsymbol{s}_2 & \boldsymbol{s}_3\end{bmatrix} = n = 3$$

torej je sistem vodljiv.

Primer 2.8

Določite omejitve gibanja in smeri možnih premikov za primer vozila (slika 2.17) s pogonom na zadnjih kolesih. Vozilo upravljamo z obodno hitrostjo zadnjih koles v in kotno hitrostjo krmilnega mehanizma γ.

Določite število in vrsto omejitev, kinematični model ter število prostostnih stopenj sistema. Ali je sistem vodljiv?



Slika 2.17: Vozilo s pogonom na zadnji osi, ki ga upravljamo s hitrostjo zadnjih koles v in kotno hitrostjo krmilnega mehanizma γ

Rešitev

Vozilo opišemo s štirimi stanji $\boldsymbol{q} = [x, y, \varphi, \alpha]^T$, saj sta regulirni veličini hitrost v in kotna hitrost krmilnega mehanizma γ. Četrto stanje ($\alpha = \int_0^t \gamma \, \mathrm{d}t$) opisuje trenutno stanje krmila.

Velja opomniti, da je kinematika obravnavanega vozila podobna kinematiki kolesa z zadnjim pogonom (2.15), le da ima slednja samo tri stanja, saj je zasuk krmilnega mehanizma že določen z enim od izhodov (2.15).

Iz slike 2.17 vidimo, da se vozilo ne more premikati v smeri bočno na kolesa, torej sta vektorja omejitev gibanja

$$\boldsymbol{a}_1(\boldsymbol{q}) = \begin{bmatrix} \sin\varphi \\ -\cos\varphi \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{a}_2(\boldsymbol{q}) = \begin{bmatrix} \sin(\alpha+\varphi) \\ -\cos(\alpha+\varphi) \\ -d\cos\alpha \\ 0 \end{bmatrix}$$

Vektorski polji osnovnih smeri gibanja sta določeni s smerjo kotaljenja zadnjih koles (smer gibanja, ko je $\gamma = 0$ in konstanten zasuk krmilnega kolesa α) ter

vrtenjem krmilnega kolesa (smer gibanja, ko je $v = 0$)

$$\boldsymbol{s}_1 = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ \frac{1}{d}\tan\alpha \\ 0 \end{bmatrix} \qquad \boldsymbol{s}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Kinematika vozila je podana z

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{s}_2 \end{bmatrix} \begin{bmatrix} v \\ \gamma \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ \frac{1}{d}\tan\alpha & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \gamma \end{bmatrix}$$

kar je podobno kinematiki (2.15), le da imamo še dodatno stanje α in vhod $\gamma = \dot{\alpha}$.

Sistem je torej opisan s štirimi stanji ter ima dve omejitvi in dva vhoda. Želimo imeti štiri linearno neodvisne smeri gibanja $\boldsymbol{s}_i$. Dve že določa kinematični model, zato poskušamo z Liejevimi oklepaji določiti še preostali dve

$$\begin{aligned}
\boldsymbol{s}_3 &= [\boldsymbol{s}_1, \boldsymbol{s}_2] \\
&= \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{q}} \boldsymbol{s}_1 - \frac{\partial \boldsymbol{s}_1}{\partial \boldsymbol{q}} \boldsymbol{s}_2 \\
&= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ \frac{1}{d}\tan\alpha \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & -\sin\varphi & 0 \\ 0 & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & \frac{1}{d\cos^2\alpha} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ \frac{-1}{d\cos^2\alpha} \\ 0 \end{bmatrix}
\end{aligned}$$

in

$$\begin{aligned}
\boldsymbol{s}_4 &= [\boldsymbol{s}_1, \boldsymbol{s}_3] \\
&= \frac{\partial \boldsymbol{s}_3}{\partial \boldsymbol{q}} \boldsymbol{s}_1 - \frac{\partial \boldsymbol{s}_1}{\partial \boldsymbol{q}} \boldsymbol{s}_3 \\
&= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-2\sin\alpha}{d\cos^3\alpha} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ \frac{1}{d}\tan\alpha \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & -\sin\varphi & 0 \\ 0 & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & \frac{1}{d\cos^2\alpha} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \frac{-1}{d\cos^2\alpha} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \frac{-\sin\varphi}{d\cos^2\alpha} \\ \frac{\cos\varphi}{d\cos^2\alpha} \\ 0 \\ 0 \end{bmatrix}
\end{aligned}$$

Vse štiri dobljene smeri $\boldsymbol{s}_i$, $i = 1, \ldots, 4$ so linearno neodvisne, zato ima sistem štiri prostostne stopnje in obe omejitvi sta neholonomični.

Rang vsebovane porazdelitve je 4, kar je enako številu prostostnih stopenj sistema. Vozilo je torej vodljivo.

2.4 Dinamični model mobilnega sistema z omejitvami

Kinematični model opisuje le statično transformacijo hitrosti robota (*psevdohitrosti*) v osnovni koordinatni sistem, podan s posplošenimi koordinatami. Dinamični model gibanja mehanskega sistema pa podaja dinamične zakonitosti, kot je gibanje sistema pod vplivom zunanjih sil in vztrajnosti sistema. Z uporabo Lagrangeove formulacije, ki je še posebej primerna za opis mehanskih sistemov [14], lahko določimo dinamični model sistema

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \frac{\partial \mathcal{L}}{\partial q_k} + \frac{\partial P}{\partial \dot{q}_k} + g_k + \tau_{d_k} = f_k \tag{2.29}$$

kjer indeks k opisuje posplošene koordinate q_k ($k = 1, \ldots, n$). Z $\mathcal{L}$ je označena razlika med kinetično in potencialno energijo sistema, imenovana *Lagrangian*, P je močnostna funkcija zaradi trenja in dušenja v sistemu, g_k označuje sile zaradi gravitacije, τ_{d_k} predstavlja vse neznane motnje, ki jih z enačbo 2.29 nismo zajeli v modelu, f_k pa je posplošena sila (zunanji vplivi na sistem), povezana s posplošeno koordinato q_k. Enačba (2.29) velja samo za sisteme brez omejitev gibanja, torej za sisteme, ki imajo n prostostnih stopenj in so brez hitrostnih omejitev. Za sisteme s kinematičnimi omejitvami lahko zapišemo dinamične enačbe gibanja z uporabo Lagrangeovih multiplikatorjev [15]

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \frac{\partial \mathcal{L}}{\partial q_k} + \frac{\partial P}{\partial \dot{q}_k} + g_k + \tau_{d_k} = f_k - \sum_{j=1}^{m} \lambda_j a_{jk} \tag{2.30}$$

kjer je m število linearno neodvisnih omejitev gibanja, λ_j Lagrangeov multiplikator, povezan z j-to omejitveno enačbo, a_{jk} ($j = 1, \ldots, m$, $k = 1, \ldots, n$) pa koeficienti omejitev.

Končni nabor enačb vsebuje $n + m$ diferencialnih in algebrajskih enačb (n Lagrangeovih enačb in m omejitvenih enačb) z $n+m$ neznankami (n posplošenih koordinat q_k in m Lagrangeovih multiplikatorjev λ_j). Enačbe so diferencialne v smislu posplošenih koordinat in algebrajske glede na Lagrangeove multiplikatorje.

Splošni dinamični model (2.30) mehanskega sistema z omejitvami lahko zapišemo v matrični obliki

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{V}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{\tau}_d = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda} \tag{2.31}$$

Tabela 2.1: Pomen matrik v dinamičnem modelu (2.31)

Oznaka	*Opis*
$\boldsymbol{q}$	vektor posplošenih koordinat (dimenzije $n \times 1$)
$\boldsymbol{M}(\boldsymbol{q})$	pozitivno definitna matrika mas in vztrajnosti (dimenzije $n \times n$)
$\boldsymbol{V}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	vektor Coriolisovih in centrifugalnih členov (dimenzije $n \times 1$)
$\boldsymbol{F}(\dot{\boldsymbol{q}})$	vektor sil trenja in dušenja (dimenzije $n \times 1$)
$\boldsymbol{G}(\boldsymbol{q})$	vektor gravitacijskih sil in navorov (dimenzije $n \times 1$)
$\boldsymbol{\tau}_d$	vektor neznanih motenj, vključno z dinamiko, ki ni zajeta v modelu (dimenzije $n \times 1$)
$\boldsymbol{E}(\boldsymbol{q})$	matrika preslikav aktuatorskega prostora v prostor posplošenih spremenljivk (dimenzije $n \times r$)
$\boldsymbol{u}$	vektor vhodov (dimenzije $r \times 1$)
$\boldsymbol{A}^T(\boldsymbol{q})$	matrika kinematičnih omejitev (dimenzije $m \times n$)
$\boldsymbol{\lambda}$	vektor omejitvenih sil (Lagrangeovi multiplikatorji) (dimenzije $m \times 1$)

kjer je pomen matrik opisan v tabeli 2.1.

2.4.1 Predstavitev dinamičnega modela mobilnega sistema z omejitvami v prostoru stanj

V nadaljevanju bomo izpeljali dinamični model sistema z m kinematičnimi omejitvami v prostoru stanj. Nadalje bomo izvedli delno linearizacijo sistema, opisanega v prostoru stanj, z vpeljavo nelinearne povratnozančne relacije [6]. Dobljeni sistem bomo zapisali kot kinematični model drugega reda.

Dinamični sistem z m kinematičnimi omejitvami zapišemo kot

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{V}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda} \tag{2.32}$$

kjer smo vpliv neznanih motenj na sistem $\boldsymbol{\tau}_d$ iz (2.31) zanemarili. Kinematični model gibanja pa podaja enačba

$$\dot{\boldsymbol{q}} = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{v}(t) \tag{2.33}$$

Dinamični model (2.32) in kinematični model (2.33) lahko združimo v enoten zapis v prostoru stanj. Poenoteno obravnavo neholonomičnih in holonomičnih omejitev lahko najdemo v [16], kjer so holonomične omejitve izražene v diferencialni obliki (s hitrostmi) kot neholonomične.

Zaradi enostavnosti zapisa bomo v nadaljevanju izpustili odvisnost od $\boldsymbol{q}$. Če odvajamo relacijo (2.33) po času, dobimo

$$\ddot{\boldsymbol{q}} = \dot{\boldsymbol{S}}\boldsymbol{v} + \boldsymbol{S}\dot{\boldsymbol{v}}$$

Dobljen izraz vstavimo v (2.32) ter zamenjamo posplošene koordinate $\boldsymbol{q}$ s psevdohitrostmi $\boldsymbol{v}$. Tako dobimo

$$\boldsymbol{M}\dot{\boldsymbol{S}}\boldsymbol{v} + \boldsymbol{M}\boldsymbol{S}\dot{\boldsymbol{v}} + \boldsymbol{V} + \boldsymbol{F} + \boldsymbol{G} = \boldsymbol{E}\boldsymbol{u} - \boldsymbol{A}^T\boldsymbol{\lambda} \tag{2.34}$$

Prisotnost Lagrangeovih multiplikatorjev $\boldsymbol{\lambda}$ zaradi omejitev gibanja lahko izločimo z upoštevanjem relacij $\boldsymbol{A}\boldsymbol{S} = \boldsymbol{0}$ in $\boldsymbol{S}^T\boldsymbol{A}^T = \boldsymbol{0}$. Z množenjem enačbe (2.34) z matriko $\boldsymbol{S}^T$ dobimo skrčeno obliko dinamičnega modela

$$\boldsymbol{S}^T\boldsymbol{M}\dot{\boldsymbol{S}}\boldsymbol{v} + \boldsymbol{S}^T\boldsymbol{M}\boldsymbol{S}\dot{\boldsymbol{v}} + \boldsymbol{S}^T\boldsymbol{V} + \boldsymbol{S}^T\boldsymbol{F} + \boldsymbol{S}^T\boldsymbol{G} = \boldsymbol{S}^T\boldsymbol{E}\boldsymbol{u}$$

s čimer smo izločili Lagrangeove multiplikatorje $\boldsymbol{\lambda}$. Z vpeljavo zamenjav $\widetilde{\boldsymbol{M}} = \boldsymbol{S}^T\boldsymbol{M}\boldsymbol{S}$, $\widetilde{\boldsymbol{V}} = \boldsymbol{S}^T\boldsymbol{M}\dot{\boldsymbol{S}}\boldsymbol{v} + \boldsymbol{S}^T(\boldsymbol{V} + \boldsymbol{F} + \boldsymbol{G})$ in $\widetilde{\boldsymbol{E}} = \boldsymbol{S}^T\boldsymbol{E}$ lahko pregledneje zapišemo

$$\widetilde{\boldsymbol{M}}\dot{\boldsymbol{v}} + \widetilde{\boldsymbol{V}} = \widetilde{\boldsymbol{E}}\boldsymbol{u} \tag{2.35}$$

od koder izrazimo vektor psevdopospeškov $\dot{\boldsymbol{v}}$

$$\dot{\boldsymbol{v}} = \widetilde{\boldsymbol{M}}^{-1}\left(\widetilde{\boldsymbol{E}}\boldsymbol{u} - \widetilde{\boldsymbol{V}}\right)$$

Če je nadalje izpolnjen pogoj $\det \boldsymbol{S}^T\boldsymbol{E} \neq 0$, kar v večini realističnih primerov je, lahko iz enačbe (2.35) izrazimo vhod v sistem

$$\boldsymbol{u} = \widetilde{\boldsymbol{E}}^{-1}\left(\widetilde{\boldsymbol{M}}\dot{\boldsymbol{v}} + \widetilde{\boldsymbol{V}}\right) \tag{2.36}$$

Z razširitvijo vektorja stanj s psevdohitrostmi $\boldsymbol{x} = [\boldsymbol{q}^T\boldsymbol{v}^T]^T$ in zapisom sistema v splošni nelinearni obliki $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$ (člen $\boldsymbol{f}(\boldsymbol{x})$ vsebuje nelinearno odvisnost od stanj) dobimo zapis sistema v prostoru stanj

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{S}\boldsymbol{v} \\ -\widetilde{\boldsymbol{M}}^{-1}\widetilde{\boldsymbol{V}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0}_{n\times r} \\ \widetilde{\boldsymbol{M}}^{-1}\widetilde{\boldsymbol{E}} \end{bmatrix}\boldsymbol{u} \tag{2.37}$$

kjer je r število vhodov v vektorju $\boldsymbol{u}$. Dimenzija vektorja stanj $\boldsymbol{x}$ je $(2n-m)\times 1$.

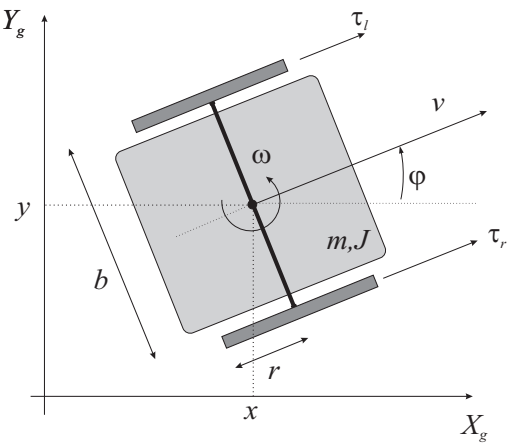
Z inverznim modelom (2.36) lahko za želene psevdopospeške sistema izračunamo potreben vhod v sistem. Z uporabo izračunanih vhodov v sistemu (2.37) dobimo naslednji skupni model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{S}\boldsymbol{v} \\ \boldsymbol{0}_{(n-m)\times 1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0}_{n\times(n-m)} \\ \boldsymbol{I}_{(n-m)\times(n-m)} \end{bmatrix}\boldsymbol{u}_z$$

kjer $\boldsymbol{u}_z$ predstavlja psevdopospeške sistema. Izraz (2.36) lahko torej uporabimo za izračun napovedanih vhodov sistema. Te vhode lahko pri vodenju uporabimo samostojno (odprtozančno vodenje) ali pa v kombinaciji z zaprtozančnim vodenjem ter tako dobimo regulacijo s predkrmiljenjem.

2.4.2 Kinematični in dinamični model robota z diferencialnim pogonom

Vozilo z diferencialnim pogonom na sliki 2.3 ima kolesa, gnana s pomočjo dveh elektromotorjev. Predpostavimo, da je težišče robota v njegovem geometrijskem



Slika 2.18: Vozilo z diferencialnim pogonom

središču. Masa vozila brez koles je m_v, masa koles pa je m_w. Vozilo se giblje po ploskvi; njegov vztrajnostni moment okoli osi z označimo kot J_c (os z je pravokotna na ploskev) in vztrajnostni moment koles kot J_w.

V praksi je običajno masa koles veliko manjša od mase ohišja vozila, zato lahko uporabimo skupno maso m in vztrajnost J. Gibanje vozila opišemo s tremi posplošenimi koordinatami $q = [x, y, \varphi]$, vhod v sistem pa predstavlja navor na levo in desno kolo (τ_l, τ_r prikazano na sliki 2.18).

Kinematični model in omejitve

Kinematični model vozila (2.2) je

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

neholonomična omejitev gibanja pa je (vozilo se ne more premikati pravokotno na smer vrtenja koles)

$$-\dot{x}\sin\varphi + \dot{y}\cos\varphi = 0$$

kar pomeni, da stolpci kinematične matrike predstavljajo dosegljive smeri premikov

$$\boldsymbol{S} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{bmatrix}$$

in matrika koeficientov omejitev je

$$\boldsymbol{A} = \begin{bmatrix} -\sin\varphi & \cos\varphi & 0 \end{bmatrix}$$

Dinamični model

Dinamični model izpeljemo z Lagrangeovo formulacijo

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \frac{\partial \mathcal{L}}{\partial q_k} + \frac{\partial P}{\partial \dot{q}_k} = f_k - \sum_{j=1}^{m} \lambda_j a_{jk} \tag{2.38}$$

kjer smo iz relacije (2.30) izpustili člen, povezan z vplivi neznanih motenj τ_{d_k}. Podobno ne upoštevamo sil in navorov zaradi gravitacije g_k, ker se vozilo vozi po ploskvi in je tako potencialna energija konstantna (brez izgube posplošenosti lahko predpostavimo $W_P = 0$).

Skupno kinetično energijo sistema lahko opišemo z relacijo

$$W_K = \frac{m}{2}\left(\dot{x}^2 + \dot{y}^2\right) + \frac{J}{2}\dot{\varphi}^2$$

Ker je potencialna energija sistema $W_P = 0$, je Lagrangian enak

$$\mathcal{L} = W_K - W_P = \frac{m}{2}\left(\dot{x}^2 + \dot{y}^2\right) + \frac{J}{2}\dot{\varphi}^2$$

Hkrati zanemarimo še vpliv dušenja in trenja pri kotaljenju koles ($P = 0$). Sile in navori v enačbi (2.38) so

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) &= m\ddot{x} \\
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{y}}\right) &= m\ddot{y} \\
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\varphi}}\right) &= J\ddot{\varphi}
\end{aligned}$$

in

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x} &= 0 \\
\frac{\partial \mathcal{L}}{\partial y} &= 0 \\
\frac{\partial \mathcal{L}}{\partial \varphi} &= 0
\end{aligned}$$

Za dinamični model lahko glede na Lagrangeovo formulacijo (2.38) zapišemo sledeče diferencialne enačbe

$$\begin{aligned}
m\ddot{x} - \lambda_1 \sin\varphi &= F_x \\
m\ddot{y} + \lambda_1 \cos\varphi &= F_y \\
J\ddot{\varphi} &= T
\end{aligned}$$

r je radij kolesa. Rezultanta sil na levem in desnem kolesu je $F = \frac{1}{r}(\tau_r + \tau_l)$. Sila v smeri osi x je $F_x = \frac{1}{r}(\tau_r + \tau_l)\cos\varphi$, v smeri osi y pa $F_y = \frac{1}{r}(\tau_r + \tau_l)\sin\varphi$. Navor, ki deluje na vozilo, je $T = \frac{L}{2r}(\tau_r - \tau_l)$, pri čemer je L razdalja med kolesi.

Imamo torej model

$$\begin{aligned} m\ddot{x} - \lambda_1 \sin\varphi - \frac{1}{r}(\tau_r + \tau_l)\cos\varphi = 0 \\ m\ddot{y} + \lambda_1 \cos\varphi - \frac{1}{r}(\tau_r + \tau_l)\sin\varphi = 0 \\ J\ddot{\varphi} - \frac{L}{2r}(\tau_r - \tau_l) = 0 \end{aligned}$$

Dobljeni dinamični model lahko zapišemo v matrični obliki

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{V}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{\lambda}$$

kjer so matrike

$$\begin{aligned} \boldsymbol{M} &= \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & J \end{bmatrix} \\ \boldsymbol{E} &= \frac{1}{r}\begin{bmatrix} \cos\varphi & \cos\varphi \\ \sin\varphi & \sin\varphi \\ \frac{L}{2} & -\frac{L}{2} \end{bmatrix} \\ \boldsymbol{A} &= \begin{bmatrix} -\sin\varphi & \cos\varphi & 0 \end{bmatrix} \\ \boldsymbol{u} &= \begin{bmatrix} \tau_r \\ \tau_l \end{bmatrix} \end{aligned}$$

ostale matrike pa so ničelne.

Model v prostoru stanj (2.37), ki vključuje kinematiko in dinamiko, je določen z matrikami (glejte poglavje 2.4.1)

$$\begin{aligned} \widetilde{\boldsymbol{M}} &= \begin{bmatrix} m & 0 \\ 0 & J \end{bmatrix} \\ \widetilde{\boldsymbol{V}} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \widetilde{\boldsymbol{E}} &= \frac{1}{r}\begin{bmatrix} 1 & 1 \\ \frac{L}{2} & -\frac{L}{2} \end{bmatrix} \end{aligned}$$

od koder lahko glede na enačbo (2.37) zapišemo sistem v prostoru stanj v obliki $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}$, kjer je vektor stanj določen z $\boldsymbol{x} = [\boldsymbol{q}^T,\ \boldsymbol{v}^T]^T$. Dobimo

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\varphi \\ v\sin\varphi \\ \omega \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{mr} & \frac{1}{mr} \\ \frac{L}{2Jr} & \frac{-L}{2Jr} \end{bmatrix} \begin{bmatrix} \tau_r \\ \tau_l \end{bmatrix}$$

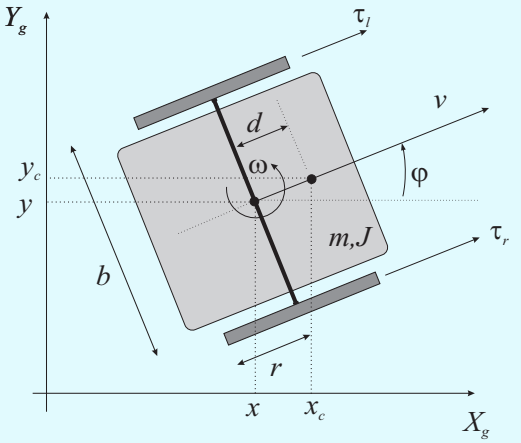
Inverzni model sistema določimo glede na relacijo (2.36) kot

$$\begin{bmatrix} \tau_r \\ \tau_l \end{bmatrix} = \begin{bmatrix} \frac{\dot{v}mr}{2} + \frac{\dot{\omega}Jr}{L} \\ \frac{\dot{v}mr}{2} - \frac{\dot{\omega}Jr}{L} \end{bmatrix}$$

Z uporabo inverznega modela lahko za določene hitrosti in pospeške robota izračunamo potrebna navora na obe kolesi. Izračunana vhoda lahko uporabimo za odprtozančno vodenje ali bolje v kombinaciji z zaprtozančnim vodenjem (regulacija s predkrmiljenjem).

Primer 2.9

Zapišite kinematični in dinamični model za vozilo z diferencialnim pogonom, prikazano na sliki 2.19. Modela izrazite s koordinatama masnega središča (x_c, y_c), ki sta od geometrijskega središča (x, y) oddaljeni za razdaljo d.



Slika 2.19: Vozilo z diferencialnim pogonom z masnim središčem (x_c, y_c) in geometrijskim središčem (x, y)

Rešitev

Če upoštevamo transformacijo med geometrijskim in masnim središčem $x = x_c - d\cos\varphi$ in $y = y_c - d\sin\varphi$ ter njuna časovna odvoda $\dot{x} = \dot{x}_c + d\dot{\varphi}\sin\varphi$ in $\dot{y} = \dot{y}_c - d\dot{\varphi}\cos\varphi$, ki ju vstavimo v kinematični model (slika 2.3), sta končni kinematični model in kinematična omejitev masnega središča

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -d\sin(\varphi) \\ \sin(\varphi) & d\cos(\varphi) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$-\dot{x}_c \sin\varphi + \dot{y}_c \cos\varphi - \dot{\varphi} d = 0$$

Lagrangian za masno središče je $\mathcal{L} = \frac{m}{2}\left(\dot{x}_c^2 + \dot{y}_c^2\right) + \frac{J}{2}\dot{\varphi}^2$. Glede na (2.38) je dinamični model

$$
\begin{aligned}
m\ddot{x}_c - \lambda_1 \sin\varphi &= \frac{1}{r}(\tau_r + \tau_l)\cos\varphi \\
m\ddot{y}_c + \lambda_1 \cos\varphi &= F_y = \frac{1}{r}(\tau_r + \tau_l)\sin\varphi \\
J\ddot{\varphi} - \lambda_1 d &= \frac{L}{2r}
\end{aligned}
$$

in matrike modela

$$
\begin{aligned}
\boldsymbol{M} &= \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & J \end{bmatrix} \\
\boldsymbol{E} &= \frac{1}{r}\begin{bmatrix} \cos\varphi & \cos\varphi \\ \sin\varphi & \sin\varphi \\ \frac{L}{2} & -\frac{L}{2} \end{bmatrix} \\
\boldsymbol{A} &= \begin{bmatrix} -\sin\varphi & \cos\varphi & -d \end{bmatrix} \\
\boldsymbol{S} &= \begin{bmatrix} \cos(\varphi(t)) & -d\sin(\varphi) \\ \sin(\varphi(t)) & d\cos(\varphi) \\ 0 & 1 \end{bmatrix}
\end{aligned}
$$

Glede na (2.37) je skupna predstavitev sistema v prostoru stanj

$$
\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\varphi} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\varphi - \omega d\sin\varphi \\ v\sin\varphi + \omega d\cos\varphi \\ \omega \\ d\omega^2 \\ \frac{-dv\omega m}{md^2+J} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{mr} & \frac{1}{mr} \\ \frac{L}{2r(d^2m+J)} & \frac{-L}{2r(d^2m+J)} \end{bmatrix} \begin{bmatrix} \tau_r \\ \tau_l \end{bmatrix}
$$

Primer 2.10

Zapišite kinematični in dinamični model vozila iz primera 2.9, kjer masno in geometrijsko središče nista enaka. Modela izrazite s pomočjo koordinat geometrijskega središča (x, y), saj je to bolj ustrezno, če se masno središče vozila spreminja z različnim tovorom.

Rešitev

Kinematični model in kinematična omejitev geometrijskega središča sta

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\varphi}\end{bmatrix} = \begin{bmatrix}\cos(\varphi) & 0\\ \sin(\varphi) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

$$-\dot{x_c}\sin\varphi + \dot{y_c}\cos\varphi = 0$$

Lagrangian je $\mathcal{L} = \frac{m}{2}\left(\dot{x}_c^2 + \dot{y}_c^2\right) + \frac{J}{2}\dot{\varphi}^2$ kjer je masno središče izraženo z geometrijskim. Končni model v prostoru stanj je

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\varphi}\\ \dot{v}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}v\cos\varphi\\ v\sin\varphi\\ \omega\\ d\omega^2\\ C\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ \frac{1}{mr} & \frac{1}{mr}\\ \frac{L-2d\sin(2\varphi)}{2r(d^2m+J)} & \frac{-L-2d\sin(2\varphi)}{2r(d^2m+J)}\end{bmatrix}\begin{bmatrix}\tau_r\\ \tau_l\end{bmatrix}$$

kjer je $C = \frac{-d\omega m(\dot{x}\cos\varphi + \dot{y}\sin\varphi - v\cos(2\varphi) - d\omega\sin(2\varphi))}{md^2+J}$.

Primer 2.11

V programskem okolju Matlab izračunajte potrebne navore za gibanje mobilnega robota iz primera 2.9 po referenčni trajektoriji $x_r = 1{,}1 + 0{,}7\sin(2\pi/30)$, $y_r = 0{,}9 + 0{,}7\sin(4\pi/30)$ brez uporabe senzorjev (odprtozančno vodenje). S pomočjo simulacije izračunajte navore robota z uporabo inverznega dinamičnega modela in prikažite dobljeno trajektorijo sistema. Parametri robota so: $m = 0{,}75\,\mathrm{kg}$, $J = 0{,}001\,\mathrm{kgm}^2$, $L = 0{,}075\,\mathrm{m}$, $r = 0{,}024\,\mathrm{m}$ in $d = 0{,}01\,\mathrm{m}$.

Rešitev

Glede na (2.36) je inverzni model robota

$$\begin{bmatrix}\tau_r\\ \tau_l\end{bmatrix} = \begin{bmatrix}\frac{r(\dot{v}m - d\omega^2 m)}{2} + \frac{r\left(\dot{\omega}\left(md^2+J\right)+d\omega^2 m\right)}{L}\\ \frac{r(\dot{v}m - d\omega^2 m)}{2} - \frac{r\left(\dot{\omega}\left(md^2+J\right)+d\omega^2 m\right)}{L}\end{bmatrix}$$

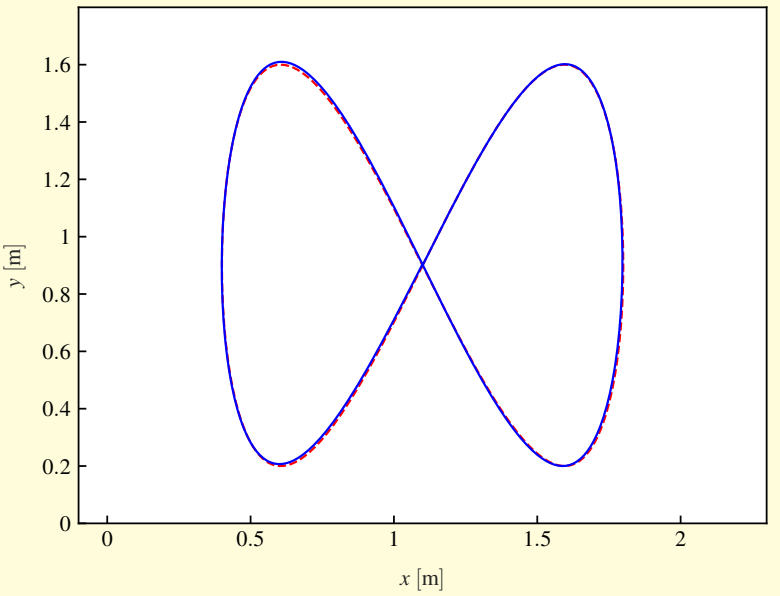
Iz referenčne točke izračunamo referenčno hitrost v_r in referenčno kotno hitrost ω_r ter njuna časovna odvoda $\dot{v}_r$ in $\dot{\omega}_r$.

Koda Matlab je podana v programu 2.3. Rezultati simulacije pa so prikazani na slikah 2.20 in 2.21.

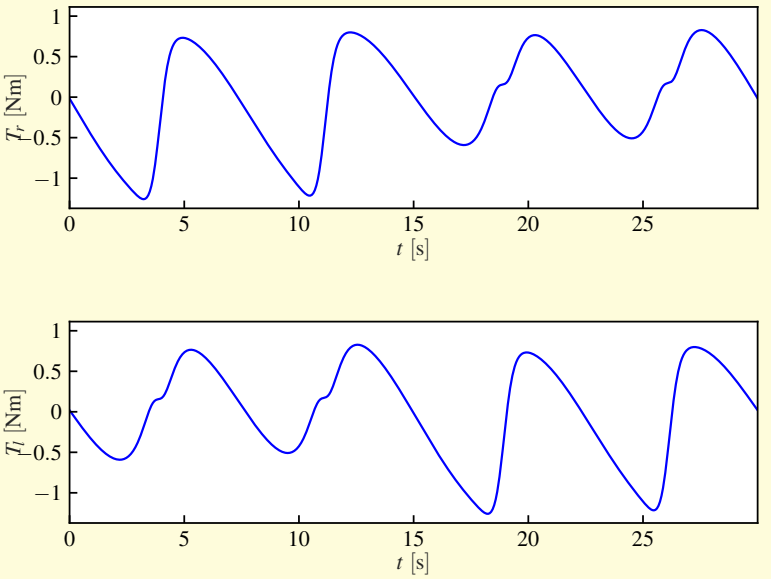
Program 2.3

./src/mdl/example_dynamic_model.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije

% Referenca
freq = 2*pi/30;
   xRef = 1.1 + 0.7*sin(freq*t);     yRef = 0.9 + 0.7*sin(2*freq*t);
  dxRef = freq*0.7*cos(freq*t);     dyRef = 2*freq*0.7*cos(2*freq*t);
 ddxRef =-freq^2*0.7*sin(freq*t);  ddyRef =-4*freq^2*0.7*sin(2*freq*t);
dddxRef =-freq^3*0.7*cos(freq*t); dddyRef =-8*freq^3*0.7*cos(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
dvRef = (dxRef.*ddxRef+dyRef.*ddyRef)./vRef;
dwRef = (dxRef.*dddyRef-dyRef.*dddxRef)./vRef.^2 - 2.*wRef.*dvRef./vRef;

q = [qRef(:,1); vRef(1); wRef(2)]; % Začetna lega robota
m = 0.75; J = 0.001; L = 0.075; r = 0.024; d = 0.01; % Parametri robota

for k = 1:length(t)
    % Izračun navorov iz trajektorije in inverznega modela
    v = vRef(k); w = wRef(k); dv = dvRef(k); dw = dwRef(k);
    tau = [(r*(dv*m-d*w*m*w))/2 + (r*(dw*(m*d^2+J) + d*w*m*v))/L; ...
           (r*(dv*m-d*w*m*w))/2 - (r*(dw*(m*d^2+J) + d*w*m*v))/L];

    % Simulacija kinematičnega in dinamičnega modela gibanja robota
    phi = q(3); v = q(4); w = q(5);
    F = [v*cos(phi) - d*w*sin(phi); ...
         v*sin(phi) + d*w*cos(phi); ...
         w; ...
         d*w^2; ...
       -(d*w*v*m)/(m*d^2 + J)];
    G = [0,                    0; ...
         0,                    0; ...
         0,                    0; ...
         1/(m*r),              1/(m*r); ...
         L/(2*r*(m*d^2 + J)), -L/(2*r*(m*d^2 + J))];
    dq = F + G*tau; % Model v prostoru stanj
    q = q + dq*Ts; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območje [-pi, pi]
end
```



Slika 2.20: Izračunana pot robota (polna krivulja) in referenčna pot (črtkana krivulja)



Slika 2.21: Navora na kolesi

Literatura

[1] H. Choset, K. Lynch in sod. *Principles of robot motion: theory, algorithms, and implementations*. MIT Press, illustrate izd., 2005, 603 str.

[2] B. Siciliano in O. Khatib. *Springer Handbook of Robotics*. Springer, 2008, 1611 str.

[3] G. Dudek in M. Jenkin. *Computational Principlec of Mobile Robotics*. Cambridge University press, 2010.

[4] G. Klančar in I. Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, zv. 55, št. 6, str. 460–469, 2007.

[5] I. Kolmanovsky in N. H. McClamroch. Developments in Nonholonomic Control Problems. *IEEE Control Systems*, zv. 15, št. 6, str. 20–36, 1995.

[6] A. De Luca in G. Oriolo. Modelling and Control of Nonholonomic Mechanical Systems. V J. Angeles in A. Kecskeméthy (Uredniki), *Kinematics and Dynamics of Multi-Body Systems*, CISM International Centre for Mechanical Sciences, str. 277–342. Springer, Vienna, 1995.

[7] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, New York, 1991.

[8] N. Sarkar, X. Yun in V. Kumar. Control of mechanical systems with rolling constraints application to dynamic control of mobile robots. *The International Journal of Robotics Research*, zv. 13, št. 1, str. 55–69, 1994.

[9] R. Hermann in A. J. Krener. Nonlinear Controllability and Observability. *IEEE Transaction on Automatic Control*, zv. 22, št. 5, str. 728–740, 1977.

[10] R. W. Brockett. Asymptotic stability and feedback stabilization. V R. S. Millman in H. J. Sussmann (Uredniki), *Differential Geometric Control Theory*, str. 181–191. Birkhuser, Boston, MA, 1983.

[11] J. P. Laumond. *Robot Motion Planning and Control*, *Lecture Notes in Control and Information Sciences*, zv. 229. Springer, 1998.

[12] W. S. Levine. *The Control Handbook, Second Edition* . Electrical Engineering Handbook. CRC Press, 2010.

[13] A. De Luca, G. Oriolo in M. Vendittelli. Control of Wheeled Mobile Robots: An Experimental Overview. V S. Nicosia, B. Siciliano, A. Bicchi in P. Valigi (Uredniki), *Ramsete*, *Lecture Notes in Control and Information Sciences*, zv. 270, str. 181–226. Springer Berlin Heidelberg, 2001.

[14] O. Egeland in J. T. Gravdahl. *Modeling and Simulation for Automatic Control* . Marine Cybernetics, Trondheim, Norway, 2002.

[15] Z. Li in J. F. Canny. *Nonholonomic Motion Planning*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1993.

[16] X. Yun. State Space Representation of Holonomic and Nonholonomic Constraints Resulting from Rolling Contacts. V *IEEE International Conference on Robotics and Automation*, str. 2690–2694. IEEE, Nagoya, 1995.

3

Vodenje kolesnih mobilnih sistemov

3.1 Uvod

Vodenje kolesnih mobilnih robotov v okolju brez ovir lahko izvedemo z vodenjem od začetne do končne lege (klasično sledilno vodenje, kjer niso predpisana vmesna stanja trajektorije) ali pa preko sledenja referenčni trajektoriji. V primeru neholonomičnih kolesnih mobilnih sistemov se izkaže, da je bolj težavno vodenje v referenčno lego, kot pa sledenje referenčni trajektoriji, ki povezuje začetno in ciljno lego. Za uspešno vodenje (tako klasično vodenje kot sledenje trajektoriji) moramo uporabiti nezvezen ali časovno spremenljiv regulator [1], ker je sistem, ki ga vodimo, nelinearen in časovno spremenljiv. Nadalje mora robot pri gibanju upoštevati neholonomične omejitve, torej njegova pot ne more biti poljubna. Dodatni razlog za uporabo sledenja trajektoriji se skriva v dejstvu, da se mobilni robot pogosto giblje v prostoru z omejitvami, ovirami in raznimi zahtevami, ki v določeni meri predpisujejo želeno pot do cilja.

Neholonomične sisteme je smiselno voditi z *dvoprostostnim regulatorjem*, sestavljen iz **predkrmiljenja** (angl. *feedforward*) in **povratne zanke** (angl. *feedback*). Predkrmiljenje se izračuna iz referenčne trajektorije, kjer določimo oz. predhodno izračunamo potrebne vhode v sistem, da bo le-ta sledil referenčni trajektoriji brez povratnih informacij senzorjev (odprtozančno vodenje). Tovrstno vodenje ni praktično uporabno, saj ni odporno na motnje v delovanju sistema in pogreške začetnih stanj. Zato je potrebno vključiti še povratno zanko (zaprtozančni del). Dvoprostostni regulator je naraven in prikladen za vodenje

neholonomičnih mehanskih sistemov, zato bo uporabljen v večini prikazanih primerov v nadaljevanju.

Kolesni mobilni roboti so **dinamični** sistemi, kjer je za želeno gibanje platforme potreben ustrezen navor na kolesa. Zato morajo algoritmi za vodenje gibanja upoštevati dinamične lastnosti sistema. Običajno se to težavo reši s pomočjo kaskadnih regulacijskih shem z zunanjim regulatorjem za vodenje hitrosti in notranjim za vodenje navora (sila, tok motorja itd.). Zunanji regulator določa potrebne hitrosti sistema, da sistem doseže referenčno lego ali sledi referenčni trajektoriji. Notranji (hitrejši) regulator pa izračuna potrebne navore (sila, tok motorja itd.), da doseže hitrosti sistema, ki jih določa zunanji regulator. Notranji regulator mora biti dovolj hiter, da dodatni fazni zamik ne povzroča težav. Pri večini primerov je notranji regulator navora že vgrajen v mobilnega robota, uporabnik pa z vodenjem določa želene hitrosti sistema glede na njegovo kinematiko.

Preostali del tega poglavja je razdeljen na različne pristope vodenja za doseg referenčne lege ter pristope vodenja sledenja referenčni trajektoriji. Prvi bodo vključevali osnovno idejo in nekaj preprostih primerov uporabe na različnih platformah, slednji pa bodo obravnavani bolj podrobno, saj so ti pristopi bolj naravni za kolesne mobilne robote, ki se vozijo v okoljih z znanimi ovirami.

3.2 Vodenje v referenčno lego

V nadaljevanju bomo predstavili osnovne pristope k vodenju mobilnega robota v referenčno lego, ki jo določata pozicija in orientacija. V tem primeru pot ali trajektorija do referenčne lege ni predpisana, zato se lahko robot vozi do cilja po katerikoli izvedljivi poti. To pot lahko eksplicitno določimo in jo med vožnjo tudi sproti prilagajamo ali pa jo podamo implicitno z izvedbo algoritma vodenja v referenčno lego.

Običajno sta podani samo začetna (ali trenutna) in končna (ali referenčna) lega s poljubno potjo med njima, kar odpira nove možnosti npr. za izbiro "optimalne" poti. Pri izbiri poti moramo upoštevati vse omejitve – kinematične, dinamične in okoljske. To nam običajno še vedno omogoča izbiro neskončno mnogo poti, kjer pa izberemo tisto, ki upošteva tudi dodatna merila, kot so čas, dolžina, ukrivljenost, poraba energije ipd. V splošnem je načrtovanje poti zahtevna naloga, zato v tem razdelku ne bomo upoštevali teh vidikov.

V nadaljevanju bo vodenje v referenčno lego razdeljeno na dve ločeni nalogi: *vodenje orientacije* in *vodenje gibanja naprej*. Ne moremo ju obravnavati ločeno, potrebno je uporabiti kombinacijo, kar privede do več regulacijskih shem za doseg referenčne lege. Ti pristopi so splošni in jih je mogoče uporabiti pri različnih kinematikah mobilnih robotov. V tem poglavju ju bomo ponazorili s primeri na diferencialnemu in Ackermannovemu pogonu.

3.2.1 Vodenje orientacije

Pri ravninskem gibanju je za izvedbo gibanja z želeno orientacijo potrebno vodenje orientacije. Vodenje orientacije je pomembno tudi zaradi prisotnosti neholonomičnih omejitev, ki onemogočajo premik kolesnega robota v določenih smereh. Čeprav orientacije ni možno voditi neodvisno od gibanja naprej, lahko na težavo pogledamo tudi z vidika klasičnega vodenja, kar nam pokaže, kako ojačenja regulatorja vplivajo na klasična merila uspešnosti pri povratnozančnem vodenju orientacije.

Predpostavimo, da je orientacija kolesnega robota v nekem času t enaka $\varphi(t)$, referenčna ali želena orientacija pa $\varphi_{ref}(t)$. Pogrešek vodenja lahko določimo kot

$$e_{\varphi}(t) = \varphi_{ref}(t) - \varphi(t)$$

V vsakem sistemu vodenja je potrebna regulirna veličina, ki lahko spremeni ali vpliva na regulirano veličino, kar je v našem primeru orientacija. Cilj vodenja je izničiti pogrešek vodenja. Običajno mora rešitev hitro konvergirati proti 0, pri čemer morajo biti upoštevane nekatere dodatne zahteve, kot so poraba energije, obremenitev aktuatorja ter robustnost sistema ob prisotnosti motenj, šuma, parazitske dinamike itd. Običajno načrtovanje vodenja začnemo z modelom sistema, ki ga želimo voditi. V nadaljevanju bomo zapisali kinematični model, natančneje enačbo za opis njegove orientacije.

Vodenje orientacije diferencialnega pogona

Kinematiko diferencialnega pogona podaja (2.2), kjer tretja enačba določa potek orientacije

$$\dot{\varphi}(t) = \omega(t) \tag{3.1}$$

Z vidika vodenja enačba (3.1) opisuje sistem z regulirno veličino $\omega(t)$ in ima integrirni značaj (njegov pol leži v izhodišču kompleksne ravnine s). Znano je, da lahko preprost proporcionalni regulator izniči pogrešek pri vodenju integrirnega procesa. Regulacijski zakon zapišemo kot

$$\omega(t) = K(\varphi_{ref}(t) - \varphi(t)) \tag{3.2}$$

kjer je ojačenje regulatorja K poljubna pozitivna konstanta. Regulacijski zakon (3.2) kaže, da je kotna hitrost platforme $\omega(t)$ sorazmerna pogrešku orientacije robota. S pomočjo enačb (3.1) in (3.2) lahko zapišemo dinamiko regulacijske zanke za orientacijo

$$\dot{\varphi}(t) = K\left(\varphi_{ref}(t) - \varphi(t)\right)$$

kar določa zaprtozančno prenosno funkcijo vodenega sistema

$$G_{cl}(s) = \frac{\phi(s)}{\phi_{ref}(s)} = \frac{1}{\frac{1}{K}s + 1}$$

kjer sta $\phi(s)$ in $\phi_{ref}(s)$ Laplaceovi transformaciji $\varphi(t)$ in $\varphi_{ref}(t)$. Prenosna funkcija $G_{cl}(s)$ je prvega reda, torej se orientacija eksponentno približuje konstantni referenci (s časovno konstanto $T = \frac{1}{K}$) in ima ojačenje 1, zato v ustaljenem stanju ni pogreška orientacije.

Regulator (3.2) torej povzroči, da se zaprtozančna prenosna funkcija obnaša kot sistem prvega reda. Včasih je zaželena prenosna funkcija drugega reda $G_{cl}(s) = \frac{\phi(s)}{\phi_{ref}(s)}$, saj omogoča več svobode pri načrtovanju poteka med prehodnim pojavom. Razvoj regulatorja začnemo z definiranjem kotnega pospeška platforme $\dot{\omega}(t)$, ki je sorazmeren pogrešku orientacije robota

$$\dot{\omega}(t) = K(\varphi_{ref}(t) - \varphi(t)) \tag{3.3}$$

Dobljen regulirani sistem

$$\dot{\omega} = \ddot{\varphi}(t) = K(\varphi_{ref}(t) - \varphi(t))$$

s prenosno funkcijo

$$G_{cl}(s) = \frac{\phi(s)}{\phi_{ref}(s)} = \frac{K}{s^2 + K}$$

je sistem drugega reda z lastno frekvenco $\omega_n = \sqrt{K}$ in koeficientom dušenja $\zeta = 0$. Tak sistem je mejno stabilen, njegovi oscilatorni odzivi pa so nesprejemljivi. Dušenje sistema dosežemo z dodatnim členom v regulatorju (3.3)

$$\dot{\omega}(t) = K_1(\varphi_{ref}(t) - \varphi(t)) - K_2\dot{\varphi}(t) \tag{3.4}$$

kjer sta K_1 in K_2 poljubni pozitivni ojačenji regulatorja. Z upoštevanjem (3.1) in (3.4) dobimo zaprtozančno prenosno funkcijo

$$G_{cl}(s) = \frac{\phi(s)}{\phi_{ref}(s)} = \frac{K_1}{s^2 + K_2 s + K_1}$$

kjer je $\omega_n = \sqrt{K_1}$ lastna frekvenca, $\zeta = \frac{K_2}{2\sqrt{K_1}}$ koeficient dušenja in sta $s_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2}$ zaprtozančna pola. Vidimo, da z izbiro K_1 in K_2 vplivamo na primerno dušenje zaprtozančnega sistema.

Vodenje orientacije Ackermannovega pogona

Regulacijo orientacije Ackermannovega pogona lahko zasnujemo podobno kot v primeru diferencialnega pogona. Edina razlika se pojavi zaradi drugačnega kinematičnega modela (2.15) za orientacijo, ki ga zapišemo kot

$$\dot{\varphi} = \frac{v_r(t)}{d}\tan(\alpha(t)) \tag{3.5}$$

Regulirna veličina α je sorazmerna velikosti pogreška orientacije

$$\alpha(t) = K(\varphi_{ref}(t) - \varphi(t)) \tag{3.6}$$

S pomočjo (3.5) in (3.6) zapišemo diferencialno enačbo, s katero lahko opišemo dinamiko pogreška orientacije

$$\dot{\varphi}(t) = \frac{v_r(t)}{d} \tan\left(K(\varphi_{ref}(t) - \varphi(t))\right)$$

Sistem je torej nelinearen. Za majhne kote $\alpha(t)$ in konstantno hitrost zadnjih koles $v_r(t) = V$ dobimo linearni približek modela

$$\dot{\varphi}(t) = \frac{V}{d} K\left(\varphi_{ref}(t) - \varphi(t)\right)$$

ki ga lahko zapišemo kot prenosno funkcijo

$$G_{cl}(s) = \frac{\phi(s)}{\phi_{ref}(s)} = \frac{1}{\frac{d}{VK}s + 1}$$

Podobno kot v primeru diferencialnega pogona se pogrešek orientacije eksponentno približuje 0 (pri konstantni referenčni orientaciji) in s časovno konstanto $T = \frac{d}{VK}$.

Če je želena povratnozančna prenosna funkcija (orientacije) drugega reda, lahko uporabimo podoben pristop kot pri diferencialnem pogonu. Regulacijski zakon, ki ga podaja enačba (3.4), lahko uporabimo tudi na Ackermannovemu pogonu

$$\dot{\alpha}(t) = K_1(\varphi_{ref}(t) - \varphi(t)) - K_2\dot{\varphi}(t) \tag{3.7}$$

Ob predpostavki, da je hitrost konstantna ($v_r(t) = V$) ter so koti α majhni, zapišemo linearno aproksimacijo enačbe (3.5) kot

$$\dot{\varphi}(t) = \frac{V}{d}\alpha(t) \tag{3.8}$$

Kot $\alpha(t)$ iz (3.8) vstavimo v (3.7)

$$\ddot{\varphi}(t) = K_1\frac{V}{d}\left(\varphi_{ref}(t) - \varphi(t)\right) - K_2\frac{V}{d}\dot{\varphi}(t)$$

Tako dobimo zaprtozančno prenosno funkcijo

$$G_c(s) = \frac{\phi(s)}{\phi_{ref}(s)} = \frac{K_1\frac{V}{d}}{s^2 + K_2\frac{V}{d}s + K_1\frac{V}{d}}$$

Dobljeni odziv orientacije robota z lastno frekvenco $\omega_n = \sqrt{K_1\frac{V}{d}}$ dušimo s koeficientom dušenja $\zeta = \frac{K_2}{2}\sqrt{\frac{V}{dK_1}}$.

3.2.2 Vodenje gibanja naprej

Z vodenjem gibanja naprej mislimo na algoritme, ki določajo translatorno hitrost mobilnega robota $v(t)$, da dosežejo določen cilj vodenja. Vendar za vodenje mobilnega robota ne moremo uporabiti samo vodenja gibanja naprej. Kot primer vzemimo diferencialni pogon. Z vodenjem orientacije, kjer spreminjamo kotno

hitrost $\omega(t)$, robot brez težav doseže ciljno orientacijo. S pomočjo vodenja naprej pa robot v splošnem ne more doseči želene pozicije, razen v primeru, ko je robot že na začetku usmerjen proti cilju. To pomeni, da je vodenje gibanja naprej neločljivo povezano z vodenjem orientacije.

Torej je potrebno za doseg cilja voditi translatorno in kotno hitrost. V primeru sledenja trajektoriji, le-ta bolj ali manj določa hitrost, medtem ko se pri vodenju v referenčno lego hitrost zmanjša, ko se robot približa cilju. Primerna izbira regulatorja je proporcinalna odvisnost hitrosti glede na razdaljo do referenčne točke $(x_{ref}(t), y_{ref}(t))$

$$v(t) = K\sqrt{(x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2} \tag{3.9}$$

Referenčni položaj je lahko konstanten ali pa se spreminja glede na neko referenčno trajektorijo. Regulator (3.9) ima vsekakor nekaj omejitev, pa tudi v primeru zelo velikih ali zelo majhnih razdalj do referenčne točke je potrebna posebna obravnava:

- Če je razdalja do referenčne točke velika, je velika tudi regulirna veličina (3.9). Priporočljivo je omejiti regulirno veličino za največjo hitrost. V praksi omejitve narekujejo omejitve pogona, vozne razmere podlage, ukrivljenost poti itd.
- Če je razdalja do referenčne točke zelo majhna, lahko robot prevozi referenčno točko (zaradi šuma ali nepopolnega modela vozila). Ko se robot oddaljuje od referenčne točke, se razdalja povečuje in robot pospešuje v skladu z enačbo (3.9). S to težavo se bomo spoprijeli po združitvi regulatorja gibanja naprej in regulatorja orientacije.

Hitrost je neločljivo povezana s pospeškom. Slednji je v praktičnih situacijah vedno omejen zaradi omejenih sil in navorov, ki jih proizvajajo aktuatorji. To je potrebno upoštevati pri načrtovanju vodenja gibanja naprej. Ena od možnosti je omejitev pospeška. Običajno je dovolj, da na izhodu regulatorja $v(t)$ uporabimo nizkoprepustni filter, preden se regulirna veličina posreduje robotu v obliki signala $v^*(t)$. V ta namen lahko uporabimo najpreprostejši filter prvega reda z enosmernim ojačenjem 1, podan z diferencialno enačbo

$$\tau_f \dot{v}^*(t) + v^*(t) = v(t)$$

ali z enakovredno prenosno funkcijo

$$G_f(s) = \frac{V^*(s)}{V(s)} = \frac{1}{\tau_f s + 1}$$

kjer je τ_f časovna konstanta filtra.

V primeru 3.1 je uporabljen preprost algoritem vodenja, ki pripelje robota z Ackermannovim pogonom v referenčno točko. Algoritem vsebuje tako vodenje orientacije kot vodenje gibanja naprej.

Primer 3.1

Napišite algoritem za vodenje trikolesnega robota s pogonom na zadnjem paru koles na referenčno pozicijo $x_{ref} = 4\,\mathrm{m}$ in $y_{ref} = 4\,\mathrm{m}$. Robota vodimo z zasukom prednjega krmilnega kolesa α in s hitrostjo pogonskih koles v_r. Medosna razdalja je $d = 0{,}1\,\mathrm{m}$, začetna lega vozila pa $[x(0), y(0), \varphi(0)] = [1\,\mathrm{m}, 0\,\mathrm{m}, -\pi]$. Algoritem vodenja mora upoštevati omejitvi vozila $v_{max} = 0{,}8\,\mathrm{m/s}$ in $\alpha_{max} = \frac{\pi}{4}$.

Napišite algoritem vodenja in ga preizkusite na simulaciji kinematike vozila z uporabo Eulerjeve integracijske metode.

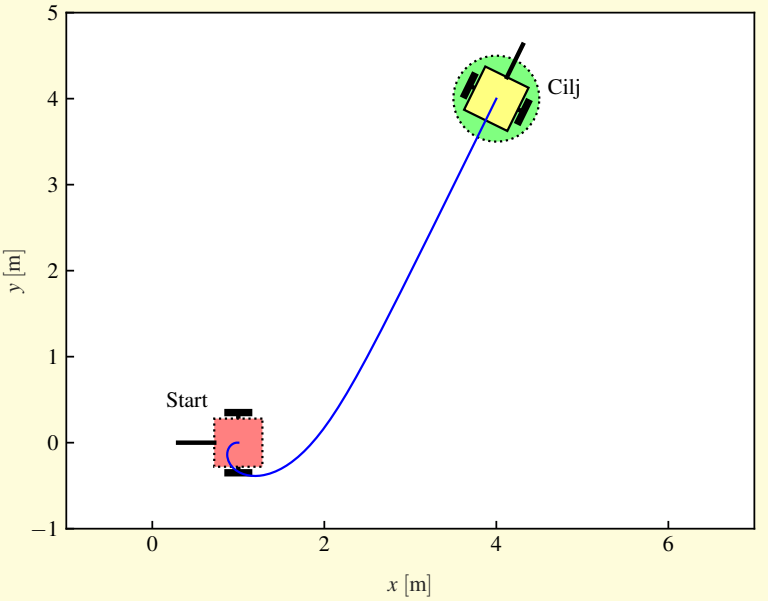
Rešitev

V tem primeru je mogoče hkrati voditi orientacijo in gibanje naprej. Zapis rešitve v programskem okolju Matlab je podan v programu 3.1. Rezultati simulacije so prikazani na slikah 3.1 in 3.2, iz katerih je razvidno, da vozilo doseže referenčno točko in se tam ustavi. Na sliki 3.2 so regulirne veličine omejene s fizičnimi omejitvami vozila.

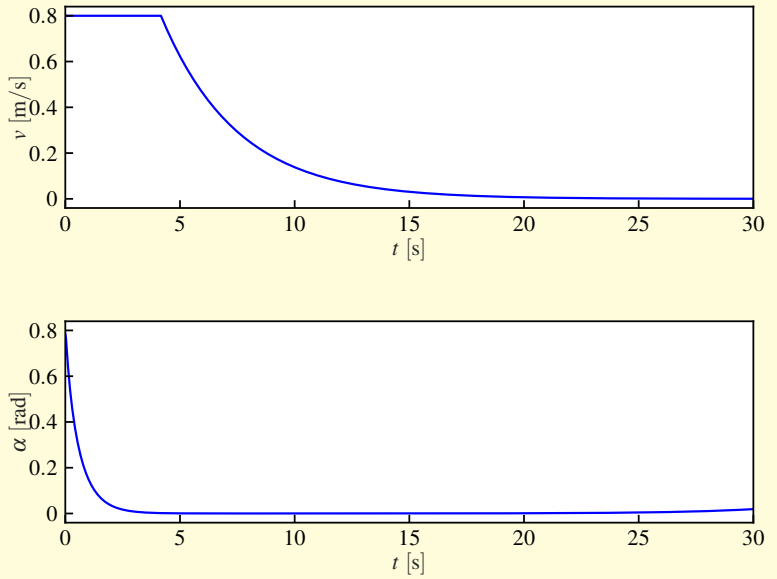
Program 3.1

./src/ctr/example_ackerman_control_point.m

```
Ts = 0.03; % Računski korak
t = 0:Ts:30; % Čas simulacije
d = 0.1; % Medosna razdalja
xyRef = [4; 4]; % Referenčni položaj
q = [1; 0; -pi]; % Začetna lega

for k = 1:length(t)
    phi_ref = atan2(xyRef(2)-q(2), xyRef(1)-q(1)); %Referenčna orientacija
    qRef = [xyRef; phi_ref];

    e = qRef - q; %Pogrešek

    % Regulator
    v     = 0.3*sqrt(e(1)^2+e(2)^2);
    alpha = 0.2*e(3);

    % Omejitve vozila
    if abs(alpha)>pi/4, alpha = pi/4*sign(alpha); end
    if abs(v)>0.8, v = 0.8*sign(v); end

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); v/d*tan(alpha)];
    noise = 0.00; % Spremeni standardno deviacijo šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
end
```



Slika 3.1: Pot robota do ciljne pozicije iz primera 3.1



Slika 3.2: Regulirni signali iz primera 3.1

Primer 3.2

Za diferencialni pogon rešite enako nalogo kot v primeru 3.1, pri čemer je največja hitrost vozila $v_{max} = 0{,}8\,\mathrm{m/s}$.

Rešitev

Ustrezno prilagodite regulator iz primera 3.1 in popravite kinematiko v simulaciji.

3.2.3 Osnovni pristopi

V nadaljevanju bomo predstavili nekaj uporabnih pristopov k vodenju kolesnega mobilnega robota v referenčno lego. Omenjeni pristopi na različne načine združujejo predhodno predstavljeno vodenje orientacije in vodenje gibanja naprej (poglavji 3.2.1 in 3.2.2) ter tako omogočajo uspešno vodenje kolesnih mobilnih robotov v referenčno lego. Ponazorjeni bodo na robotu z diferencialnim pogonom, lahko pa jih prilagodimo tudi drugim vrstam kolesnih robotov.

Vodenje v referenčno pozicijo

V tem primeru mora robot priti v referenčno (končno) pozicijo, pri tem pa ni predpisana končna orientacija, torej je lahko poljubna. Da robot prispe do referenčne točke, moramo nenehno voditi njegovo orientacijo v smeri proti referenčni točki. To smer označimo z φ_r (slika 3.3) in jo lahko enostavno določimo s pomočjo geometrijskih relacij

$$\varphi_r(t) = \arctan \frac{y_{ref} - y(t)}{x_{ref} - x(t)}$$

Vodenje kotne hitrosti $\omega(t)$ je tako določeno kot

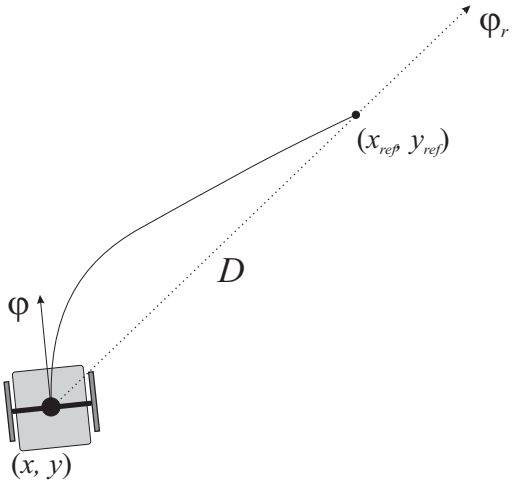
$$\omega(t) = K_1(\varphi_r(t) - \varphi(t)) \tag{3.10}$$

kjer je K_1 pozitivno ojačenje regulatorja. Osnovno vodenje je podobno kot v primeru 3.1 in je prikazano na sliki 3.3. Najprej s pomočjo enačbe (3.9) določimo translatorno hitrost robota

$$v(t) = K_2\sqrt{(x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2} \tag{3.11}$$

Kot smo že omenili, je potrebno v začetni fazi omejiti največjo hitrost zaradi številnih fizičnih omejitev. Upoštevati moramo zlasti omejitve hitrosti in pospeška.

Regulacijski zakon (3.11) skriva tudi potencialno nevarnost, ko se robot približa ciljni legi. Ukaz za hitrost (3.11) je vedno pozitiven, zato se lahko robot med



Slika 3.3: Vodenje v referenčno lego

zaviranjem proti končni poziciji pomotoma zapelje preko nje. Težava je v tem, da se bo takrat regulirna veličina za hitrost povečala, ker se poveča razdalja med robotom in referenco (robot se oddaljuje od cilja). Druga težava je v tem, da prečkanje referenčne točke prav tako obrne referenčno orientacijo, kar vodi do hitrega vrtenja robota. Obstaja nekaj preprostih rešitev:

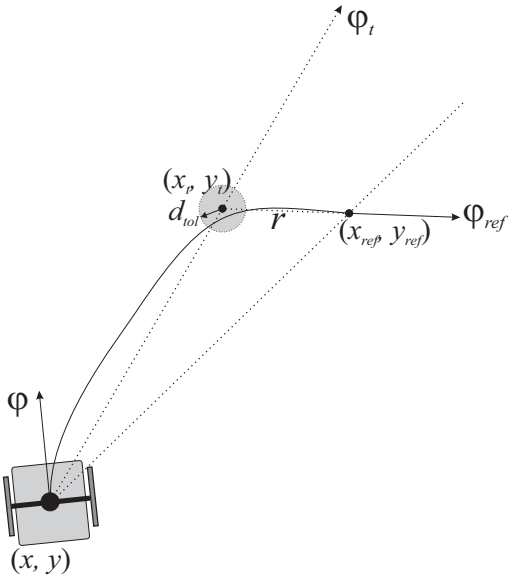
- Ko robot prevozi referenčno točko, se pogrešek orientacije nenadoma spremeni za 180°. Zato bo algoritem preveril, ali njegova absolutna vrednost presega 90°. Pogrešek orientacije se bo nato povečal ali zmanjšal za 180° (tako da se nahaja v intervalu $[-180°, 180°]$), preden vstopi v regulator. Poleg tega izhod regulatorja (3.11) spremeni svoj predznak. Torej nadgrajeni različici regulacijskih zakonov (3.10) in (3.11) zaobideta omenjene težave

$$\begin{aligned} e_\varphi(t) &= \varphi_r(t) - \varphi(t) \\ \omega(t) &= K_1 \arctan(\tan(e_\varphi(t))) \\ v(t) &= K_2 \sqrt{(x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2} \operatorname{sgn}(\cos(e_\varphi(t))) \end{aligned} \tag{3.12}$$

- Ko robot doseže določeno okolico referenčne točke, se konča faza približevanja in regulirne veličine za hitrosti postanejo ničelne. Ta mehanizem za popolno zaustavitev vozila je potrebno uporabiti tudi pri spremenjenem regulacijskem zakonu (3.12), zlasti v primeru šumnih meritev.

Vodenje v referenčno lego z vpeljavo vmesne točke

Naslednji algoritem vodenja je enostaven za izvedbo, saj uporablja preprost regulator, določen z (3.11) in (3.10), ki robota pripelje v želeno referenčno točko. Ker imamo poleg $(x_{ref}(t), y_{ref}(t))$ tudi zahtevo za referenčno smer φ_{ref}, moramo dodati vmesno točko, ki bo oblikovala trajektorijo tako, da dobimo pravilno



Slika 3.4: Vodenje v referenčno lego z vpeljavo vmesne točke

končno orientacijo. Vmesna točka (x_t, y_t) je od referenčne točke oddaljena za razdaljo r, pri čemer smer od vmesne točke proti referenčni sovpada z referenčno orientacijo, kot prikazuje slika 3.4. Vmesno točko določimo z

$$x_t = x_{ref} - r\cos\varphi_{ref}$$
$$y_t = y_{ref} - r\sin\varphi_{ref}$$

Algoritem vodenja je sestavljen iz dveh faz. V prvi fazi vodimo robota proti vmesni točki. Ko se ji dovolj približa (kar preverja pogoj $\sqrt{(x-x_t)^2+(y-y_t)^2} < d_{tol}$), algoritem preide v drugo fazo, kjer vodimo robota proti referenčni točki. Ta pristop zagotavlja, da robot pride na referenčno pozicijo z zahtevano orientacijo (v referenčni legi je možen zelo majhen pogrešek orientacije). Možne so različne variacije algoritma in vpeljava več vmesnih točk za boljše delovanje.

Predstavljen algoritem je zelo enostaven in uporaben na mnogih področjih. Glede na aplikacijo je potrebno izbrati ustrezno razdaljo r in tolerančno področje d_{tol}.

Primer 3.3

Za robota z diferencialnim pogonom napišite algoritem vodenja v referenčno lego $[x_{ref}, y_{ref}, \varphi_{ref}] = [4\,\mathrm{m}, 4\,\mathrm{m}, 0°]$ z vpeljavo vmesne točke. Poiščite ustrezne vrednosti parametrov r in d_{tol}. Začetna lega vozila je $[x(0), y(0), \varphi(0)] = [1\,\mathrm{m}, 0\,\mathrm{m}, 100°]$.

Preizkusite algoritem vodenja s pomočjo simulacije kinematike vozila z diferencialnim pogonom.

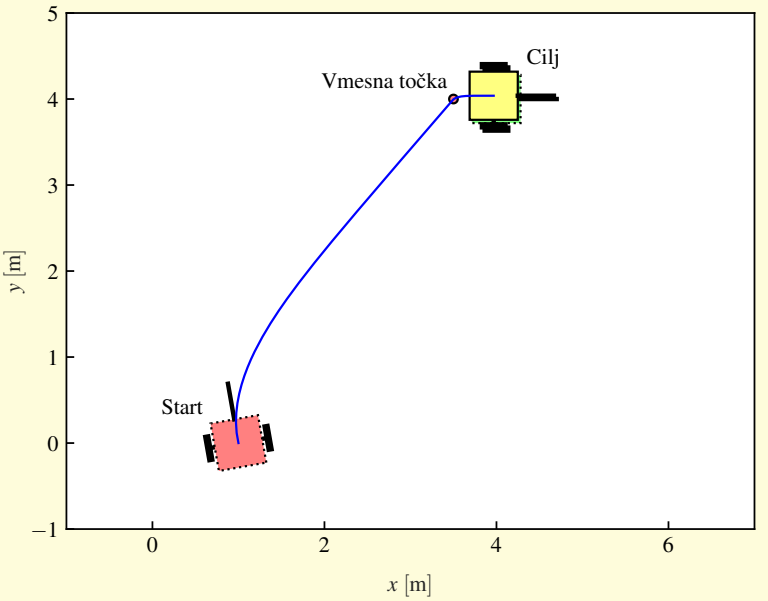
Rešitev

Matlab koda možne rešitve je podana v programu 3.2. Rezultati simulacije so prikazani na slikah 3.5 in 3.6.

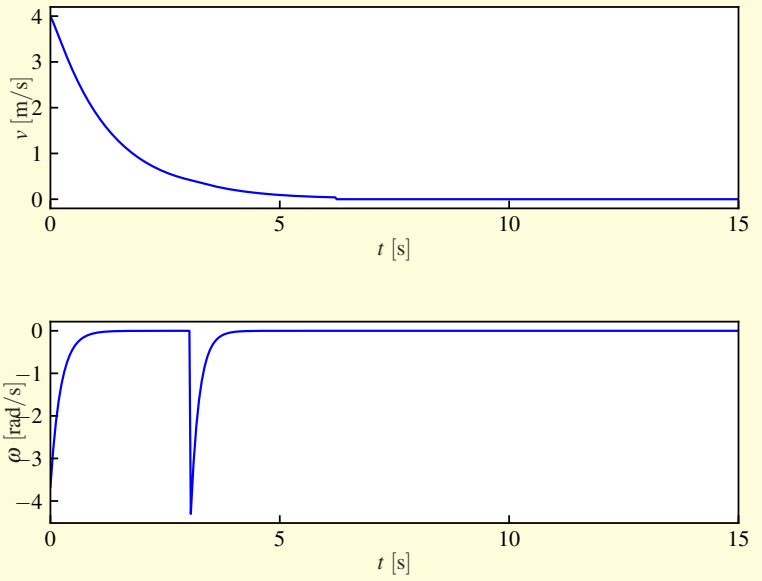
Program 3.2

./src/ctr/example_diff_control_intermediate_point.m

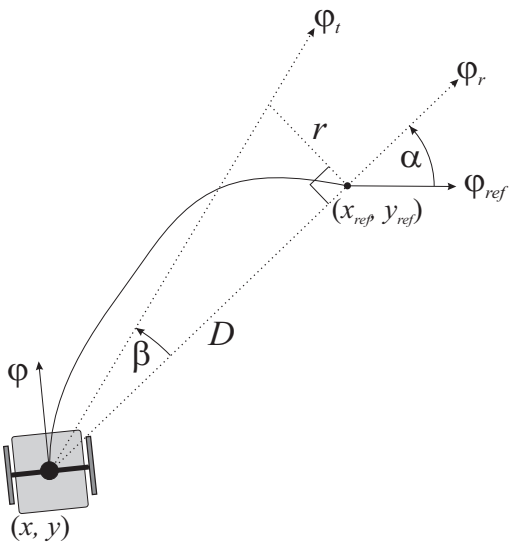
```
Ts = 0.03; % Računski korak
t = 0:Ts:15; % Čas simulacije
r = 0.5; % Razdalja vmesne točke od cilja
dTol = 0.05; % Tolerančna razdalja od vmesne točke za preklop
qRef = [4; 4; 0]; % Referenčna lega
q = [1; 0; 100/180*pi]; % Začetna lega

% Vmesna točka
xT = qRef(1) - r*cos(qRef(3));
yT = qRef(2) - r*sin(qRef(3));

state = 0; % Vodenje proti: 0 - vmesni točki, 1 - referenčni točki
for k = 1:length(t)
    D = sqrt((qRef(1)-q(1))^2 + (qRef(2)-q(2))^2);
    if D<dTol % Ustavitev v bližini cilja
        v = 0;
        w = 0;
    else
        if state==0
            d = sqrt((xT-q(1))^2+(yT-q(2))^2);
            if d<dTol, state = 1; end

            phiT = atan2(yT-q(2), xT-q(1));
            ePhi = phiT - q(3);
        else
            ePhi = qRef(3) - q(3);
        end
        ePhi = wrapToPi(ePhi);

        % Regulator
        v = D*0.8;
        w = ePhi*5;
    end

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.5: Pot robota do ciljne lege z uporabo vmesne točke iz primera 3.3



Slika 3.6: Regulirni signali iz primera 3.3



Slika 3.7: Vodenje v referenčno lego z vpeljavo vmesne usmeritve

Vodenje v referenčno lego z vpeljavo vmesne usmeritve

Robot se mora pripeljati iz začetne v referenčno lego, kjer sta podani pozicija (x_{ref}, y_{ref}) in orientacija φ_{ref}. Ideja algoritma z vpeljavo vmesne usmeritve je prikazana na sliki 3.7 [2]. Najprej določimo pravokotni trikotnik in postavimo referenčno točko v oglišče s pravim kotom. Kateta, dolžine $D = +\sqrt{(x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2}$, povezuje trenutni položaj robota z referenčnim in določa orientacijo φ_r, ki kaže od robota proti cilju. Druga kateta ima določeno dolžino $r > 0$, ki jo izberemo sami. Kot med kateto D in hipotenuzo označimo z $\beta(t)$. Pri tem pristopu imata ključno vlogo dva kota, ki ju določimo kot

$$\alpha(t) = \varphi_r(t) - \varphi_{ref}$$

$$\beta(t) = \begin{cases} \arctan \frac{+r}{D} & ; \quad \alpha(t) > 0 \\ -\arctan \frac{r}{D} & ; \quad \text{sicer} \end{cases}$$

Upoštevamo, da imata kota $\alpha(t)$ in $\beta(t)$ vedno enak predznak (v primeru na sliki 3.7 sta oba pozitivna). Če je α ob določenem času 0, je takrat β nepomemben in je lahko njegov znak poljuben. Velike (absolutne) vrednosti α nakazujejo, da vožnja naravnost do referenčne točke ni dobra ideja, saj bo pogrešek orientacije v referenčni legi velik. Zato je potrebno zmanjšati (absolutno) vrednost kota α. To dosežemo z vpeljavo vmesne usmeritve, ki je premaknjena iz φ_r, premik pa je vedno stran od referenčne orientacije φ_{ref}; če referenčna orientacija kaže desno (z vidika robota), se robot približa referenčni poziciji z leve strani in obratno. Medtem ko se med približevanjem cilju (absolutna) vrednost kota $\alpha(t)$ običajno zmanjšuje, se (absolutna) vrednost kota $\beta(t)$ povečuje z manjšanjem oddaljenosti od reference. To bomo izkoristili pri načrtovanju algoritma vodenja.

Podobno kot v primeru vodenja z vpeljavo vmesne točke je algoritem sestavljen iz

dveh faz. V prvi fazi (kjer je $|\alpha(t)|$ velik) vodimo orientacijo robota proti vmesni usmeritvi $\varphi_t(t) = \varphi_r(t) + \beta(t)$ (omenjen primer je prikazan na sliki 3.7). V drugi fazi, ko postaneta kota α in β enaka, trenutna referenčna orientacija preide v $\varphi_t(t) = \varphi_r(t) + \alpha(t)$. Torej v prvem delu robota vodimo v smeri referenčne pozicije, v drugem delu pa poskrbimo, da se robot pripelje na referenčno pozicijo z referenčno orientacijo. Preklop med fazama je izveden brez nezveznega skoka, saj sta obe usmeritvi v času prehoda enaki. Regulacijski zakon za orientacijo zapišemo kot

$$e_\varphi(t) = \varphi_r(t) - \varphi(t) + \begin{cases} \alpha(t) & ; \quad |\alpha(t)| < |\beta(t)| \\ \beta(t) & ; \quad \text{sicer} \end{cases}$$

$$\omega(t) = K e_\varphi(t)$$

Vidimo, da trenutna referenčna usmeritev nikoli ne kaže proti referenčni točki, ampak je vedno rahlo zamaknjena. Zamik je izbran tako, da gre kot $\alpha(t)$ proti 0. To pomeni, da referenčna usmeritev kaže proti referenčni točki in robot bo prispel do nje s pravilno referenčno orientacijo. Upoštevamo, da ta algoritem velja tudi za negativne vrednosti kotov α in β. Paziti moramo le, da so vsi koti v intervalu $(-\pi, \pi]$.

Translatorna hitrost je določena podobno kot v prejšnjem poglavju.

Primer 3.4

Za robota z diferencialnim pogonom napišite algoritem vodenja do referenčne lege $[x_{ref}, y_{ref}, \varphi_{ref}] = [4\,\text{m}, 4\,\text{m}, 0°]$ z vpeljavo vmesne usmeritve. Poiščite ustrezno vrednost parametra r. Začetna lega vozila je $[x(0), y(0), \varphi(0)] = [1\,\text{m}, 0\,\text{m}, 100°]$. Preizkusite algoritem na simulaciji kinematičnega modela.

Rešitev

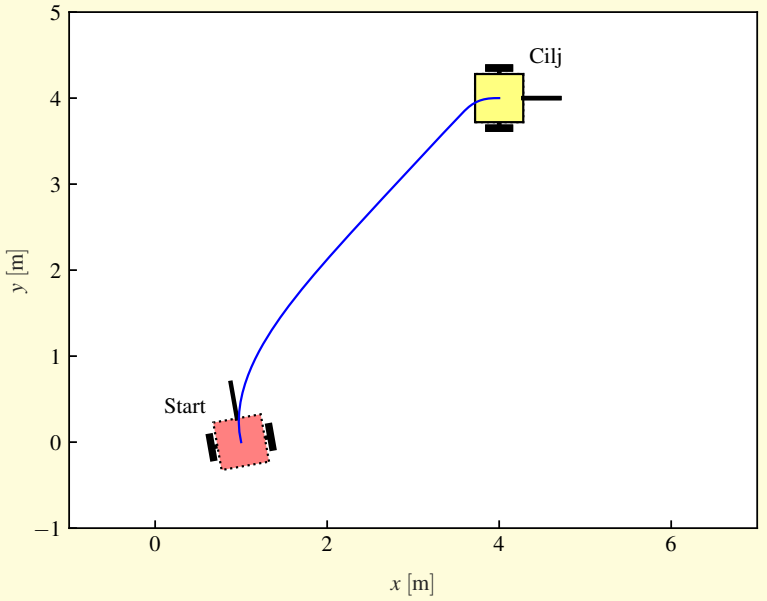
Matlab koda možne rešitve je predstavljena v programu 3.3. Rezultati simulacije so prikazani na slikah 3.8 in 3.9.

Program 3.3

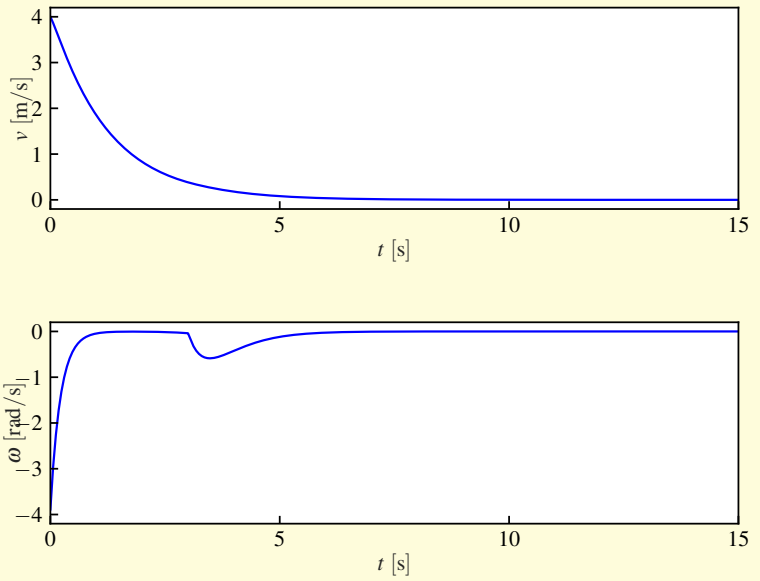
./src/ctr/example_diff_control_intermediate_direction.m

```
1  Ts = 0.03; % Računski korak
2  t = 0:Ts:15; % Čas simulacije
3  r = 0.2; % Parameter razdalje
4  qRef = [4; 4; 0]; % Referenčna lega
5  q = [1; 0; 100/180*pi]; % Začetna lega
6
7  for k = 1:length(t)
8      % Izračunaj zamik zaradi referenčne usmeritve
9      phiR = atan2(qRef(2)-q(2), qRef(1)-q(1));
10     D = sqrt((qRef(1)-q(1))^2 + (qRef(2)-q(2))^2);
11
```

```
    alpha = wrapToPi(phiR - qRef(3));
    beta = atan(r/D);
    if alpha<0, beta = -beta; end

    % Controller
    if abs(alpha) < abs(beta)
        ePhi = wrapToPi(phiR - q(3) + alpha); % Drugi del
    else
        ePhi = wrapToPi(phiR - q(3) + beta); % Prvi del
    end
    v = D*0.8;
    w = ePhi*5;

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območje [-pi, pi]
end
```



Slika 3.8: Pot robota do ciljne lege z vpeljano vmesno usmeritvijo iz primera 3.4

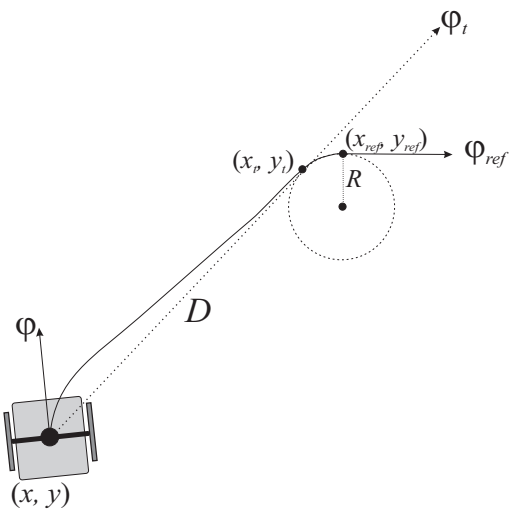


Slika 3.9: Regulirni signali iz primera 3.4

Vodenje po odsekoma zvezni poti, določena s premico in krožnim lokom

Pot, sestavljena iz daljic in krožnih lokov, je znana kot najkrajša možna pot za robote z Ackermannovim pogonom, kjer polmer krožnic predstavlja najkrajši možni polmer zavoja vozila [3–5]. Takšna pot je najkrajša tudi za robota z diferencialnim pogonom, kjer je najkrajši polmer kroga omejen na nič, kar pomeni, da se lahko robot vrti na mestu.

Osnovna ideja algoritma je prikazana na sliki 3.10. Najprej skozi referenčno točko narišemo krožnico s polmerom R, ki je tangencialen na referenčno orientacijo. O primerni dolžini polmera R bomo razpravljali kasneje. Obstajata dve rešitvi – izberemo krožnico, katere središče je bližje robotu. Ta krožnica oz. natančneje določen lok predstavlja drugi del načrtovane poti. Robot najprej sledi premici, ki se v bližini referenčne točke tangencialno poveže na krožnico. Ponovno obstajata dve rešitvi tangente in izberemo tisto, ki daje pravilno smer vožnje po krožnem loku; smer vožnje po loku je določena z referenčno orientacijo (v primeru na sliki 3.10 se bo robot peljal v smeri urinega kazalca). Rešitev preprosto izberemo s preverjanjem predznakov vektorskih produktov radialnih vektorjev (od središča krožnice do potencialne tangentne točke na njej) in tangencialnih vektorjev (od robota do potencialne tangentne točke na krožnici).



Slika 3.10: Vodenje po odsekoma zvezni poti, sestavljeni iz premice in krožnega loka

Cilj prvega dela algoritma je vodenje robota proti točki (x_t, y_t), kjer se premica dotika krožnega loka. Vzemimo preprost regulator orientacije

$$\omega(t) = K(\varphi_t(t) - \varphi(t))$$

kjer je $\varphi_t(t) = \arctan \frac{y_t - y(t)}{x_t - x(t)}$. Ko je razdalja do vmesne točke dovolj majhna, se začne druga faza, ki vključuje vožnjo vzdolž trajektorije. Regulator spremenimo v

$$\omega(t) = \frac{v(t)}{R} + K(\varphi_{tang}(t) - \varphi(t))$$

kjer je R polmer kroga, $v(t)$ želena translatorna hitrost in $\varphi_{tang}(t)$ smer tangente na krožni lok v trenutni poziciji robota. Prvi del regulatorja predstavlja predkrmiljenje, ki zagotavlja vožnjo robota po krožnem loku s polmerom R, drugi del pa povratno zanko, ki popravlja regulacijske pogreške.

Da dosežemo večjo robustnost, se referenčna pot izračuna v vsakem računskem koraku regulacijske zanke, kar zagotovi, da je robot vedno na referenčni premici ali krožnici. Končna prevožena pot se zato nekoliko razlikuje od idealne poti, sestavljene iz daljice in krožnega loka. Omenjena razlika v gibanju nastane zaradi neujemanja začetnih pogojev (orientacija robota se ne ujema popolnoma s tangento), šuma in motenj (zdrs koles ipd.).

Referenčno pot je v realnem času razmeroma enostavno določiti. Sama pot je zvezna, vendar zahtevani vhodi niso. Zaradi prehoda iz premice na krožni lok kotna hitrost robota hipoma skoči iz nič na $\frac{v(t)}{R}$. V praksi to ni možno zaradi omejenega pospeška robota, zato se pri tem prehodu pojavi nekaj sledilnega pogreška.

Takšno vodenje je primerno za primere, ko mora robot priti v referenčno lego po najkrajši poti poljubne oblike (npr. robotski nogomet). Pri robotih z omejenim

polmerom zavoja (npr. Ackermannov pogon) je najkrajša pot do ciljne lege, ko je parameter R enak najkrajšemu polmeru zavoja robota. Seveda pa to vodi v visoke vrednosti radialnih pospeškov, zato je morda zaželena večja vrednost R.

Primer 3.5

Za robota z diferencialnim pogonom napišite algoritem vodenja v referenčno lego $[x_{ref}, y_{ref}, \varphi_{ref}] = [0\,\mathrm{m}, 0\,\mathrm{m}, 0°]$. Polmer krožnice naj bo $R = 0{,}4\,\mathrm{m}$. Začetna lega vozila je $[x(0), y(0), \varphi(0)] = [-3\,\mathrm{m}, -3\,\mathrm{m}, 100°]$. Preizkusite algoritem na simulaciji kinematičnega modela.

Rešitev

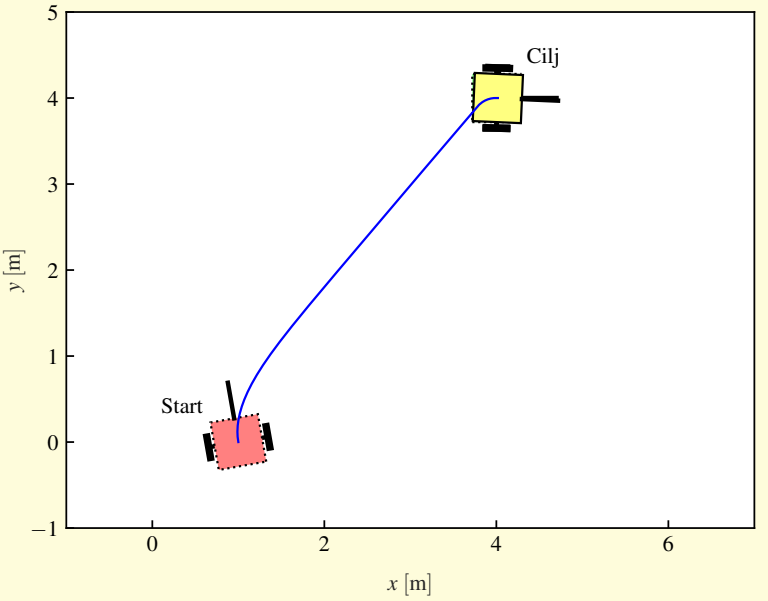
Čeprav je osnovna ideja algoritma vodenja dokaj preprosta, je izvedba nekoliko bolj zapletena, saj je potrebno izračunati ustrezna središča krožnice in tangentne točke na njej ter nekatere druge parametre. Možna rešitev je podana v programu 3.4, rezultati simulacije pa so prikazani na slikah 3.11 in 3.12.

Program 3.4

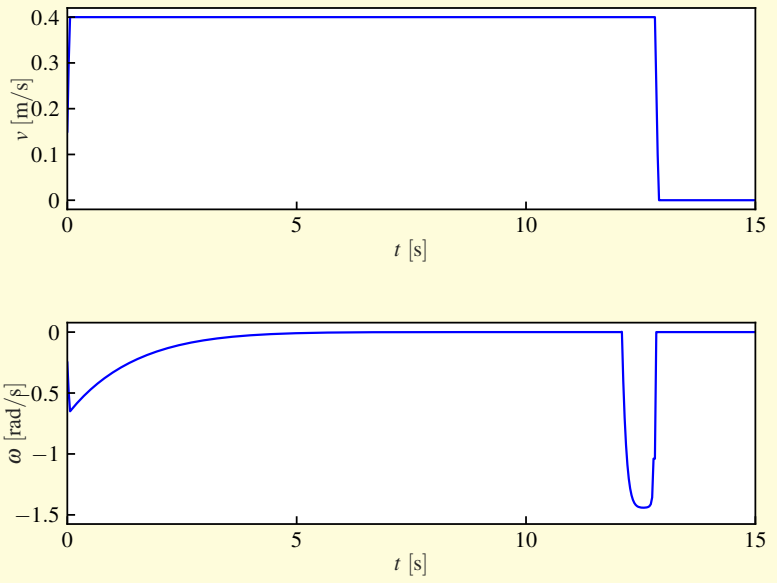
./src/ctr/example_diff_control_line_circle.m

```
Ts = 0.03; % Računski korak
t = 0:Ts:15; % Čas simulacije
r = 0.2; % Parameter razdalje
qRef = [4; 4; 0]; % Referenčna lega
q = [1; 0; 100/180*pi]; % Začetna lega

aMax = 5; % Maksimalni pospešek
vMax = 0.4; % Maksimalna hitrost
accuracy = vMax*Ts; % Točnost
curveZone = 0.6; % Radij
Rr = 0.99*curveZone/2; % Radij
slowDown = false; v = 0; vDir = 1; w = 0; % Začetna stanja
X = [0, 1; -1, 0]; % Pomožna matrika: a.'*X*b = a(1)*b(2) - a(2)*b(1)

for k = 1:length(t)
    fin = [cos(qRef(3)); sin(qRef(3))];
    D = qRef(1:2); % Ciljna točka
    S = q(1:2); % Položaj robota
    M = (D + S)/2;
    Ov = [cos(q(3)); sin(q(3))]; % Vektor orientacije
    SDv = D - S; % Vektor SD
    l2 = norm(SDv); % Razdalja

    if slowDown
        v = v - aMax*Ts; if v < 0, v = 0; end
        w = 0;
    else
        if fin.'*X*SDv > SDv.'*X*fin
            Ps = D - Rr*X.'*fin; % Center kroga
        else
            Ps = D - Rr*X*fin; % Center kroga
        end

        l = norm(Ps-S);
```

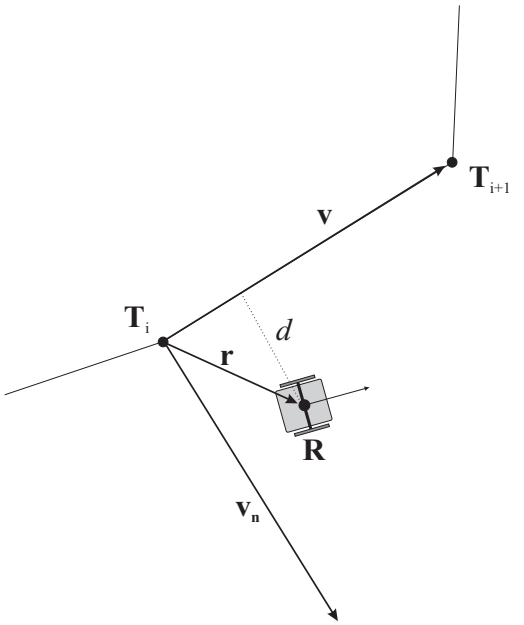
```
        if l < curveZone/2
            Dv = fin;
        else
            d = sqrt(sum((S-Ps).^2) - Rr^2);
            alpha = atan(Rr/d);
            phi = wrapTo2Pi(atan2(Ps(2)-S(2), Ps(1)-S(1)));
            U1 = S + d*[cos(phi+alpha); sin(phi+alpha)];
            U2 = S + d*[cos(phi-alpha); sin(phi-alpha)];
            if ((U1 - S).'*X*(Ps - U1)) * (fin.'*X*(Ps - D)) >= 0
                D = U1;
            else
                D = U2;
            end
            M = (D + S)/2;
            SDv = D - S;
            Dv = SDv/(norm(SDv)+eps);
        end

        if l2 > accuracy % Če položaj ni dosežen
            v = v + aMax*Ts; if v > vMax, v = vMax; end

            Ev = X*(D-S);
            DTv = X*Dv;
            if abs(DTv.'*X*Ev) < 0.000001 % Pojdi naravnost
                gamma = 0;
                Sv = SDv/(norm(SDv)+eps);
            else % Go on a circle
                C = DTv * Ev.'*X*(D - M)/(DTv.'*X*Ev) + D; % Center kroga
                if SDv.'*X*Dv > 0, a = 1; else a = -1; end
                Sv = a*X*(C-S);
                Sv = Sv/(norm(Sv)+eps);
                gamma = a*acos(Dv.'*Sv);
                if a*Sv.'*X*Dv < 0, gamma = a*2*pi - gamma; end
                l = abs(gamma*norm(S-C)); % Dolžina krivulje
            end

            if v > eps
                if Ov.'*Sv < 0, vDir = -1; else vDir = 1; end % Usmerjenost
                ePhi = acos(vDir*Sv.'*Ov); % Kotni pogrešek
                if vDir*Ov.'*X*Sv < 0, ePhi = -ePhi; end
                dt = l/v; if dt < 0.00001, dt = 0.00001; end
                w = gamma/dt + ePhi/dt*10*(1-exp(-l2/0.1)); % Kotna hitrost
            else
                w = 0;
            end
        else
            slowDown = true;
        end
    end
    u = [vDir*v; w]; % Tangencialna in kotna hitrost

    % Simulacija gibanja robota
    dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.11: Pot robota do ciljne lege na podlagi premic in krožnic iz primera 3.5



Slika 3.12: Regulirni signali iz primera 3.5



Slika 3.13: Vodenje po odsekoma zvezni poti, določeni s sekvenco točk. Referenčna pot med sosednjima točkama je daljica, ki ti dve točki povezuje.

Od vodenja v referenčno lego do vodenja po referenčni poti

Pogosto je cilj vodenja določen z zaporedjem referenčnih točk, ki naj bi jih robot prevozil. V tem primeru ne govorimo več o vodenju v referenčno lego, temveč je skozi te točke speljana referenčna pot. Za povezave med posameznimi točkami so pogosto uporabljene daljice. Cilj vodenja je, da robot prispe v vsako referenčno točko s pravilno orientacijo ter se samodejno odpelje v naslednjo referenčno točko. Ta pristop je enostaven za izvedbo in običajno zadostuje za uporabo v praksi. Njegova pomanjkljivost je nezveznost med sosednjimi daljicami, zato se tam pojavi skokovit pogrešek sledenja.

Pot je določena z zaporedjem točk $\boldsymbol{T}_i = [x_i, y_i]^T$, kjer je $i \in 1, 2, \ldots, n$ in n število točk. Na začetku mora robot slediti prvemu segmentu (daljica med točkama $\boldsymbol{T}_1$ in $\boldsymbol{T}_2$) in tako priti do $\boldsymbol{T}_2$ z orientacijo, ki jo določa vektor $\overrightarrow{\boldsymbol{T}_1\boldsymbol{T}_2}$. Ko doseže konec tega segmenta, začne slediti naslednjemu (med točkama $\boldsymbol{T}_2$ in $\boldsymbol{T}_3$) in tako naprej. Slika 3.13 prikazuje aktualen segment med točkama $\boldsymbol{T}_i$ in $\boldsymbol{T}_{i+1}$ z označenimi veličinami. Vektor $\boldsymbol{v} = \boldsymbol{T}_{i+1} - \boldsymbol{T}_i = [\Delta x, \Delta y]^T$ predstavlja smer segmenta, vektor $\boldsymbol{r} = \boldsymbol{R} - \boldsymbol{T}_i$ pa smer od točke $\boldsymbol{T}_i$ proti središču robota $\boldsymbol{R}$. Vektor $\boldsymbol{v}_n = [\Delta y, -\Delta x]$ je pravokoten na vektor $\boldsymbol{v}$.

Robot mora slediti trenutnemu segmentu, medtem ko je projekcija vektorja $\boldsymbol{r}$ na vektor $\boldsymbol{v}$ znotraj intervala, ki ga določata točki $\boldsymbol{T}_i$ in $\boldsymbol{T}_{i+1}$. Ta pogoj lahko

izrazimo na naslednji način

$$\begin{cases} \text{Sledi trenutnemu segmentu } (\boldsymbol{T}_i,\ \boldsymbol{T}_{i+1}) & \text{če} \quad 0 < u < 1 \\ \text{Sledi naslednjemu segmentu } (\boldsymbol{T}_{i+1},\ \boldsymbol{T}_{i+2}) & \text{če} \quad u > 1 \end{cases}$$

kjer je u skalarni produkt

$$u = \frac{\boldsymbol{v}^T \boldsymbol{r}}{\boldsymbol{v}^T \boldsymbol{v}}$$

Spremenljivka u nam torej pove, ali je trenutni segment še vedno aktualen ali pa je potreben prehod na naslednjega.

Pravokotna razdalja med robotom in segmentom (daljico) je določena z normalnim vektorjem $\boldsymbol{v}_n$

$$d = \frac{\boldsymbol{v}_n^T \boldsymbol{r}}{\sqrt{\boldsymbol{v}_n^T \boldsymbol{v}_n}}$$

Normiramo razdaljo d z dolžino daljice ter dobimo normirano pravokotno razdaljo d_n med robotom in daljico

$$d_n = \frac{\boldsymbol{v}_n^T \boldsymbol{r}}{\boldsymbol{v}_n^T \boldsymbol{v}_n}$$

Ko je robot na segmentu (daljici), je normirana razdalja d_n nič. Ko pa je robot na desni strani segmenta (glede na vektor $\boldsymbol{v}$), je d_n pozitivna in obratno. Normirana razdalja d_n se uporablja za določanje želene smeri vožnje robota. Če je robot na daljici ali v njeni neposredni bližini, ji mora slediti. Če pa je robot daleč stran od daljice, se mora voziti pravokotno nanjo, da (čim hitreje) prispe do nje. Referenčno smer vožnje v nekem trenutku lahko določimo kot

$$\varphi_{ref} = \varphi_{lin} + \varphi_{rot}$$

kjer je $\varphi_{lin} = \operatorname{atan2}(\Delta y, \Delta x)$ (štirikvadratna inverzna funkcija tangens je definirana v (2.11)) smer daljice in $\varphi_{rot} = \arctan(K_1 d_n)$ popravek dodatne referenčne rotacije, ki robotu omogoča, da doseže daljico. Ojačenje K_1 spreminja občutljivost dodatnega referenčnega kota φ_{rot} glede na d_n. Ker je φ_{ref} pridobljen s seštevanjem dveh kotov, je potrebno poskrbeti, da bo v veljavnem območju $[-\pi, \pi]$ (cikličnost kota).

Zaenkrat smo določili referenčno smer, ki ji mora robot slediti, kar lahko dosežemo z uporabo primernega regulacijskega algoritma. Regulacijski pogrešek je opredeljen kot

$$e_\varphi = \varphi_{ref} - \varphi$$

kjer je φ orientacija robota. Iz pogreška orientacije s pomočjo proporcionalnega regulatorja izračunamo kotno hitrost robota

$$\omega = K_2 e_\varphi$$

kjer je K_2 proporcionalno ojačenje. Podobno lahko izvedemo tudi PID regulator, kjer z integracijskim členom povečujemo hitrost približevanja robota daljici (vse manjši e_φ), z diferencialnim členom pa zmanjšamo oscilacije, ki nastanejo zaradi dodanega integracijskega člena. Translatorno hitrost robota v lahko vodimo z osnovnimi pristopi, obravnavanimi v prejšnjih poglavjih.

Primer 3.6

Napišite algoritem vodenja za robota z diferencialnim pogonom, ki naj prevozi zaporedje daljic, ki jih definirajo točke $\boldsymbol{T}_1 = [3,\ 0]$, $\boldsymbol{T}_2 = [6,\ 4]$, $\boldsymbol{T}_3 = [3,\ 4]$, $\boldsymbol{T}_4 = [3,\ 1]$ in $\boldsymbol{T}_5 = [0,\ 3]$. Poiščite ustrezne vrednosti parametrov K_1 in K_2. Začetna lega vozila je $[x(0), y(0), \varphi(0)] = [5\,\mathrm{m}, 1\,\mathrm{m}, 108°]$. Preizkusite algoritem na simulaciji kinematičnega modela.

Rešitev

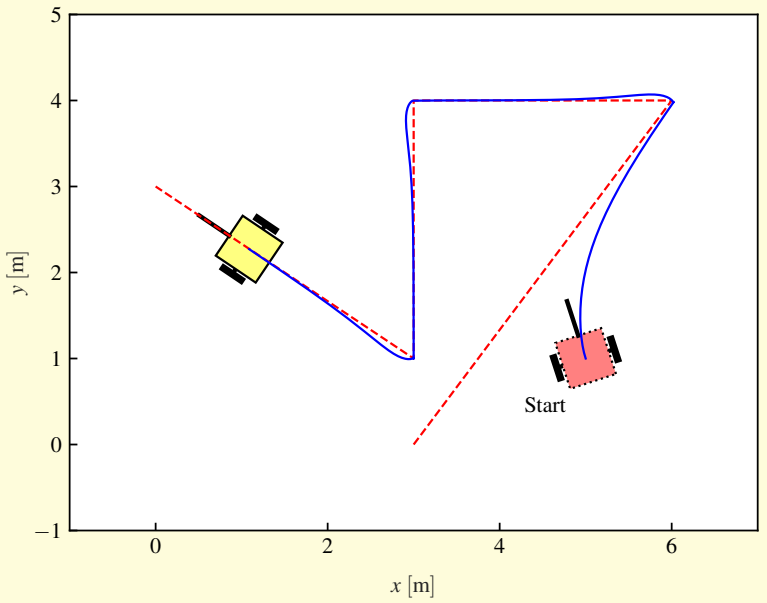
Matlab koda možne rešitve je podana v programu 3.5. Referenčna pot in dejanska trajektorija gibanja robota sta prikazani na sliki 3.14, regulirni signali pa na sliki 3.15.

Program 3.5

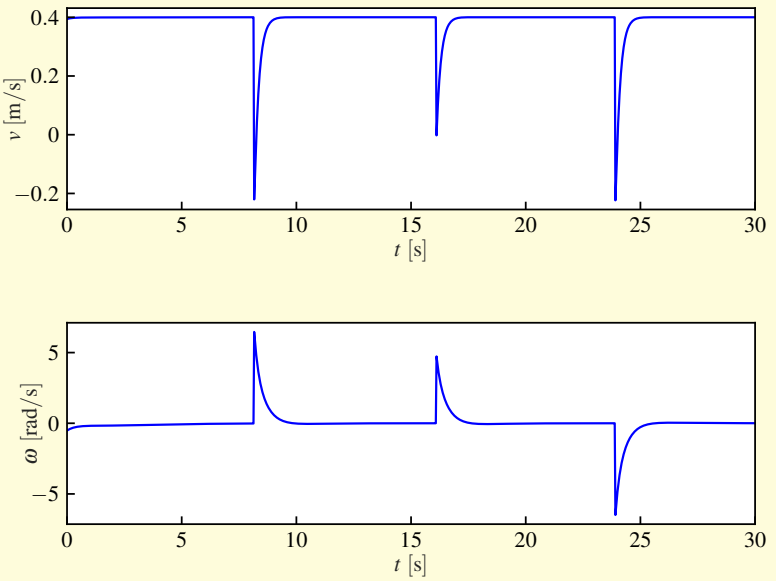
./src/ctr/example_diff_point_sequence.m

```
Ts = 0.03; % Računski korak
t = 0:Ts:30; % Čas simulacije
T = [3, 0; 6, 4; 3, 4; 3, 1; 0, 3].'; % Točke referenčnih daljic
q = [5; 1; 0.6*pi]; % Začetna lega

i = 1; % Indeks prve točke
for k = 1:length(t)
    % Referenčna daljica
    dx = T(1,i+1) - T(1,i);
    dy = T(2,i+1) - T(2,i);

    v = [dx; dy]; % Usmeritveni vektor referenčne daljice
    vN = [dy; -dx]; % Vektor ortogonalne usmeritve
    r = q(1:2) - T(:,i);
    u = v.'*r/(v.'*v);

    if u>1 && i<size(T,2)-1 % Pogoj za preklop na naslednjo daljico
        i = i + 1;
        dx = T(1,i+1) - T(1,i);
        dy = T(2,i+1) - T(2,i);
        v = [dx; dy];
        vN = [dy; -dx];
        r = q(1:2) - T(:,i);
    end

    dn = vN.'*r/(vN.'*vN); % Normirana ortogonalna razdalja

    phiLin = atan2(v(2), v(1)); % Usmeritev premice daljice
    phiRot = atan(5*dn); % Če smo daleč od premice, potem je potreben
    % dodaten zasuk, da se usmerimo proti premici. Če smo na levi strani,
    % se obrnemo v smeri urinega kazalca, sicer v obratni smeri.
    % Ojačenje 5 poveča občutljivost.

    phiRef = wrapToPi(phiLin + phiRot);

    % Kotni pogrešek
    ePhi = wrapToPi(phiRef - q(3));

    % Regulator
    v = 0.4*cos(ePhi);
```

```
41      w = 3*ePhi;
42
43      % Simulacije gibanja robota
44      dq = [v*cos(q(3)); v*sin(q(3)); w];
45      noise = 0.00; %SL Parameter za nastavljanje šuma (npr. 0.001)
46      q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
47      q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
48  end
```



Slika 3.14: Referenčna pot in dejansko gibanje robota iz primera 3.6



Slika 3.15: Regulirni signali iz primera 3.6

Poskusite dopolniti kodo, da bo robot lahko vozil tudi vzvratno, če je $|e_\varphi| > \frac{\pi}{2}$.

3.3 Vodenje po referenčni trajektoriji

V mobilni robotiki je pot krivulja, ki jo mora robot prevoziti v prostoru posplošenih koordinat. Če je pot parametrizirana po času, torej gibanje po poti je sinhronizirano s časom, govorimo o *trajektoriji*. Kadarkoli je načrt gibanja robota znan vnaprej, lahko (referenčno) trajektorijo robota zapišemo kot časovno funkcijo v prostoru posplošenih koordinat: $\boldsymbol{q}_{ref}(t) = [x_{ref}(t), y_{ref}(t), \varphi_{ref}(t)]^T$. Iz praktičnih razlogov je trajektorija vedno definirana na končnem časovnem intervalu $t \in [0, T]$, kar pomeni, da ima referenčna trajektorija začetno in končno točko. Vodenje po referenčni trajektoriji je mehanizem, ki zagotavlja, da je trajektorija robota $\boldsymbol{q}(t)$ kljub morebitnim težavam čim bolj podobna referenčni $\boldsymbol{q}_{ref}(t)$.

3.3.1 Osnovni pristopi k vodenju po referenčni trajektoriji

Pri načrtovanju vodenja si najprej predstavljamo referenčno trajektorijo kot referenčno pozicijo, ki se v vsakem računskem koraku regulatorja premakne na trenutno točko referenčne trajektorije $(x_{ref}(t), y_{ref}(t))$. V ta namen uporabimo vodenje do referenčne pozicije z regulacijskima zakonoma (3.10) in (3.11). Pozorni moramo biti na to, da se robot čimbolj približa namišljeni referenčni točki. Pri majhni hitrosti in šumni meritvi pozicije se lahko zgodi, da se meritev pozicije robota znajde pred trajektorijo. Zato je v takšnih situacijah izredno pomembno, da pravilno ukrepamo, npr. z uporabo posodobljenega regulacijskega zakona (3.12).

Ta pristop je nekoliko problematičen zaradi dejstva, da je tu povratna zanka bolj obremenjena in so zato potrebna sorazmerno velika ojačenja regulatorja, da bi bili regulacijski pogreški majhni. Posledično je omenjen pristop dovzeten za motnje v regulacijski zanki. Zatorej je koristno vpeljati predkrmiljenje, kar bo predstavljeno v poglavju 3.3.2.

Primer 3.7

Trikolesni robot s pogonom na zadnjih kolesih iz primera 3.1 naj bo voden tako, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Začetna lega vozila je $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. Napišite dva algoritma vodenja in ju preizkusite na simulaciji kinematičnega modela:

- Prvi algoritem naj uporabi osnovna regulacijska zakona (3.10) in (3.11).
- Drugi algoritem naj uporabi nadgrajen regulacijski zakon (3.12).

Rešitev

S spreminjanjem vrednosti spremenljivke `UpgradedLaw` lahko vodenje preklopimo med t. i. osnovnim načinom, podanim s (3.10) in (3.11), ter nadgrajenim, ki ga podaja (3.12). Rezultati primera 3.7 so prikazani na slikah 3.16 in 3.17. V prikazanem primeru osnovni in nadgrajeni regulacijski zakon delujeta enako.

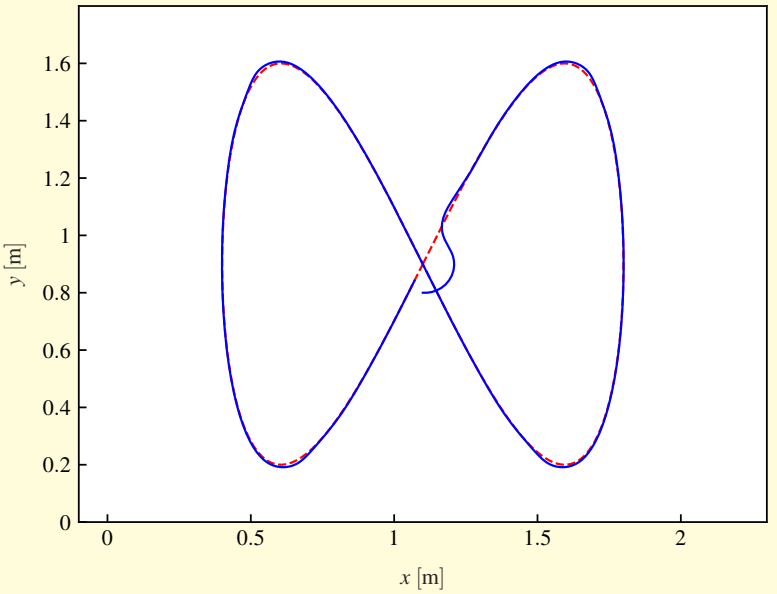
Matlab koda je podana v programu 3.6.

Program 3.6

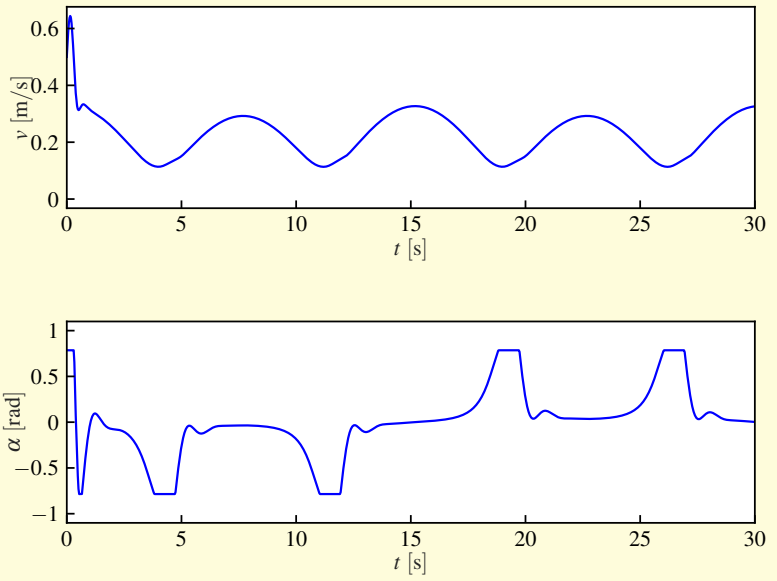
./src/ctr/example_tracking_simple_control.m

```
1 Ts = 0.03; % Računski korak
2 t = 0:Ts:30; % Čas simulacije
3 d = 0.1; % Razdalja med prednjo in zadnjo osjo
```

```
q = [1.1; 0.8; 0]; % Začetna lega

% Referenčna trajektorija
freq = 2*pi/30;
xRef = 1.1 + 0.7*sin(freq*t);
yRef = 0.9 + 0.7*sin(2*freq*t);

% Ojačenja regulatorja
Kphi = 2;
Kv = 5;

upgradedControl = true; % Ta nastavitev se lahko spremeni na false
for k = 1:length(t)
    % Referenca
    phiRef = atan2(yRef(k)-q(2), xRef(k)-q(1));
    qRef = [xRef(k); yRef(k); phiRef];

    % Pogrešek glede na trenutno referenčno točko
    e = qRef - q;          % Pogrešek po x, y in kotu
    e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]

    % Regulator
    alpha = e(3)*Kphi;          % Regulacija usmeritve (osnovna)
    v = sqrt(e(1)^2+e(2)^2)*Kv; % Krmiljenje (osnovno)
    if upgradedControl
        % % Če e(3) ni v območju [-pi/2, pi/2], je potrebno prišteti +/- pi
        % k e(3) in hitrost mora obrniti predznak
        v = v*sign(cos(e(3)));  % Sprememba predznaka hitrosti, če je potrebno
        e(3) = atan(tan(e(3))); % Zapis kota v območju [-pi, pi]
        alpha = e(3)*Kphi;      % Regulacija usmeritve (nadgradnja)
    end

    % Mehanske omejitve robota
    if abs(alpha)>pi/4, alpha = pi/4*sign(alpha); end
    if abs(v)>0.8, v = 0.8*sign(v); end

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); v/d*tan(alpha)];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.16: Preprosto vodenje po referenčni trajektoriji Ackermannovega pogona. Črtkana krivulja prikazuje referenčno pot, polna krivulja pa dejansko pot.



Slika 3.17: Regulirni signali preprostega vodenja po referenčni trajektoriji Ackermannovega pogona

3.3.2 Razčlenitev vodenja na predkrmiljenje in povratno zanko

Vodenje po referenčni trajektoriji je pomembno tako s praktičnega kot teoretičnega vidika, saj se po Brockettovem pogoju neholonomični sistemi ne morejo asimptotično stabilizirati okoli težišča z uporabo gladke (tj. zvezno odvedljive) časovno nespremenljive povratne zanke [1]. Dobljeni rezultat je mogoče enostavno preveriti na diferencialnem pogonu ter ga razširiti na druge kinematike, vključno z Ackermannovo. Najprej preverimo, ali je sistem brez lezenja (stanja se ne spreminjajo, če ni vzbujanja). Ker je pri diferencialnem pogonu brez vodenja ($v = 0$, $\omega = 0$) odvod vektorskega polja $\dot{\boldsymbol{q}}(t)$ enak 0, je to sistem brez lezenja. Brockett [1] je dokazal, da mora imeti tak sistem enako število vhodov in stanj, da ga lahko stabiliziramo z uporabo zvezne časovno nespremenljive povratne zanke. V primeru diferencialnega pogona je ta pogoj očitno kršen, zato je potrebno poiskati druge vrste povratnih zank. Kljub temu so popolnoma neholonomični sistemi brez lezenja še vedno vodljivi v nelinearnem smislu, zato je možno izvesti asimptotično stabilizacijo z uporabo časovno spremenljivih, nezveznih ali hibridnih regulacijskih zakonov. Omejitvi, ki jo določa Brockettov pogoj, se lahko izognemo z uvedbo drugačne strukture vodenja. V primeru vodenja po referenčni trajektoriji se zelo pogosto uporablja dvoprostostni regulator, kjer en del pripada predkrmiljenju, drugi pa povratni zanki.

Pred uvedbo predkrmiljenja in povratne zanke moramo določiti še eno pomembno lastnost sistema. Sistem je **diferencialno plosk** (angl. *differentially flat*), če obstaja nabor t. i. ploskih izhodov ter so lahko vsa stanja in vhodi sistema zapisani kot funkcije teh ploskih izhodov in končnega števila njihovih časovnih odvodov. To pomeni, da morata obstajati nelinearni funkciji $\boldsymbol{f}_x$ in $\boldsymbol{f}_u$, ki izpolnjujeta

$$\boldsymbol{x} = \boldsymbol{f}_x(\boldsymbol{z}_f, \dot{\boldsymbol{z}}_f, \ddot{\boldsymbol{z}}_f, \dots \frac{\mathrm{d}^p}{\mathrm{d}t^p}\boldsymbol{z}_f)$$
$$\boldsymbol{u} = \boldsymbol{f}_u(\boldsymbol{z}_f, \dot{\boldsymbol{z}}_f, \ddot{\boldsymbol{z}}_f, \dots \frac{\mathrm{d}^p}{\mathrm{d}t^p}\boldsymbol{z}_f)$$

kjer vektorji $\boldsymbol{x}$, $\boldsymbol{u}$ in $\boldsymbol{z}_f$ predstavljajo stanja sistema, vhode in ploske izhode, medtem ko je p končno celo število. Potrebno je omeniti, da morajo biti ploski izhodi funkcije stanj sistema, njegovih vhodov in končnega števila njihovih (vhodnih) odvodov. To pomeni, da so v splošnem ploski izhodi fiktivni – niso podobni dejanskim izhodom.

V primeru kinematičnega modela diferencialnega pogona, ki ga podaja (2.2), sta ploska izhoda dejanska izhoda sistema x in y. Oba vhoda (hitrosti) in

tretje stanje (orientacija robota) so lahko predstavljeni kot funkciji x in y ter njuni odvodi. Vemo, da si lahko $\dot{x}$ in $\dot{y}$ predstavljamo kot kartezični koordinati translatorne hitrosti robota, zato ju uporabimo v izračunu hitrosti s pomočjo Pitagorovega izreka

$$v(t) = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{3.13}$$

Zaradi neholonomičnih omejitev se kolesni robot z diferencialnim pogonom vedno vozi v smeri svoje orientacije, kar pomeni, da je tangenta orientacije enaka količniku kartezičnih komponent translatorne hitrosti

$$\varphi(t) = \arctan\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right) \tag{3.14}$$

Kotna hitrost $\omega(t)$ je določena kot časovni odvod orientacije $\varphi(t)$, ki jo podaja enačba (3.14)

$$\omega(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left[\arctan\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right)\right] = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{\dot{x}^2(t) + \dot{y}^2(t)} \tag{3.15}$$

in ni definirana le v primeru, ko je translatorna hitrost enaka 0.

Kartezični koordinati pozicije x in y sta prav tako ploska izhoda kinematičnega modela kolesnega pogona na zadnje kolo, ki ga podaja enačba (2.15). Prvi dve enačbi tega kinematičnega modela sta enaki kot pri diferencialnem pogonu (2.2), zato lahko orientacijo in hitrost zadnjega kolesa izračunamo s pomočjo (3.13) in (3.14). Tretja enačba v sistemu enačb (2.15) je

$$\dot{\varphi} = \frac{v_r(t)}{d}\tan(\alpha(t))$$

od koder sledi

$$\alpha(t) = \arctan\frac{d\dot{\varphi}(t)}{v_r(t)} \tag{3.16}$$

Vhod $\alpha(t)$ lahko zapišemo z ploskimi izhodi in njihovimi odvodi z vstavitvijo izrazov (3.13) in (3.15) v (3.16) namesto $v_r(t)$ in $\dot{\varphi}(t)$

$$\alpha(t) = \arctan\frac{d\left(\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)\right)}{\left(\dot{x}^2(t) + \dot{y}^2(t)\right)^{(3/2)}}$$

Zgornja analiza nam poda pomembno ugotovitev, da je Ackermannov pogon strukturno enak diferencialnemu. Če je določena regulirna veličina $\{v(t), \omega(t)\}$ uporabljena na robotu z diferencialnim pogonom, je dobljena trajektorija enaka, kot če bi regulirno veličino $(v_r(t),\ \alpha(t)) = (v(t),\ \arctan\frac{d\omega(t)}{v(t)})$ uporabili na robotu z Ackermannovim pogonom. Večina primerov v nadaljevanju obravnava robota z diferencialnim pogonom. Te rezultate je torej mogoče enostavno razširiti na robote z Ackermannovim pogonom in tudi na nekatere druge kinematične strukture.

Če je sistem plosk, lahko vse sistemske spremenljivke izrazimo iz ploskih izhodov brez integracije. Koristna posledica tega dejstva je, da se lahko na podlagi referenčne trajektorije analitično izračunajo zahtevane regulirne veličine. V primeru

kolesnega robota z diferencialnim pogonom enačbi (3.13) in (3.15) podajata formule za izračun referenčnih hitrosti $v_{ref}(t)$ in $\omega_{ref}(t)$ iz referenčne trajektorije, ki jo podajata $x_{ref}(t)$ in $y_{ref}(t)$

$$v_{ref}(t) = \sqrt{\dot{x}_{ref}^2(t) + \dot{y}_{ref}^2(t)^2} \tag{3.17}$$

$$\omega_{ref}(t) = \frac{\dot{x}_{ref}(t)\ddot{y}_{ref}(t) - \dot{y}_{ref}(t)\ddot{x}_{ref}(t)}{\dot{x}_{ref}^2(t) + \dot{y}_{ref}^2(t)} \tag{3.18}$$

Podobne formule lahko pridobimo tudi za druge kinematične strukture, ki so ploski sistemi.

Enačbi (3.17) in (3.18) podajata odprtozančno vodenje, ki zagotavlja, da se v idealnem primeru, kadar kinematični model robota natančno opisuje gibanje ter ni motenj, merskih napak in pogreška začetne lege, robot vozi po referenčni trajektoriji. Teh predpostavk nikoli popolnoma ne izpolnimo, zato je potrebno tudi povratnozančno vodenje. V teh primerih sta referenčni hitrosti iz enačb (3.17) in (3.18) uporabljeni v predkrmiljenju regulacijskega zakona, medtem ko je za povratnozančno vodenje mogoče uporabiti širok spekter regulacijskih zakonov. Nekatere od njih bomo obravnavali tudi v nadaljevanju.

3.3.3 Povratnozančna linearizacija

Ideja povratnozančne linearizacije je uvedba transformacije (običajno sistemskega vhoda), ki linearizira sistem med novim vhodom in izhodom. Ker je novi sistem linearen je možna uporaba kateregakoli od obstoječih linearnih načrtovalnih postopkov vodenja. Najprej moramo zagotoviti, da je sistem diferencialno plosk [6, 7]. V razdelku 3.3.2 smo pokazali, da je veliko kinematičnih struktur ploskih. Nato je postopek načrtovanja povratnozančne linearizacije sledeč:

- Izbrati moramo ustrezne ploske izhode. Njihovo število naj bo enako številu sistemskih vhodov.
- Ploske izhode odvajamo, za dobljene odvode pa je potrebno preveriti funkcijsko odvisnost od vhodov sistema. Ta korak ponavljamo, dokler se vsi vhodi (ali njihovi odvodi) ne pojavijo v odvodih ploskih izhodov. Če lahko iz tega sistema enačb izrazimo vse vhode (natančneje njihove najvišje odvode), lahko preidemo na naslednji korak.
- Rešimo sistem enačb za najvišje odvode posameznih vhodov. Za pridobitev dejanskih vhodov sistema je potrebno na njihovih odvodih uporabiti verigo integratorjev. Po drugi strani pa odvodi ploskih izhodov služijo kot novi vhodi v sistem.
- Ker je dobljeni sistem linearen, lahko na teh novih vhodih uporabimo širok nabor možnih regulacijskih zakonov.

V primeru kolesnega mobilnega robota z diferencialnim pogonom sta ploska izhoda $x(t)$ in $y(t)$. Njun prvi odvod glede na kinematični model (2.2) je

$$\dot{x} = v\cos\varphi$$
$$\dot{y} = v\sin\varphi$$

V prvih odvodih se pojavi le translatorna hitrost v, zato ponovno odvajamo

$$\ddot{x} = \dot{v}\cos\varphi - v\dot{\varphi}\sin\varphi$$
$$\ddot{y} = \dot{v}\sin\varphi + v\dot{\varphi}\cos\varphi$$

V drugih odvodih pa sta prisotni obe hitrosti (v in $\omega = \dot{\varphi}$). Zdaj je sistem enačb preurejen tako, da so drugi odvodi ploskih izhodov opisani kot funkcije najvišjih odvodov posameznih vhodov (v tem primeru sta to $\dot{v}$ in ω)

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}\cos\varphi & -v\sin\varphi\\ \sin\varphi & v\cos\varphi\end{bmatrix}\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix} = \boldsymbol{F}\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix}$$

Uvedemo matriko $\boldsymbol{F}$, ki je nesingularna, če je $v \neq 0$. Sistem enačb je torej mogoče rešiti za $\dot{v}$ in ω

$$\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix} = \boldsymbol{F}^{-1}\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}\cos\varphi & \sin\varphi\\ -\frac{\sin\varphi}{v} & \frac{\cos\varphi}{v}\end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} \tag{3.19}$$

Rešitev ω iz enačbe (3.19) je dejanski vhod robota, medtem ko je treba rešitev $\dot{v}$ integrirati, preden jo lahko uporabimo kot vhod. Novo pridobljeni linearni sistem ima vhoda $[u_1, u_2]^T = [\ddot{x}, \ddot{y}]^T$ in stanja $\boldsymbol{z} = [x, y, \dot{x}, \dot{y}]^T$ (kinematični model (2.2) ima tri stanja, četrto je posledica dodatnega integratorja). Dinamiko novega sistema lahko priročno opišemo z zapisom v prostoru stanj

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{y}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ y\\ \dot{y}\end{bmatrix} + \begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} \tag{3.20}$$

ali v matrični obliki kot

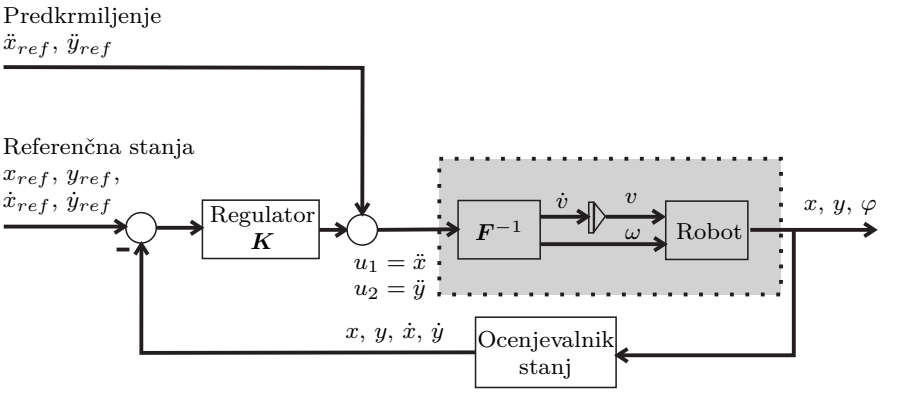
$$\dot{\boldsymbol{z}} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{u} \tag{3.21}$$

Sistem (3.21) je vodljiv, ker ima matrika vodljivosti

$$\boldsymbol{Q}_c = \begin{bmatrix}\boldsymbol{B} & \boldsymbol{A}\boldsymbol{B}\end{bmatrix} \tag{3.22}$$

polni rang in zato regulator stanj obstaja za poljubno izbran karakteristični polinom zaprte zanke. Dodatna zahteva je zasnova regulacijskega zakona, da bo robot sledil referenčni trajektoriji. Pri ploskih sistemih je za ploske izhode podana referenčna trajektorija, v tem primeru je to $x_{ref}(t)$ in $y_{ref}(t)$. Potem je mogoče zlahka pridobiti referenco za stanje sistema $\boldsymbol{z}_{ref}(t) = [x_{ref}, \dot{x}_{ref}, y_{ref}, \dot{y}_{ref}]^T$ in vhod sistema $\boldsymbol{u}_{ref} = [\ddot{x}_{ref}, \ddot{x}_{ref}]^T$. Enačbo (3.21) lahko zapišemo tudi za referenčne signale

$$\dot{\boldsymbol{z}}_{ref} = \boldsymbol{A}\boldsymbol{z}_{ref} + \boldsymbol{B}\boldsymbol{u}_{ref} \tag{3.23}$$



Slika 3.18: Povratnozančna linearizacija za sledenje referenci

Pogrešek med dejanskimi in referenčnimi stanji je opredeljen kot $\tilde{\boldsymbol{z}} = \boldsymbol{z} - \boldsymbol{z}_{ref}$. Če odštejemo (3.23) od (3.21), dobimo

$$\dot{\tilde{\boldsymbol{z}}} = \boldsymbol{A}\tilde{\boldsymbol{z}} + \boldsymbol{B}(\boldsymbol{u} - \boldsymbol{u}_{ref}) \tag{3.24}$$

Enačba (3.24) opisuje dinamiko pogreška stanj. Ta dinamika mora biti stabilna in primerno hitra. Dinamiko zaprte zanke lahko dosežemo s predpisanimi zaprtozančnimi poli. Kot smo že pokazali, je par $(\boldsymbol{A}, \boldsymbol{B})$ vodljiv in tako lahko s pravilno izbiro konstantne matrike ojačenj regulatorja $\boldsymbol{K}$ (dimenzije 2×4) dosežemo poljubne lokacije zaprtozančnih polov na levi strani kompleksne ravnine s. Enačbo (3.24) lahko preuredimo kot

$$\dot{\tilde{\boldsymbol{z}}} = (\boldsymbol{A}-\boldsymbol{B}\boldsymbol{K})\tilde{\boldsymbol{z}}+\boldsymbol{B}\boldsymbol{K}\tilde{\boldsymbol{z}}+\boldsymbol{B}(\boldsymbol{u}-\boldsymbol{u}_{ref}) = (\boldsymbol{A}-\boldsymbol{B}\boldsymbol{K})\tilde{\boldsymbol{z}}+\boldsymbol{B}(\boldsymbol{K}\tilde{\boldsymbol{z}}+\boldsymbol{u}-\boldsymbol{u}_{ref}) \tag{3.25}$$

Če je zadnji člen enačbe (3.25) enak 0, pogreški stanj konvergirajo proti 0 s predpisano dinamiko, ki jo podaja zaprtozančna matrika $(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})$. Da bo zadnji izraz 0, moramo definirati sledeči regulacijski zakon

$$\boldsymbol{u}(t) = \boldsymbol{K}(\boldsymbol{z}_{ref}(t) - \boldsymbol{z}(t)) + \boldsymbol{u}_{ref}(t) \tag{3.26}$$

Shematski prikaz celotnega sistema vodenja je podan na sliki 3.18.

Parametre regulatorja (matriko ojačenj $\boldsymbol{K}$) lahko določimo z metodo premikanja polov s pomočjo Ackermannove formule, ki jo najdete v klasičnih knjigah s področja teorije regulacij [8]. Zaradi posebne oblike matrik $\boldsymbol{A}$ in $\boldsymbol{B}$ v (3.20), kjer u_1 vpliva samo na stanji z_1 in z_2 ter u_2 vpliva samo na stanji z_3 in z_4, ima matrika ojačenj regulatorja posebno obliko

$$\boldsymbol{K} = \begin{bmatrix} k_1 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & k_4 \end{bmatrix}$$

Regulacijski zakon (3.26) je torej mogoče popolnoma razčleniti

$$\begin{aligned} u_1(t) = \ddot{x}(t) &= k_1(x_{ref}(t) - x(t)) + k_2(\dot{x}_{ref}(t) - \dot{x}(t)) + \ddot{x}_{ref}(t) \\ u_2(t) = \ddot{y}(t) &= k_3(y_{ref}(t) - y(t)) + k_4(\dot{y}_{ref}(t) - \dot{y}(t)) + \ddot{y}_{ref}(t) \end{aligned} \tag{3.27}$$

Predlagan pristop zahteva, da so vsa stanja znana. Medtem ko običajno izmerimo x in y, njihovih odvodov ne. Odvode sicer lahko ocenimo z numeričnim odvajanjem, vendar to povečuje šum in se ga zato v praksi izogibamo. Na voljo sta dve rešitvi:

- Neizmerjena stanja lahko ocenijo *opazovalniki stanj*.
- Če izmerimo orientacijo robota φ, lahko izračunamo odvode kot $\dot{x} = v\cos\varphi$, $\dot{y} = v\sin\varphi$.

Praktična uporaba tega pristopa je prikazana v primeru 3.8.

Primer 3.8

Vodite vozilo z diferencialnim pogonom, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Računski korak je $T_s = 0{,}033\,\mathrm{s}$. Začetna lega je $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. V Matlab kodi izvedite algoritem, predstavljen v tem razdelku, in grafično prikažite rezultate.

Rešitev

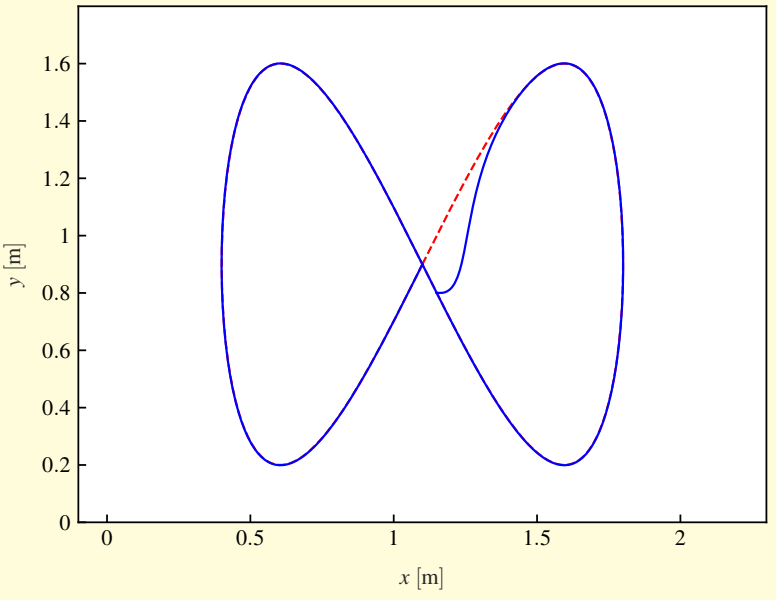
Koda je predstavljena v programu 3.7, rezultati primera 3.8 pa so prikazani na slikah 3.19 in 3.20. V tem pristopu se ne pojavijo težave periodične orientacije (ni potrebno preslikati kotov na interval $(-\pi, \pi]$). To izhaja iz dejstva, da se orientacija vedno pojavi znotraj trigonometričnih funkcij, ki so same po sebi periodične.

Program 3.7

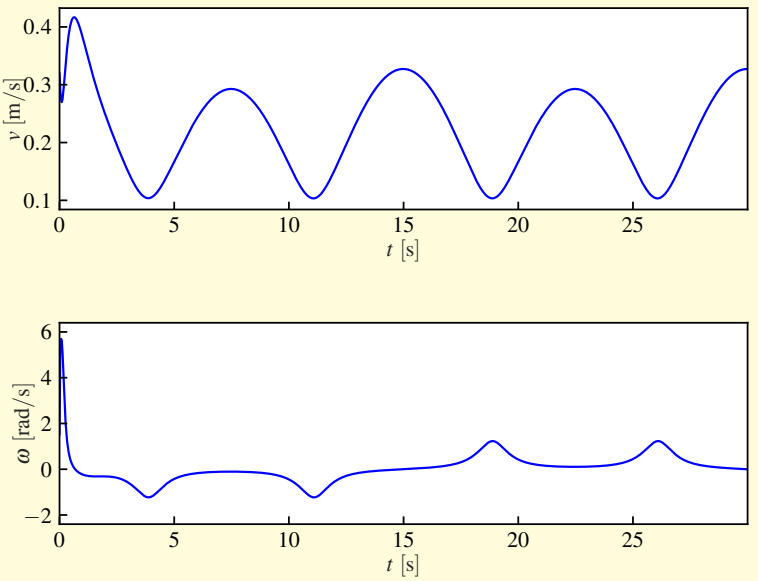
./src/ctr/example_tracking_feedback_lin.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)];
uRef = [ddxRef; ddyRef];

q = [xRef(1)+.05; yRef(1)-0.1; 0]; % Začetna lega
z1 = [q(1); dxRef(1)]; % Začetno stanje [x, x']
z2 = [q(2); dyRef(1)]; % Začetno stanje [y, y']
v = sqrt(z1(2)^2+z2(2)^2); % Začetno stanje hitrosti

% Matrike linearizirane sistema
A = [0, 1; 0, 0]; B = [0; 1]; C = [1, 0];
% Regulator stanj
desPoles = [-2-1i; -2+1i]; % Želeni zaprtozančni poli
K = acker(A, B, desPoles); % Ojačenje regularja po metodi premikanja polov
```

```
22
23 for k = 1:length(t)
24     % Referenčna stanja
25     zRef1 = [xRef(k); dxRef(k)];
26     zRef2 = [yRef(k); dyRef(k)];
27
28     % Pogrešek in regulator
29     ez1 = zRef1 - z1;
30     ez2 = zRef2 - z2;
31     uu = [ddxRef(k); ddyRef(k)] + [K*ez1; K*ez2];
32
33     % Izračun regulirnih signalov
34     F = [cos(q(3)), -v*sin(q(3)); ...
35          sin(q(3)),  v*cos(q(3))];
36     vv = F\uu; % Translatorni pospešek in kotna hitost
37     v = v + Ts*vv(1); % Integracija translatornega pospeška
38     u = [v; vv(2)]; % Regulirna signala
39
40     % Simulacija gibanja robota
41     dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
42     noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
43     q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
44     q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
45
46     % Izračun stanj na podlagi znane (izmerjene) orientacije in hitrosti
47     z1 = [q(1); u(1)*cos(q(3))];
48     z2 = [q(2); u(1)*sin(q(3))];
49 end
```



Slika 3.19: Vodenje po referenčni trajektoriji diferencialnega pogona na podlagi povratnozančne linearizacije iz primera 3.8 (referenca je označena s črtkano krivuljo)



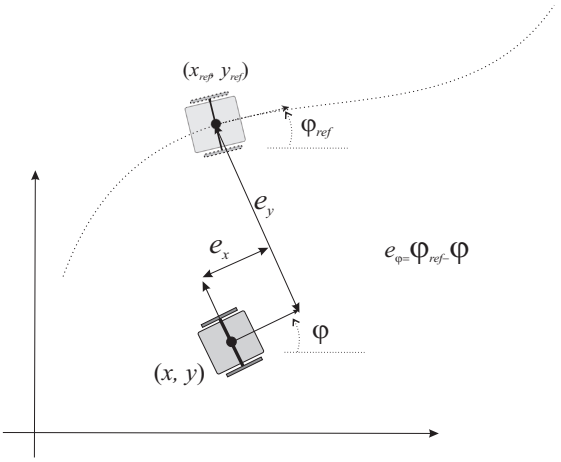
Slika 3.20: Regulirni signali vodenja po referenčni trajektoriji diferencialnega pogona na podlagi povratnozančne linearizacije iz primera 3.8

3.3.4 Izpeljava kinematičnega modela pogreška vodenja pri sledenju referenčne trajektorije

Da rešimo problem vodenja, običajno izvedemo ustrezno transformacijo koordinat robota. Pozicijski pogrešek je ponavadi podan v lokalnem koordinatnem sistemu (robota), poravnan s pogonskim mehanizmom, in izražen kot odstopanje virtualnega referenčnega robota od dejanskega robota, kar prikazuje slika 3.21. Na sliki 3.21 so predstavljeni tudi vsi dobljeni pogreški: e_x podaja pogrešek v smeri vožnje, e_y podaja pogrešek v pravokotni smeri in e_φ podaja pogrešek orientacije. Opisani pristop je bil prvič uporabljen v [9].

Pogrešek lege $\boldsymbol{e}(t) = [e_x(t), e_y(t), e_\varphi(t)]^T$ je določen z dejansko lego $\boldsymbol{q}(t) = [x(t), y(t), \varphi(t)]^T$ resničnega robota in referenčno lego $\boldsymbol{q}_{ref}(t) = [x_{ref}(t), y_{ref}(t), \varphi_{ref}(t)]^T$ virtualnega referenčnega robota

$$\begin{bmatrix} e_x(t) \\ e_y(t) \\ e_\varphi(t) \end{bmatrix} = \begin{bmatrix} \cos(\varphi(t)) & \sin(\varphi(t)) & 0 \\ -\sin(\varphi(t)) & \cos(\varphi(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} (\boldsymbol{q}_{ref}(t) - \boldsymbol{q}(t)) \tag{3.28}$$



Slika 3.21: Prikaz pozicijskega pogreška v lokalnih koordinatah

Ob predpostavki, da imata dejanski in referenčni robot enak kinematični model, ki ga podaja (2.2), in ob upoštevanju transformacije (3.28), lahko model pogreška lege zapišemo na naslednji način

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\varphi \end{bmatrix} = \begin{bmatrix} \cos e_\varphi & 0 \\ \sin e_\varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} -1 & e_y \\ 0 & -e_x \\ 0 & -1 \end{bmatrix} \boldsymbol{u} \tag{3.29}$$

kjer sta v_{ref} in ω_{ref} linearna in kotna referenčna hitrost, podani z (3.17) in (3.18). Regulator določa vhod $\boldsymbol{u} = [v, \omega]^T$. Zelo pogosto [10] je regulirna veličina $\boldsymbol{u}$ razčlenjena kot

$$\boldsymbol{u} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_{ref} \cos e_\varphi + v_{fb} \\ \omega_{ref} + \omega_{fb} \end{bmatrix} \tag{3.30}$$

kjer sta v_{fb} in ω_{fb} povratnozančna (regulacijska) signala, ki bosta določena kasneje, $v_{ref} \cos e_\varphi$ in ω_{ref} pa sta signala predkrmiljenja, čeprav je tehnično gledano $v_{ref} \cos e_\varphi$ moduliran s pogreškom orientacije, ki izvira iz izhoda. Po drugi strani pa $v_{ref} \cos e_\varphi$ postane "pravo" predkrmiljenje, ko je pogrešek orientacije enak 0. Če vstavimo regulirno veličino (3.30) v (3.29), dobimo model sledilnega pogreška

$$\begin{aligned} \dot{e}_x &= \omega_{ref} e_y - v_{fb} + e_y \omega_{fb} \\ \dot{e}_y &= -\omega_{ref} e_x + v_{ref} \sin e_\varphi - e_x \omega_{fb} \\ \dot{e}_\varphi &= -\omega_{fb} \end{aligned} \tag{3.31}$$

Cilj vodenja je izničiti pogreške modela sledilnega pogreška (3.31) z ustreznima regulirnima veličinama v_{fb} in ω_{fb}. S tem se bomo ukvarjali v nadaljevanju.

3.3.5 Linearni regulator

Model pogreška (3.31) je nelinearen. V tem razdelku ga bomo linearizirali, kar omogoča uporabo linearne regulacije. Linearizacija mora potekati okoli

ravnotežne točke, zato izberemo točko ničelnega pogreška ($e_x = e_y = 0, e_\varphi = 0$), ki je logična ravnotežna točka modela (3.31), če sta tudi obe hitrosti povratne zanke enaki 0 ($v_{fb} = 0$, $\omega_{fb} = 0$). Linearizacijo modela (3.31) okoli točke ničelnega pogreška zapišemo v obliki

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\varphi \end{bmatrix} = \begin{bmatrix} 0 & \omega_{ref} & 0 \\ -\omega_{ref} & 0 & v_{ref} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\varphi \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_{fb} \\ \omega_{fb} \end{bmatrix} \tag{3.32}$$

Ta linearen sistem je časovno spremenljiv, ker sta $v_{ref}(t)$ in $\omega_{ref}(t)$ časovno odvisni.

Sistem (3.32) je predstavitev dinamičnega sistemskega pogreška v prostoru stanj, kjer so vsa stanja (v tem primeru pogreški) dostopna. Povratna zanka iz stanj je torej mogoča (rezultira v uspešno vodenje), če je sistem vodljiv. Ob predpostavki, da sta v_{ref} in ω_{ref} konstantni (referenčna pot je sestavljena iz daljic in krožnih lokov), lahko enostavno dokažemo, da je matrika vodljivosti (3.22) polnega ranga in lahko vse pogreške izničimo z regulatorjem stanj. V primeru da v_{ref} in ω_{ref} nista konstantni, je sistem še vedno vodljiv, če je katerikoli od referenčnih signalov različen od 0. Tovrstna analiza pa je veliko bolj zapletena.

Zaradi posebne strukture sistema (3.32) se pogosto uporablja linearni regulator stanj s preprosto obliko matrike ojačenj

$$\begin{bmatrix} v_{fb} \\ \omega_{fb} \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & k_\varphi \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\varphi \end{bmatrix}$$

Vidimo, da se pogrešek v smeri vožnje popravi za v_{fb}, medtem ko se pogreški v orientaciji in bočnih smereh popravijo z ω_{fb}.

Ojačenja regulatorja (k_x, k_y, k_φ) je mogoče določiti s poskušanjem, z njihovo optimizacijo na modelu sistema, z metodo premikanja polov itd. V nadaljevanju so ojačenja regulatorja določena z metodo premikanja polov tako, da poli sistema ležijo na ustreznih lokacijah v kompleksni ravnini s. Sistem ima tri pole, torej je vsaj en pol realen, druga dva pa lahko izberemo, da sta konjugirano kompleksna. Predpostavimo, da so želene lege zaprtozančnih polov $s_1 = -2\zeta\omega_n$ in $s_{2,3} = -\zeta\omega_n \pm \omega_n\sqrt{1-\zeta^2}$. Lastna frekvenca $\omega_n > 0$ in koeficient dušenja $0 < \zeta < 1$ sta parametra, ki ju lahko nastavimo tako, da dosežemo zadovoljivo dušenje in hiter prehodni pojav. Če karakteristični polinom zaprtozančnega sistema

$$\left| s\boldsymbol{I}_{3\times 3} - \begin{bmatrix} 0 & \omega_{ref} & 0 \\ -\omega_{ref} & 0 & v_{ref} \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & k_\varphi \end{bmatrix} \right|$$

primerjamo z želenim

$$(s + 2\zeta\omega_n)(s^2 + 2\zeta\omega_n s + \omega_n^2)$$

lahko dobimo rešitev za ojačenja regulatorja [6]

$$\begin{aligned} k_x &= k_\varphi = 2\zeta\omega_n \\ k_y(t) &= \frac{\omega_n^2 - \omega_{ref}^2(t)}{v_{ref}(t)} \end{aligned} \tag{3.33}$$

Upoštevamo, da mora biti ω_n večja od največje vrednosti $|\omega_{ref}(t)|$. Ojačenja regulatorja (3.33) se praktično ne uporabljajo, ker $k_y(t)$ postane izjemno velik, ko je referenčna hitrost $v_{ref}(t)$ majhna. To težavo odpravimo s časovno spremenljivo lastno frekvenco ω_n. Ker je smiselno prilagoditi čas umiritve prehodnega pojava glede na referenčne hitrosti, se zdi primerna izbira: $\omega_n(t) = \sqrt{\omega_{ref}^2(t) + gv_{ref}^2(t)}$, $g > 0$. Po ponovitvi podobnega postopka (kot zgoraj), dobimo naslednja ojačenja regulatorja

$$\begin{aligned} k_x(t) &= k_\varphi(t) = 2\zeta\sqrt{\omega_{ref}^2(t) + gv_{ref}^2(t)} \\ k_y(t) &= gv_{ref}(t) \end{aligned}$$

V okviru algoritmov vodenja, predstavljenih v tem poglavju, moramo izpostaviti dve pripombi:

- Regulacijski zakoni so zasnovani na podlagi lineariziranih modelov. Linearizirani model je veljaven le v bližini delovne točke (v tem primeru je to točka ničelnega pogreška) in pri velikih regulacijskih pogreških njegova učinkovitost morda ne bo takšna, kot je bila pričakovana.

- Če imamo opravka z linearnim, a časovno spremenljivim sistemom, nekateri rezultati linearnih časovno nespremenljivih sistemov niso več veljavni. Tu je potrebno omeniti, da je sistem morda nestabilen, četudi vsi poli ležijo na (fiksnih) lokacijah na levi strani kompleksne ravnine s.

Kljub omenjenim možnim težavam se linearni regulacijski zakoni v praksi pogosto uporabljajo zaradi njihove enostavnosti, razmeroma enostavne prilagoditve ter sprejemljive zmogljivosti in robustnosti. Simulacija uporabe je podana v primeru 3.9.

Primer 3.9

Vodite vozilo z diferencialnim pogonom, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Računski korak je $T_s = 0{,}033\,\mathrm{s}$, začetna lega pa $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. Zapišite predstavljen algoritem z ojačenji regulatorja, ki jih podaja (3.33), in grafično prikažite rezultate.

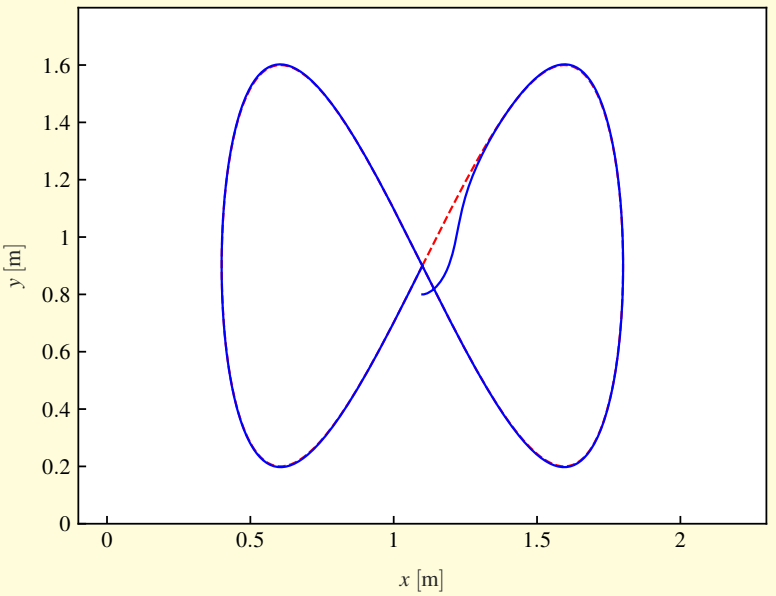
Rešitev

Matlab koda je podana v programu 3.8. Rezultati simulacije so prikazani na slikah 3.22 in 3.23, kjer je prikazano dobro sledenje.

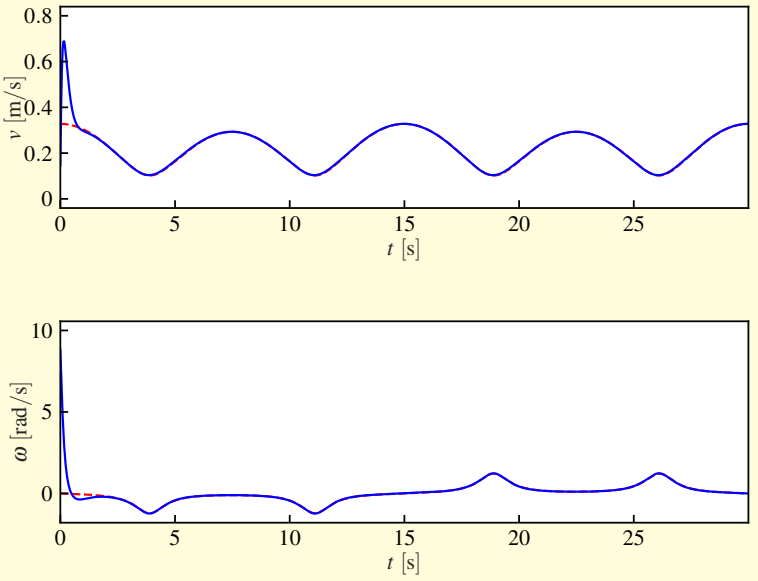
Program 3.8

./src/ctr/example_tracking_linear_control.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Reference trajectory
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

for k = 1:length(t)
    e = [cos(q(3)), sin(q(3)), 0; ...
        -sin(q(3)), cos(q(3)), 0; ...
         0,         0,         1]*(qRef(:,k) - q); % Vektor pogreška
    e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]

    % Trenutni referenčni vhodi
    vRef = uRef(1,k);
    wRef = uRef(2,k);

    % Regulator
    eX = e(1); eY=e(2); ePhi=e(3);
    zeta = 0.9; % Parameter za nastavljanje
    g = 85;     % Parameter za nastavljanje
    Kx = 2*zeta*sqrt(wRef^2+g*vRef^2);
    Kphi = Kx;
    Ky = g*vRef;
    % Ojačenja so lahko tudi konstantna, npr.: Kx = Kphi = 3; Ky = 30;

    % Regulator: krmiljenje in regulacija
    v = vRef*cos(e(3))+ Kx*e(1);
    w = wRef + Ky*e(2) + Kphi*e(3);

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.22: Vodenje diferencialnega pogona po referenčni trajektoriji iz primera 3.9 (referenca je označena s črtkano krivuljo)



Slika 3.23: Regulirni signali vodenja diferencialnega pogona po referenčni trajektoriji iz primera 3.9

3.3.6 Načrtovanje vodenja na osnovi funkcij Ljapunova

Kot smo že omenili, je model pogreška (3.31) sam po sebi nelinearen. Nelinearne sisteme je najbolje voditi z nelinearnim regulatorjem, ki med načrtovanjem vodenja upošteva vse lastnosti sistema. Teorija, ki temelji na funkcijah Ljapunova, se pogosto uporablja za reševanje težav pri stabilizaciji nelinearnega sistema. V našem primeru bomo (asimptotično) stabilnost modela pogreška (3.31) analizirali glede na različne regulacijske zakone.

Stabilnost Ljapunova

Na kratko je predstavljena druga metoda Ljapunova, ki zagotavlja zadostne pogoje za (asimptotično) stabilnost ravnotežnih točk nelinearnega dinamičnega sistema $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}$. Najprej predpostavimo, da ravnotežje leži v $\boldsymbol{x} = 0$. Pristop temelji na pozitivno definitnih skalarnih funkcijah $V(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$, za katere velja $V(\boldsymbol{x}) = 0$, če je $\boldsymbol{x} = 0$, in $V(\boldsymbol{x}) > 0$, ko je $\boldsymbol{x} \neq 0$. Stabilnost ravnotežne točke preverimo z odvodom funkcije V. Pomembno je, da dobimo odvod kot rešitev diferencialne enačbe sistema

$$\dot{V} = \frac{\partial V}{\partial \boldsymbol{x}} \dot{\boldsymbol{x}} = \frac{\partial V}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$$

Če velja $\dot{V} \leq 0$ ($\dot{V}$ je negativno semidefinitna funkcija), je ravnotežje (lokalno) stabilno. Če pa velja $\dot{V} < 0$, razen pri $\boldsymbol{x} = 0$ ($\dot{V}$ je negativno definitna funkcija), je ravnotežje (lokalno) asimptotično stabilno. Ko je $\lim_{|\boldsymbol{x}| \to \infty} V(\boldsymbol{x}) = \infty$, so rezultati globalni. Zato pristop temelji na iskanju funkcij z navedenimi lastnostmi, ki jih imenujemo funkcije Ljapunova. Za kandidata običajno izberemo kvadratno funkcijo Ljapunova in če je možno pokazati, da je njen odvod negativen ali vsaj nič, je sistem stabilen.

Klasična razlaga funkcij Ljapunova temelji na energiji sistema. Če se energija disipativnega sistema uporablja kot funkcija Ljapunova, se njegova energija ne more povečati (odvod funkcije ni pozitiven). Posledično ostanejo vsi signali omejeni in lahko potrdimo stabilnost sistema. Vendar je potrebno poudariti, da funkcija Ljapunova morda ni povezana z energijo sistema. Predvsem pa je v okviru nelinearnosti "stabilnost sistema" napačen termin. Namesto tega je potrebno analizirati stabilnost ravnotežnih točk ali bolj splošno stabilnost invariantnih množic. Možno je najti sisteme, v katerih obstajajo tako stabilne kot nestabilne ravnotežne točke.

Načrtovanje vodenja v okviru stabilnosti Ljapunova

V nadaljevanju bomo pokazali, kako lahko teorem stabilnosti Ljapunova uporabimo za namen načrtovanja vodenja. Naš nelinearni sistem (3.31) ima tri stanja z ravnotežno točko pri $\boldsymbol{e} = 0$. Želimo oblikovati vodenje, ki bo to točko stabiliziralo; če je možno, naj bo točka asimptotično stabilna, kar pomeni, da bi se vse trajektorije sčasoma približale referenčni in tam ostale za vedno. Najočitnejši kandidat za funkcijo Ljapunova je vsota treh kvadratov pogreškov

$$V(\boldsymbol{e}) = \frac{k_y}{2}(e_x^2 + e_y^2) + \frac{1}{2}e_\varphi^2$$

to si lahko razlagamo kot uravnoteženo vsoto kvadratov pogreškov razdalje in orientacije. Zaradi različnih enot moramo dodati pozitivno konstanto k_y, vendar se bo pozneje pokazalo, da ta konstanta igra pomembno vlogo pri zasnovi regulacijskega zakona. Časovni odvod funkcije V je

$$\dot{V} = k_y e_x \dot{e}_x + k_y e_y \dot{e}_y + e_\varphi \dot{e}_\varphi$$

vendar je potrebno ta odvod ovrednotiti glede na rešitve modela (3.31), kar pomeni, da moramo vpeljati odvode pogreška iz (3.31)

$$\begin{aligned} \dot{V}(\boldsymbol{e}) &= k_y e_x \left(\omega_{ref} e_y - v_{fb} + e_y \omega_{fb}\right) + \\ &\quad + k_y e_y \left(-\omega_{ref} e_x + v_{ref} \sin e_\varphi - e_x \omega_{fb}\right) + e_\varphi \left(-\omega_{fb}\right) \\ &= -k_y e_x v_{fb} + k_y v_{ref} e_y \sin e_\varphi - e_\varphi \omega_{fb} \end{aligned} \tag{3.34}$$

Osnovna ideja vodenja, ki temelji na metodi Ljapunova, je ustrezna izbira regulacijskega zakona, ki zagotovi, da je odvod funkcije Ljapunova negativen. V tem primeru je precej očitno, kako lahko izvedemo regulacijski algoritem. Člen $-k_y e_x v_{fb}$ v (3.34) bo negativen, če bo linearna hitrost v_{fb} proporcionalna pogrešku e_x, saj je kvadrat pogreška e_x^2 pozitiven. Podobno bo člen $-e_\varphi \omega_{fb}$ v (3.34) vedno negativen, če bo kotna hitrost ω_{fb} proporcionalna pogrešku e_φ. S primerno modifikacijo kotne hitroste ω_{fb} lahko dosežemo še, da izničimo člen $k_y v_{ref} e_y \sin e_\varphi$ v (3.34). S tem zagotovimo, da je odvod funkcije Ljapunova (3.34) negativen. Regulacijski zakon, ki to izpolnjuje, je

$$\begin{aligned} v_{fb} &= k_x e_x \\ \omega_{fb} &= k_y v_{ref} \frac{\sin e_\varphi}{e_\varphi} e_y + k_\varphi e_\varphi \end{aligned} \tag{3.35}$$

Ta regulacijski zakon je dobro znan in uveljavljen [6, 11]. Uvedemo predlagano vodenje (3.35) in $\dot{V}$ postane

$$\dot{V} = -k_x k_y e_x^2 - k_\varphi e_\varphi^2 \tag{3.36}$$

Ojačenja vodenja so pozitivna, kasneje pa se bo pokazalo, da sta lahko k_x in k_φ poljubni enakomerno zvezni pozitivni funkciji, medtem ko mora biti k_y pozitivna konstanta. Odvod funkcije Ljapunova očitno ni pozitiven, ker pa je ocenjen na nič pri $e_x = 0$, $e_\varphi = 0$, ne glede na e_y, je odvod negativen in semidefiniten; tudi

ravnotežje je stabilno. To pomeni, da bo pogrešek ostal omejen, vendar nismo dokazali njegove konvergence proti 0.

Analiza konvergence pogreška je bistveno težja, zato moramo uvesti nekaj dodatnih matematičnih orodij. Pomembno vlogo bodo igrale norme signalov. Norma $\mathcal{L}_p$ funkcije $x(t)$ je definirana kot

$$\|x\|_p = \left(\int_0^\infty |x(\tau)|^p \, \mathrm{d}\tau \right)^{1/p}$$

kjer je $|\cdot|$ (skalarna) dolžina vektorja. Če zgornji integral obstaja (je končen), funkcija $x(t)$ pripada $\mathcal{L}_p$. Omejitev p na neskončnost zagotavlja zelo pomemben razred funkcij $\mathcal{L}_\infty$, t. i. *omejene funkcije*.

Za dokazovanje stabilnosti regulacijskih zakonov bomo uporabili dve zelo znani lemi. Prva je Barbălatova lema, druga pa je njena izpeljava. Obe lemi sta vzeti iz [12].

Lema 3.1 (Barbălatova lema)**.** *Če* $\lim_{t\to\infty} \int_0^t f(\tau)\,\mathrm{d}\tau$ *obstaja in je končna ter je* $f(t)$ *enakomerno zvezna funkcija, potem velja* $\lim_{t\to\infty} f(t) = 0$*.*

Lema 3.2. *Če velja* $f, \dot{f} \in \mathcal{L}_\infty$ *in* $f \in \mathcal{L}_p$ *za določene* $p \in [1, \infty)$*, potem* $f(t) \to 0$ *ko* $t \to \infty$*.*

Zdaj smo pripravljeni obravnavati problem konvergence pogreška v (3.31). Zaradi (3.36) je $\dot{V} \leq 0$, zato funkcija Ljapunova ne narašča in ima limito $\lim_{t\to\infty} V(t)$. Posledično so stanja modela (3.31) omejena

$$e_x,\ e_y,\ e_\varphi \in \mathcal{L}_\infty$$

Poleg tega iz (3.35) izhaja, da so regulirni signali omejeni, iz (3.31) pa da so omejeni odvodi pogreškov

$$v_{fb},\ \omega_{fb},\ \dot{e}_x,\ \dot{e}_y,\ \dot{e}_\varphi \in \mathcal{L}_\infty$$

kjer smo upoštevali, da so v_{ref}, ω_{ref}, k_x in k_φ omejeni. Slednje velja v primeru ploskih referenčnih trajektorij $(x_{ref}, y_{ref}, \varphi_{ref})$.

Da dokažemo asimptotično stabilnost modela (3.31), najprej izračunamo integral $\dot{V}$ iz (3.36)

$$\int_0^\infty \dot{V} dt = V(\infty) - V(0) = -\int_0^\infty k_x k_y e_x^2 \,\mathrm{d}t - \int_0^\infty k_\varphi e_\varphi^2 \,\mathrm{d}t$$

Ker je V pozitivno definitna funkcija, velja naslednja neenakost

$$V(0) \geq \int_0^\infty k_x k_y e_x^2 \,\mathrm{d}t + \int_0^\infty k_\varphi e_\varphi^2 \,\mathrm{d}t \geq \underline{k}_x k_y \int_0^\infty e_x^2 \,\mathrm{d}t + \underline{k}_\varphi \int_0^\infty e_\varphi^2 \,\mathrm{d}t$$

kjer sta uvedeni spodnji meji funkcij $k_x(t)$ in $k_\varphi(t)$

$$k_x(t) \geq \underline{k}_x > 0$$
$$k_\varphi(t) \geq \underline{k}_\varphi > 0$$

Iz (3.3.6) izhaja, da pogreška $e_x(t)$ in $e_\varphi(t)$ pripadata $\mathcal{L}_2$. Na podlagi leme 3.2 je možno enostavno pokazati, da pogreška $e_x(t)$ in $e_\varphi(t)$ konvergirata proti 0. Ker obstaja limita $\lim_{t\to\infty} V(t)$, potem obstaja tudi $\lim_{t\to\infty} e_y(t)$.

Videli smo, da je razmeroma enostavno prikazati konvergenco pogreškov $e_x(t)$ in $e_\varphi(t)$ proti 0. Tudi pogoji za konvergenco so dokaj blagi – ojačenja regulatorja in referenčne trajektorije morajo biti omejeni. Konvergenco e_y proti 0 pa je težje dokazati, saj so zahteve veliko težje dosegljive, kot bo prikazano v nadaljevanju. Poleg tega, da so ojačenja regulatorja enakomerno zvezna, morajo biti referenčne hitrosti neprestano vzbujene, torej v_{ref} in ω_{ref} ne smeta limitirati proti 0. Zato bomo obravnavali dva primera. V prvem predpostavimo $v_{ref} \nrightarrow 0$, v drugem pa $\omega_{ref} \nrightarrow 0$.

Predpostavimo, da je $\lim_{t\to\infty} v_{ref}(t) \neq 0$. Uporaba leme 3.1 na $\dot{e}_\varphi(t)$ iz (3.31) zagotavlja, da $\lim_{t\to\infty} \dot{e}_\varphi(t) = 0$, saj limita $\lim_{t\to\infty} e_\varphi(t)$ obstaja in je končna, odvod $\dot{e}_\varphi(t)$ pa je enakomerno zvezen. Slednje velja zaradi (3.31), če je ω_{fb} enakomerno zvezna. Enakomerno zveznost funkcije $f(t)$ na $[0,\infty)$ preverimo tako, da pogledamo, ali velja $f, \dot{f} \in \mathcal{L}_\infty$. Prej smo dokazali, da sta e_y in e_φ enakomerno zvezna, medtem ko sta ojačenje regulatorja k_φ in referenčna hitrost v_{ref} enakomerno zvezna ob predpostavki iz (3.35), da je odvod $\dot{e}_\varphi(t)$ tudi enakomerno zvezen. Tako smo dokazali, da $\lim_{t\to\infty} \dot{e}_y(t) = 0$ velja (kar je enako $\lim_{t\to\infty} \omega_{fb}(t) = 0$). Konvergenca e_y proti 0 izhaja iz (3.35)

$$e_\varphi \to 0, k_\varphi \in \mathcal{L}_\infty, \omega_{fb} \to 0 \Rightarrow k_y v_{ref} \tfrac{\sin e_\varphi}{e_\varphi} e_y \to 0$$
$$k_y v_{ref} \tfrac{\sin e_\varphi}{e_\varphi} e_y \to 0, \tfrac{\sin e_\varphi}{e_\varphi} \to 1, k_y > 0, v_{ref} \to 0 \Rightarrow e_y \to 0$$

Zdaj predpostavimo $\lim_{t\to\infty} \omega_{ref}(t) \neq 0$. Spet je potrebno zagotoviti, da velja $\lim_{t\to\infty} \omega_{fb} = 0$. Kot smo že pokazali, to drži, če sta v_{ref} in k_φ enakomerno zvezna. Nato se Barbălatova lema (lema 3.1) uporabi na $\dot{e}_x$ v (3.31). Za e_x, e_y in ω_{fb} smo tudi že pokazali, da so enakomerno zvezni. Ob predpostavki, da je k_x enakomerno zvezen, sta tudi v_{fb} in ω_{ref} enakomerno zvezni. To dokazuje izraz $\lim_{t\to\infty} \dot{e}_x(t) = 0$. Podobno lahko sklepamo, da zadnja dva izraza v enačbi (3.31) za $\dot{e}_x$ limitirata proti 0, ko gre t proti neskončnosti. Posledično gre tudi produkt $\omega_{ref} e_y$ proti 0. Ker je ω_{ref} neprestano vzbujena in ne gre proti 0, mora iti e_y proti 0.

Še enkrat je potrebno poudariti, da je za konvergenco e_x in e_φ potrebna le omejenost v_{ref} ali ω_{ref}. Precej težja naloga je voditi e_y na 0. To dosežemo z neprestanim vzbujanjem v_{ref} ali ω_{ref}. Vsi rezultati so veljavni globalno, kar pomeni, da je konvergenca zagotovljena ne glede na začetno lego.

Primer 3.10

Vodite vozilo z diferencialnim pogonom, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Računski korak je $T_s = 0{,}033\,\mathrm{s}$, začetna lega pa je $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. V Matlab kodi izvedite

predstavljeni algoritem vodenja, preizkusite različna ojačenja in grafično prikažite rezultate.

Rešitev

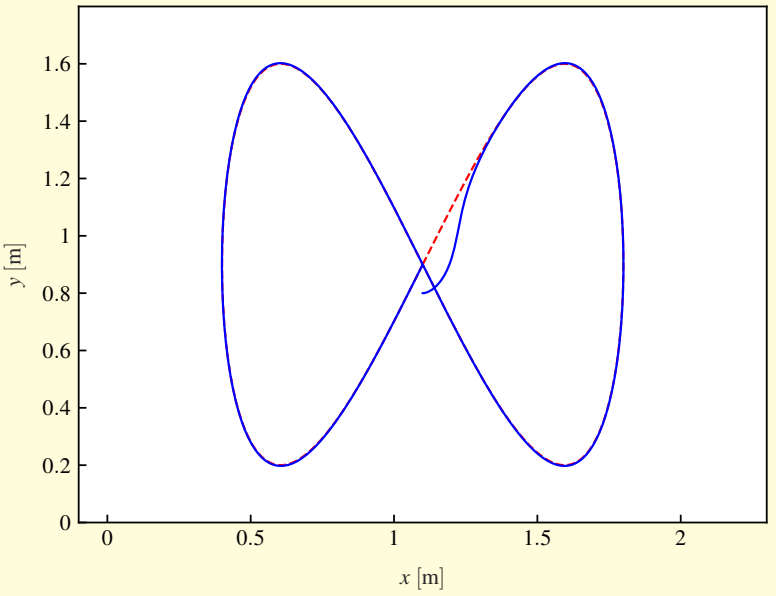
Koda je podana v programu 3.9. Rezultati simulacije primera 3.10 so prikazani na slikah 3.24 in 3.25, kjer je ponazorjeno dobro sledenje. Upoštevamo, da se za $k_x(t)$ in $k_\varphi(t)$ lahko uporabi poljubna pozitivna funkcija. V tem primeru smo izbrali takšne funkcije, da dobimo enak linearni model sistema kot v primeru linearnega regulatorja (primer 3.9). Regulacijski zakoni niso enaki razen v mejnih primerih ($e_\varphi \to 0$). Tako je oblika prehoda podobna referenčni trajektoriji, ne glede na referenčne hitrosti.

Program 3.9

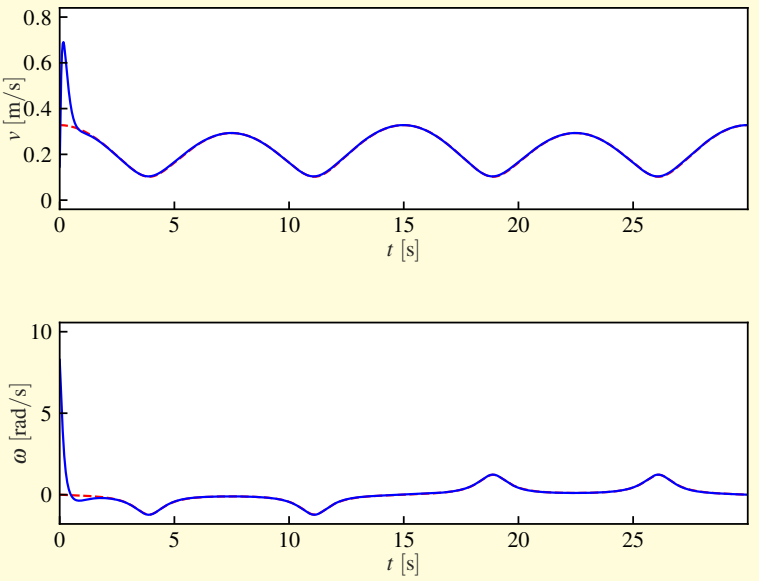
./src/ctr/example_tracking_nonlinear_control.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

for k = 1:length(t)
    e = [cos(q(3)), sin(q(3)), 0; ...
        -sin(q(3)), cos(q(3)), 0; ...
         0,         0,         1]*(qRef(:,k) - q); % Vektor pogreška
    e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]

    % Trenutni referenčni vhodi
    vRef = uRef(1,k);
    wRef = uRef(2,k);

    % Regulator
    zeta = 0.9; % Parameter za nastavljanje
    g = 85;     % Parameter za nastavljanje
    Kx = 2*zeta*sqrt(wRef^2 + g*vRef^2);
    Kphi = Kx;
    Ky = g;
    % Ojačenji Kx in Kphi sta lahko tudi konstantni.
    % Ta oblika omogoča, da je dušenje v prehodnem pojavu
    % neodvisno od referenčnih hitrosti.

    % Regulator: krmiljenje in regulacija
    v = vRef*cos(e(3)) + Kx*e(1);
    w = wRef + Ky*vRef*sinc(e(3)/pi)*e(2) + Kphi*e(3);

    % Simulacija gibanja robota
```

```
40      dq = [v*cos(q(3)); v*sin(q(3)); w];
41      noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
42      q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
43      q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
44  end
```



Slika 3.24: Nelinearno vodenje po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.9 (referenca je označena s črtkano krivuljo)



Slika 3.25: Regulirni signalni nelinearnega vodenja po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.9

Razvoj periodičnega regulacijskega zakona

Težava sledenja je očitno periodičnost glede na orientacijo. To je mogoče opaziti iz kinematičnega modela z uporabo poljubne regulirne veličine in poljubnega začetnega pogoja, ki podajata določeno trajektorijo robota. Če isto regulirno veličino uporabimo na dejanskem robotu in se začetni pogoj razlikuje od prejšnjega samo za večkratnik 2π, dobimo enak odziv za $x(t)$ in $y(t)$; tudi $\varphi(t)$ se od prejšnje rešitve razlikuje za isti večkratnik 2π. Periodična narava se mora odražati tudi v uporabljenemu regulacijskemu zakonu. To pomeni, da moramo poiskati tak regulacijski zakon, ki je periodičen glede na pogrešek orientacije e_φ (perioda je 2π) in zagotavlja, da je konvergenca pogreška lege $\boldsymbol{e}$ enaka $[0\ 0\ 2k\pi]^T$ $(k \in \mathbb{Z})$. Tako zmanjšamo vse običajne probleme s preslikavo orientacije na $(-\pi, \pi]$. Ti problemi lahko v določenih aplikacijah postanejo kritični okoli $\pm 180°$, npr. pri uporabi opazovalnika za oceno lege robota iz zakasnelih meritev. Velja opomniti, da so določeni regulacijski zakoni periodični v smislu predhodne diskusije, npr. regulacijski zakon povratnozančne linearizacije, ki ga podajata (3.19) in (3.27).

Očitno bi morale biti funkcije, uporabljene v razdelku o analizi konvergence, periodične tudi glede na e_φ. To pomeni, da imajo te funkcije več lokalnih minimumov in zato ne izpolnjujejo pogojev za klasične funkcije Ljapunova. Čeprav je analiza

stabilnosti podobna direktni metodi Ljapunova (druga metoda Ljapunova), s to teorijo stabilnosti konvergenca ni dokazana, ker pri našem pristopu ni potrebno, da $\boldsymbol{e}$ konvergira proti nič. Kljub temu bomo v tem poglavju funkcije, uporabljene za analizo konvergence, še vedno imenovali "funkcije Ljapunova".

Naš cilj je spraviti pozicijski pogrešek na nič, medtem ko pogrešek orientacije konvergira proti poljubnemu večkratniku 2π. Da to dosežemo, bomo uporabili funkcijo Ljapunova, ki je periodična glede na e_φ (z osnovno periodo 2π). Najprej bo koncept prikazan na eni funkciji Ljapunova, kasneje pa ga bomo posplošili. Prvi kandidat za funkcijo Ljapunova je izbran kot

$$V = \tfrac{k_y}{2}\left(e_x^2 + e_y^2\right) + \tfrac{1}{2}\left(\frac{\tan\frac{e_\varphi}{2}}{\frac{1}{2}}\right)^2$$

kjer je k_y pozitivna konstanta. Njen odvod, upoštevajoč enačbe (3.31), pa je

$$\begin{aligned}\dot{V} &= k_y e_x\left(\omega_{ref} e_y - v_{fb} + e_y\omega_{fb}\right) + \\ &\quad + k_y e_y\left(-\omega_{ref} e_x + v_{ref}\sin e_\varphi - e_x\omega_{fb}\right) - 2\frac{\tan\frac{e_\varphi}{2}}{\cos^2\frac{e_\varphi}{2}}\omega_{fb} \\ &= -k_y e_x v_{fb} + k_y v_{ref} e_y \sin e_\varphi - 2\frac{\tan\frac{e_\varphi}{2}}{\cos^2\frac{e_\varphi}{2}}\omega_{fb}\end{aligned} \tag{3.37}$$

Če uporabimo regulacijski zakon

$$\begin{aligned}v_{fb} &= k_x e_x \\ \omega_{fb} &= k_y v_{ref} e_y \cos^4\frac{e_\varphi}{2} + k_\varphi \sin e_\varphi\end{aligned}$$

kjer sta k_x in k_φ pozitivno omejeni funkciji, je odvod $\dot{V}$ iz enačbe (3.37) enak

$$\dot{V} = -k_x k_y e_x^2 - k_\varphi\left(\frac{\tan\frac{e_\varphi}{2}}{\frac{1}{2}}\right)^2$$

Nato lahko sledimo istim korakom kot v analizi regulacijskega zakona (3.35) in ugotovimo, da e_x in $\tan\frac{e_\varphi}{2}$ konvergirata proti 0 (to pomeni, da velja $e_\varphi \to 2k\pi$, $k \in \mathbb{Z}$) v primeru omejenih ojačenj regulatorja in omejene trajektorije. Konvergenca e_y proti 0 se lahko zaključi tudi po dolgotrajni analizi, če so izpolnjeni isti pogoji kot v primeru regulacijskega zakona (3.35).

Primer 3.11

Vodite vozilo z diferencialnim pogonom, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Računski korak je $T_s = 0{,}033\,\mathrm{s}$, začetna lega pa $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. V Matlab kodi izvedite predstavljeni algoritem vodenja, preizkusite različna ojačenja regulatorja in grafično prikažite rezultate.

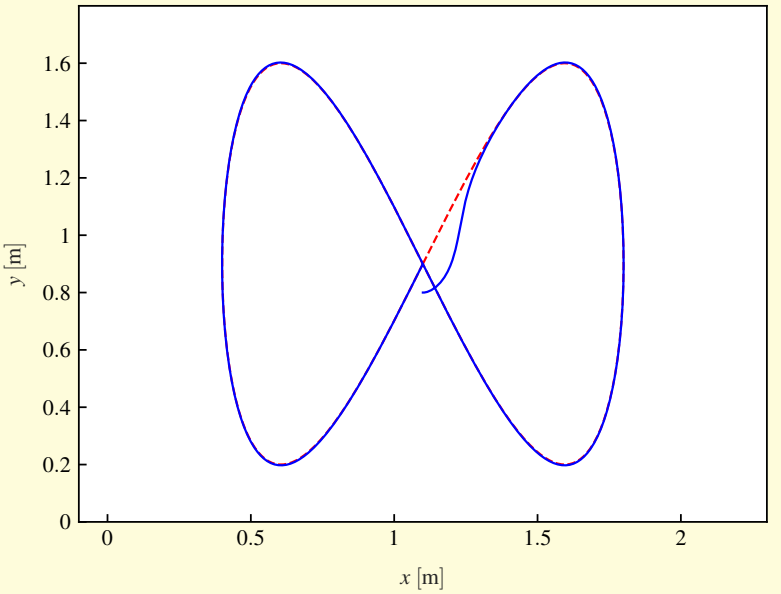
Rešitev

Koda je predstavljena v programu 3.10. Rezultati simulacije so prikazani na slikah 3.26 in 3.27, kjer je ponazorjeno dobro sledenje.

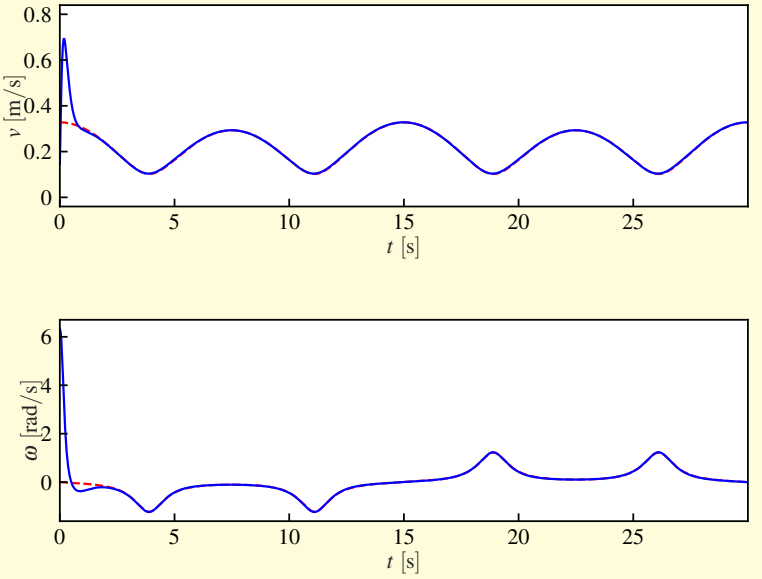
Program 3.10

./src/ctr/example_tracking_periodic_control.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

for k = 1:length(t)
    e = [cos(q(3)), sin(q(3)), 0; ...
        -sin(q(3)), cos(q(3)), 0; ...
         0,         0,         1]*(qRef(:,k) - q); % Vektor pogreška
    e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]

    % Trenutni referenčni vhodi
    vRef = uRef(1,k);
    wRef = uRef(2,k);

    % Regulator
    eX = e(1); eY = e(2); ePhi = e(3);
    zeta = 0.9; % Parameter za nastavljanje
    g = 85;     % Parameter za nastavljanje
    Kx = 2*zeta*sqrt(wRef^2+g*vRef^2);
    Kphi = Kx;
    Ky = g;
    % Ojačenji Kx in Kphi sta lahko tudi konstantni.
    % Ta oblika omogoča, da je dušenje v prehodnem pojavu
    % neodvisno od referenčnih hitrosti.

    % Regulator: krmiljenje in regulacija
    v = vRef*cos(e(3)) + Kx*eX;
    w = wRef + Ky*vRef*(cos(ePhi/2))^4*eY + Kphi*sin(ePhi);

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.26: Nelinearno vodenje po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.11 (referenca je označena s črtkano krivuljo)



Slika 3.27: Regulirni signali nelinearnega vodenja po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.11

Dobro znan regulacijski zakon Kanayama [10] lahko analiziramo v predlaganem okvirju. Rezultat izbire funkcije Ljapunova

$$V = \frac{k_y}{2}\left(e_x^2 + e_y^2\right) + \frac{1}{2}\left(\frac{\sin\frac{e_\varphi}{2}}{\frac{1}{2}}\right)^2$$

in uporabe regulacijskega zakona, predlaganega v [10] (v drugem izrazu za ω_{fb} je bil tudi tretji faktor $|v_{ref}|$, ki ga lahko vključimo v k_φ)

$$\begin{aligned} v_{fb} &= k_x e_x \\ \omega_{fb} &= k v_{ref} e_y + k_\varphi \sin e_\varphi \end{aligned} \tag{3.38}$$

je stabilen sistem pogreškov, kjer se lahko konvergenca vseh pogreškov prikaže pod enakimi pogoji kot prej. Upoštevamo, da poleg stabilnih ravnotežij pri $e_\varphi = 2k\pi$, $k \in \mathbb{Z}$ obstaja tudi nestabilno (odbijajoče se) ravnotežje pri $e_\varphi = (2k+1)\pi$, $k \in \mathbb{Z}$.

Okvir za zasnovo periodičnega zakona vodenja je predstavljen v [13]. Velja omeniti, da je precej preprosto razširiti predlagane tehnike na zasnovo vodenja za simetrična vozila, ki se lahko med normalnim delovanjem premikajo naprej in nazaj. V tem primeru morajo biti funkcije Ljapunova periodične s periodo π na e_φ.

Model pogreška sistema s štirimi stanji

Zdaj se bomo lotili istega problema kot v prejšnjem poglavju. Z vidika vodenja pogosto želimo slediti vsaki legi robota, ki se razlikuje od referenčne za večkratnik kota 360°. Model (3.31) ne olajša omenjenega problema, ker je običajno potrebno pogrešek orientacije izničiti z uporabo (3.31). V tem poglavju je predstavljen kinematični model sistema, kjer so vse lege, ki se v orientaciji razlikujejo za večkratnik kota 360°, predstavljene kot ena lega. To lahko dosežemo z razširitvijo vektorja stanj za en element. Spremenljivko $\varphi(t)$ iz prvotnega kinematičnega modela (2.2) zamenjata dve novi spremenljivki $s(t) = \sin(\varphi(t))$ in $c(t) = \cos(\varphi(t))$. Njuna odvoda sta

$$\begin{aligned} \dot{s}(t) &= \cos(\varphi(t))\dot{\varphi}(t) = c(t)\omega(t) \\ \dot{c}(t) &= -\sin(\varphi(t))\dot{\varphi}(t) = -s(t)\omega(t) \end{aligned}$$

Tako dobimo nov kinematični model

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{s} \\ \dot{c} \end{bmatrix} = \begin{bmatrix} c & 0 \\ s & 0 \\ 0 & c \\ 0 & -s \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Novi pogreški so opredeljeni kot

$$\begin{aligned} e_x &= c(x_{ref} - x) + s(y_{ref} - y) \\ e_y &= -s(x_{ref} - x) + c(y_{ref} - y) \\ e_s &= \sin(\varphi_{ref} - \varphi) = c\sin\varphi_{ref} - s\cos\varphi_{ref} \\ e_{cos} &= \cos(\varphi_{ref} - \varphi) = c\cos\varphi_{ref} + s\sin\varphi_{ref} \end{aligned} \tag{3.39}$$

Po odvajanju enačb (3.39) in nekaj manipulacij dobimo naslednji sistem

$$\begin{aligned} \dot{e}_x &= v_{ref}e_{cos} - v + e_y\omega \\ \dot{e}_y &= v_{ref}e_s - e_x\omega \\ \dot{e}_s &= \omega_{ref}e_{cos} - e_{cos}\omega \\ \dot{e}_{cos} &= -\omega_{ref}e_s + e_s\omega \end{aligned}$$

Tako kot v (3.30) bosta tudi v tem regulacijskem zakonu uporabljena $v = v_{ref}e_{cos} + v_{fb}$ in $\omega = \omega_{ref} + \omega_{fb}$. Cilj vodenja je voditi e_x, e_y in e_s proti 0. Spremenljivka e_{cos} je pridobljena kot kosinus pogreška orientacije in jo je treba voditi proti 1. Zato bo nov pogrešek definiran kot $e_c = e_{cos} - 1$ in tako je končni model sistemskega pogreška

$$\begin{aligned} \dot{e}_x &= \omega_{ref}e_y - v_{fb} + e_y\omega_{fb} \\ \dot{e}_y &= -\omega_{ref}e_x + v_{ref}e_s - e_x\omega_{fb} \\ \dot{e}_s &= -e_c\omega_{fb} - \omega_{fb} \\ \dot{e}_c &= e_s\omega_{fb} \end{aligned} \tag{3.40}$$

Na podlagi pristopa Ljapunova bomo razvili regulator, ki doseže asimptotično stabilnost modela pogreška (3.40). Zelo neposredna ideja je uporaba sledeče funkcije Ljapunova

$$V_0 = \frac{k}{2}\left(e_x^2 + e_y^2\right) + \frac{1}{2}\left(e_s^2 + e_c^2\right) \tag{3.41}$$

Zanimivo, ta funkcija Ljapunova vodi do regulacijskega zakona (3.38). Vendar bo tukaj predlagana nekoliko bolj kompleksna funkcija, ki kot poseben primer vključuje tudi funkcijo (3.41). Za doseg cilja vodenja je predlagan naslednji kandidat za funkcijo Ljapunova

$$V = \frac{k}{2}\left(e_x^2 + e_y^2\right) + \frac{1}{2\left(1 + \frac{e_c}{a}\right)}\left(e_s^2 + e_c^2\right) \tag{3.42}$$

kjer sta $k > 0$ in $a > 2$ konstanti. Upoštevamo, da je $[-2,\ 0]$ območje funkcije $e_c = \cos(\varphi_{ref} - \varphi) - 1$ in zato

$$\begin{aligned} 0 < \frac{a-2}{a} \le 1 + \frac{e_c}{a} \le 1 \\ 1 \le \frac{1}{1 + \frac{e_c}{a}} \le \frac{a}{a-2} \end{aligned} \tag{3.43}$$

Zaradi (3.43) je funkcija V_0 (3.41) spodnja meja funkcije V (3.42), pa tudi V izpolnjuje pogoje za funkcijo Ljapunova. Vloga člena $(1 + \frac{e_c}{a})$ bo pojasnjena kasneje. Funkcijo V lahko poenostavimo na naslednji način

$$e_s^2 + e_c^2 = e_s^2 + (e_{cos} - 1)^2 = 2 - 2e_{cos} = -2e_c \tag{3.44}$$

Upoštevajoč enačbe modela pogreška (3.40) in (3.44), je odvod V (3.42) enak

$$\begin{aligned}\dot{V} &= -ke_x v_{fb} + kv_{ref} e_y e_s + \frac{1}{2\left(1+\frac{e_c}{a}\right)}(-2e_s\omega_{fb}) + \frac{-\frac{1}{a}e_s\omega_{fb}(-2e_c)}{2\left(1+\frac{e_c}{a}\right)^2} \\ &= -ke_x v_{fb} + e_s\left(kv_{ref}e_y - \frac{\omega_{fb}}{\left(1+\frac{e_c}{a}\right)^2}\right)\end{aligned}$$

Da bo $\dot{V}$ negativno semidefinitna, predlagamo sledeči regulacijski zakon

$$\begin{aligned}v_{fb} &= k_x e_x \\ \omega_{fb} &= kv_{ref}e_y\left(1+\frac{e_c}{a}\right)^2 + k_s e_s\left[\left(1+\frac{e_c}{a}\right)^2\right]^n\end{aligned} \tag{3.45}$$

kjer sta funkciji $k_x(t)$ in $k_s(t)$ pozitivni za $n \in \mathbb{Z}$. Iz praktičnih razlogov je n majhno število (običajno izberemo -2, -1, 0, 1 ali 2). Z upoštevanjem regulacijskega zakona (3.45) postane funkcija $\dot{V}$

$$\dot{V} = -kk_x e_x^2 - k_s e_s^2\left[\left(1+\frac{e_c}{a}\right)^2\right]^{n-1} \tag{3.46}$$

Ponovno je preprosto prikazati konvergenco e_x in e_s na podlagi (3.46). Prikaz konvergenc e_y in e_c pa je spet nekoliko zahtevnejši [14].

Primer 3.12

Vodite vozilo z diferencialnim pogonom, da sledi referenčni trajektoriji $x_{ref} = 1{,}1 + 0{,}7\sin(\frac{2\pi t}{30})$ in $y_{ref} = 0{,}9 + 0{,}7\sin(\frac{4\pi t}{30})$. Računski korak je $T_s = 0{,}033\,\mathrm{s}$, začetna lega pa je $[x(0), y(0), \varphi(0)] = [1{,}1,\ 0{,}8,\ 0]$. V Matlab kodi izvedite predstavljeni algoritem vodenja ter preizkusite različna ojačenja in dodatne parametre vodenja (a in n).

Rešitev

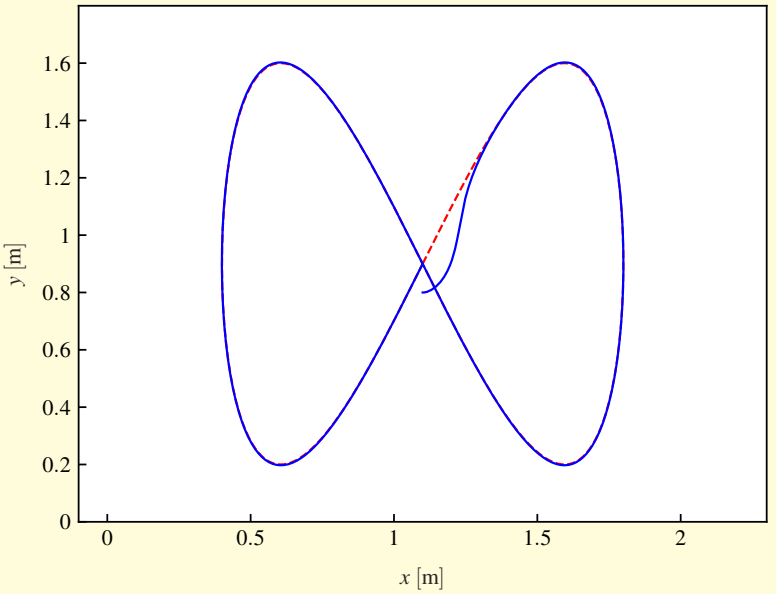
Matlab koda je navedena v programu 3.11. Rezultati simulacije so prikazani na slikah 3.28 in 3.29, kjer je predstavljeno dobro sledenje.

Program 3.11

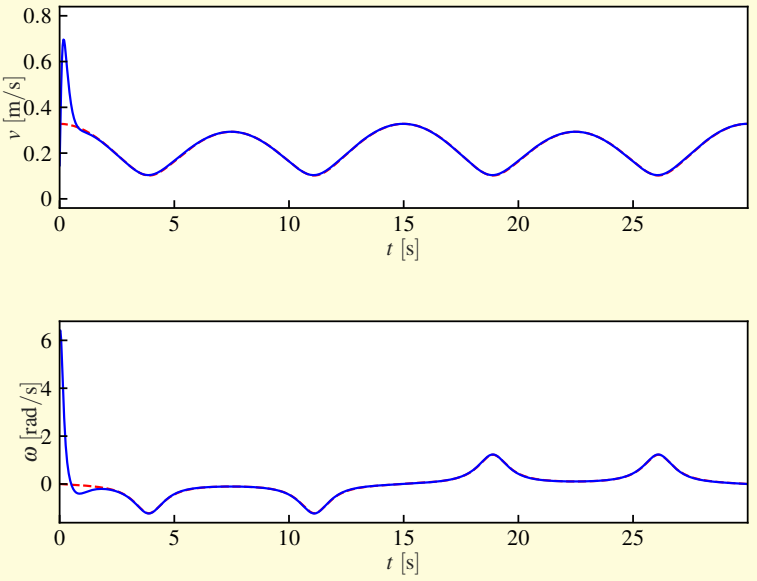
./src/ctr/example_tracking_four_state_control.m

```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
```

```
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna lega
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

for k = 1:length(t)
    e = [cos(q(3)), sin(q(3)), 0; ...
        -sin(q(3)), cos(q(3)), 0; ...
         0,         0,         1]*(qRef(:,k) - q); % Vektor pogreška
    eX = e(1); eY = e(2); % Pogrešek po razdalji
    eS = sin(e(3)); eCos = cos(e(3)); eC = eCos - 1; % Kotni pogrešek

    % Trenutni referenčni vhodi
    vRef = uRef(1,k);
    wRef = uRef(2,k);

    % Regulator
    zeta = 0.9; % Parameter za nastavljanje
    g = 85;     % Parameter za nastavljanje
    a = 10;     % Parameter za nastavljanje
    n = 2;      % Parameter za nastavljanje (celo število)
    Kx = 2*zeta*sqrt(wRef^2+g*vRef^2);
    Ks = Kx;
    K = g;
    % Ojačenji Kx in Ks sta lahko tudi konstantni.
    % Ta oblika omogoča, da je dušenje v prehodnem pojavu
    % neodvisno od referenčnih hitrosti.

    % Regulator: krmiljenje in regulacija
    v = vRef*cos(e(3)) + Kx*eX;
    w = wRef + K*vRef*eY*(1+eC/a)^2 + Ks*eS*(1+eC/a)^(2*n);

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.28: Nelinearno vodenje po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.12 (referenca je označena s črtkano krivuljo)



Slika 3.29: Regulirni signalni nelinearnega vodenja po referenčni trajektoriji vozila z diferencialnim pogonom iz primera 3.12

3.3.7 Načrtovanje mehkega vodenja Takagi-Sugeno v okviru linearnih matričnih neenačb

Kot smo že poudarili, je model pogreška (3.31) nelinearen. Modeli Takagi-Sugeno (TS) opisujejo dinamiko nelinearnih sistemov. V tem poglavju bo model (3.31) zapisan v obliki modela Takagi-Sugeno, ki omogoča zasnovo vodenja kot t. i. paralelno porazdeljena kompenzacija v okviru linearnih matričnih neenačb (LMI, angl. *linear matrix inequality*).

Mehki model pogreška Takagi-Sugeno kolesnega mobilnega robota z diferencialnim pogonom

Modeli Takagi-Sugeno (TS) imajo svoje korenine v mehki (angl. *fuzzy*) logiki, kjer je model podan v obliki pravil *če-potem* (angl. *if-then*). Model TS je lahko predstavljen tudi v bolj kompaktni obliki [15]

$$\dot{\boldsymbol{\xi}}(t) = \sum_{i=1}^{r} h_i(\boldsymbol{z}(t)) (\boldsymbol{A}_i \boldsymbol{\xi}(t) + \boldsymbol{B}_i \boldsymbol{u}(t))$$
$$\boldsymbol{v}(t) = \sum_{i=1}^{r} h_i(\boldsymbol{z}(t)) (\boldsymbol{C}_i \boldsymbol{\xi}(t))$$

kjer je $\boldsymbol{\xi}(t) \in \mathbb{R}^n$ vektor stanj, $\boldsymbol{v}(t) \in \mathbb{R}^p$ izhodni vektor in $\boldsymbol{z}(t) \in \mathbb{R}^q$ pogojni vektor, odvisen od vektorja stanj (ali neke druge veličine), $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$ pa so konstantne matrike. Nelinearne utežnostne funkcije $h_i(\boldsymbol{z}(t))$ so vse nenegativne in takšne, da velja $\sum_{i=1}^{r} h_i(\boldsymbol{z}(t)) = 1$ za poljuben $\boldsymbol{z}(t)$. Za vsak nelinearen model je mogoče najti enakovreden mehki model TS v kompaktnem območju spremenljivke prostora stanj z razdelitvijo nelinearnega področja, kjer se vsak omejeni nelinearni izraz razgradi v konveksno kombinacijo njegovih meja. Število pravil r je povezano s številom nelinearnosti modela, kot bo prikazano v nadaljevanju.

V tem poglavju bomo izkoristili dejstvo, da so v primeru modela pogreška kolesnega robota nelinearne funkcije znane vnaprej, kar omogoča uporabo prej omenjenega koncepta. Nelinearen model sledilnega pogreška (3.31) bo torej prepisan v enakovredno matrično obliko

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\varphi \end{bmatrix} = \begin{bmatrix} 0 & \omega_{ref} & 0 \\ -\omega_{ref} & 0 & v_{ref} \frac{\sin e_\varphi}{e_\varphi} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_\varphi \end{bmatrix} + \begin{bmatrix} -1 & e_y \\ 0 & -e_x \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_{fb} \\ \omega_{fb} \end{bmatrix} \tag{3.47}$$

V tem modelu se pojavijo štiri omejene nelinearne funkcije: ω_{ref}, $v_{ref}\frac{\sin e_\varphi}{e_\varphi}$ (ali z drugačno oznako $v_r\text{sinc}(\varphi)$), e_y in e_x. Tako dobimo pogojni vektor

$$\boldsymbol{z}(t) = \begin{bmatrix} \omega_{ref} \\ v_{ref}(t)\text{sinc}\left(e_\varphi(t)\right) \\ e_y(t) \\ e_x(t) \end{bmatrix}$$

Najprej bomo v linearnem smislu analizirali vodljivost modela (3.47). V bližini točke, v kateri je pogrešek sistema enak nič, je sistem (3.47) vodljiv, če je v_{ref} različen od 0 in $|e_\varphi|$ drugačen od π ali pa če je ω_{ref} različna od 0. V praktičnih primerih ω_{ref} pogosto prečka 0, zato v_{ref} ne more biti enak 0 in $|e_\varphi|$ ne more biti enak π. Da preprečimo izgubo vodljivosti in se osredotočimo na določeno kompaktno območje prostora pogreška, so potrebne naslednje predpostavke

$$\begin{aligned}
&\underline{\omega}_{ref} \le \omega_{ref} \le \bar{\omega}_{ref} \\
&|e_\varphi| \le \bar{e}_\varphi < \pi,\ 0 < \underline{v}_{ref} \le v_{ref} \le \bar{v}_{ref} \quad \Rightarrow \quad \underline{v}_{ref}\text{sinc}\left(\bar{e}_\varphi\right) \le v_{ref}\text{sinc}\left(e_\varphi\right) \le \bar{v}_{ref} \\
&|e_y| \le \bar{e}_y \\
&|e_x| \le \bar{e}_x
\end{aligned}$$

Meje od v_{ref} in ω_{ref} so pridobljene iz dejanske referenčne trajektorije, medtem ko so meje sledilnega pogreška izbrane na podlagi (predhodno) znanih informacij. Pomembno je, da so te meje nižje od pogreška zaradi merilnega šuma, začetnih pogreškov itd. Meje iz (3.3.7) označujemo kot $\underline{z}_j$ in $\bar{z}_j$, $j = 1, 2, 3, 4$. V sistemu so 4 nelinearnosti, zato je število pravil *če-potem* r enako $2^4 = 16$. Model TS (3.47) je

$$\dot{\boldsymbol{e}}\left(t\right) = \boldsymbol{A}_{\boldsymbol{z}(t)}\boldsymbol{e}\left(t\right) + \boldsymbol{B}_{\boldsymbol{z}(t)}\boldsymbol{u}_{fb}\left(t\right)$$

$$\boldsymbol{A}_i = \begin{bmatrix} 0 & \varepsilon_i^1 & 0 \\ -\varepsilon_i^1 & 0 & \varepsilon_i^2 \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{B}_i = \begin{bmatrix} -1 & \varepsilon_i^3 \\ 0 & -\varepsilon_i^4 \\ 0 & -1 \end{bmatrix}$$

Indeks i gre poljubno skozi vsa oglišča hiperkocke, ki jo definira (3.3.7). Običajno se uporabi binarno naštevanje

$$\begin{aligned}
i_1 &= \begin{cases} 0 & ; \quad i \le \frac{r}{2} \\ 1 & ; \quad \text{sicer} \end{cases} \\
i_2 &= \begin{cases} 0 & ; \quad i - \frac{r}{2}i_1 \le \frac{r}{4} \\ 1 & ; \quad \text{sicer} \end{cases} \\
i_3 &= \begin{cases} 0 & ; \quad i - \frac{r}{2}i_1 - \frac{r}{4}i_2 \le \frac{r}{8} \\ 1 & ; \quad \text{sicer} \end{cases} \\
i_4 &= \begin{cases} 0 & ; \quad i - \frac{r}{2}i_1 - \frac{r}{4}i_2 - \frac{r}{8}i_3 \le \frac{r}{16} \\ 1 & ; \quad \text{sicer} \end{cases}
\end{aligned}$$

Potem je ε_i^j v (3.3.7) opredeljen kot

$$\varepsilon_i^j = \underline{z}_j + i_j \left(\bar{z}_j - \underline{z}_j\right) \; ; \quad i = 1, 2, \ldots, 16, \quad j = 1, 2, 3, 4$$

Na koncu pa določimo še pripadnostne funkcije h_i

$$h_i(\boldsymbol{z}) = w_{i_1}^1(z_1) w_{i_2}^2(z_2) w_{i_3}^3(z_3) w_{i_4}^4(z_4) \qquad i = 1, 2, \ldots, 16$$
$$w_1^j(z_j) = \frac{z_j - \underline{z}_j}{\bar{z}_j - \underline{z}_j}, w_0^j(z_j) = 1 - w_1^j(z_j) \qquad j = 1, 2, 3, 4$$

Model TS (3.3.7) modela sledilnega pogreška predstavlja natančen model sistema (3.47), torej pri tem pristopu model TS ne deluje kot aproksimator, ampak upošteva vse znane nelinearnosti v sistemu. Tako je izraz (3.3.7) zelo primeren za naloge načrtovanja in analize, kot bo prikazano v nadaljevanju.

Vodenje kolesnega mobilnega robota z diferencialnim pogonom z uporabo paralelne porazdeljene kompenzacije

Za stabilizacijo modela TS (3.3.7) se uporablja paralelna porazdeljena kompenzacija (PDC, angl. *parallel distributed compensation*) [16]

$$\boldsymbol{u}_{fb}(t) = -\sum_{i=1}^{r} h_i(\boldsymbol{z}(t)) \boldsymbol{F}_i \boldsymbol{e}(t) = -\boldsymbol{F}_{\boldsymbol{z}(t)} e(t)$$

Problem stabilizacije z uporabo PDC je dobro znan. Zaradi posebne strukture, v kateri imata model naprave in regulator enake pripadnostne funkcije, je mogoče temu nelinearnemu sistemu prilagoditi določena orodja za analizo in načrtovanje linearnih sistemov. Še posebej pomembna je možnost formalne in neposredne obravnave stabilnosti sistema. V grobem je sistem, ki ga opisujeta (3.3.7) in (3.3.7), asimptotično stabilen, če je $(\boldsymbol{A}_i - \boldsymbol{B}_i \boldsymbol{F}_j)$ Hurwitzeva matrika za vsak i in j, kar pomeni, da vsi njeni poli ležijo na levi polovici kompleksne ravnine s. Število matrik, potrebnih za analizo, zelo hitro narašča, zato uporabimo sistematičen pristop. Dokaj kmalu so ugotovili, da je LMI popolno orodje za to nalogo [17]. Parametri naprave so podani v obliki matrik $\boldsymbol{A}_i$ in $\boldsymbol{B}_i$, zato je mogoče najti tak nabor parametrov regulatorja $\boldsymbol{F}_j$, ki asimptotično stabilizirajo sistem.

Prvotni pristop je preveč konzervativen, saj ne upošteva posebnih lastnosti sistema, kot je oblika pripadnostnih funkcij ipd. Izvirne pogoje LMI je možno omiliti. Prilagoditev rezultata glede na [18] je:

Model TS (3.3.7) je mogoče stabilizirati z regulacijskim zakonom PDC (3.3.7), če obstajajo matrike $\boldsymbol{M}_i$ ($i = 1, 2, \ldots, r$) in $\boldsymbol{X} > 0$, tako da veljata naslednja pogoja LMI

$$\boldsymbol{\Upsilon}_{ii} < 0 \; ; \quad i = 1, 2, \ldots, r$$
$$\frac{2}{r-1} \boldsymbol{\Upsilon}_{ii} + \boldsymbol{\Upsilon}_{ij} + \boldsymbol{\Upsilon}_{ji} < 0 \qquad i, \; j = 1, 2, \ldots, r, \quad i \neq j$$

kjer je $\boldsymbol{\Upsilon}_{ij} = \boldsymbol{X}\boldsymbol{A}_i^T + \boldsymbol{A}_i \boldsymbol{X} - \boldsymbol{M}_j^T \boldsymbol{B}_i^T - \boldsymbol{B}_i \boldsymbol{M}_j$. Ojačenja $\boldsymbol{F}_i$ regulacijskega zakona PDC (3.3.7) podaja $\boldsymbol{F}_i = \boldsymbol{M}_i \boldsymbol{X}^{-1}$.

3.3.8 Modelno prediktivno vodenje

Modelno prediktivno vodenje (MPC, angl. *model-based predictive control*) temelji na naprednih metodah, ki jih je mogoče uporabiti na različnih področjih; tudi v mobilni robotiki, kjer je referenčna trajektorija znana vnaprej. Uporaba prediktivnih pristopov vodenja v mobilni robotiki se običajno nanaša na lineariziran (lahko tudi nelinearen) kinematični ali dinamični model za napovedovanje stanj sistema. Znane so številne uspešne implementacije v kolesni mobilni robotiki, kot so posplošeno prediktivno vodenje v [19], prediktivno vodenje s Smithovim prediktorjem za upravljanje časovne zakasnitve sistema v [20], MPC na osnovi linearnega časovno spremenljivega sistema v [21], nelinearni prediktivni regulator z modelom sistema v obliki večplastnega nevronskega omrežja v [22] in mnoge druge. Rešitve regulacijskih zakonov so v večini pristopov pridobljene z optimizacijo cenilke. Drugi pristopi pridobijo regulacijski zakon kot analitično rešitev, ki je računsko učinkovita in jo je mogoče enostavno uporabiti pri hitrih izvedbah v realnem času [23].

To poglavje obravnava mobilnega robota z diferencialnim pogonom in vodenje po referenčni trajektoriji, ki mora biti zvezna in dvakrat zvezno odvedljiva funkcija časa. Za napovedovanje se uporablja linearni dinamični model pogreška, ki ga pridobimo z linearizacijo sistema okoli referenčne trajektorije. Regulator zmanjšuje razliko med napovedanim sledilnim pogreškom robota in referenčnim pogreškom z definirano želeno dinamiko.

Modelne strategije vodenja združujejo rešitev predkrmiljenja in akcija povratne zanke v vhodnem vektorju u, kar zapišemo kot

$$\boldsymbol{u} = \boldsymbol{u}_{ff} + \boldsymbol{u}_{fb} = \begin{bmatrix} v_{ref} \cos e_{\varphi} + v_{fb} \\ \omega_{ref} + \omega_{fb} \end{bmatrix}$$

kjer se vhodni vektor predkrmiljenja $\boldsymbol{u}_{ff} = [v_{ref} \cos e_{\varphi}\ \omega_{ref}]^T$ izračuna iz referenčne trajektorije z uporabo relacij (3.17) in (3.18). Vhodni vektor povratne zanke je $\boldsymbol{u}_{fb} = [v_{fb}\, \omega_{fb}]^T$, kar je izhod MPC regulatorja.

Problem vodenja upošteva linearen dinamični model sledilnega pogreška (3.32), ki ga lahko na kratko zapišemo kot

$$\dot{\boldsymbol{e}} = \boldsymbol{A}_c(t)\boldsymbol{e} + \boldsymbol{B}_c\boldsymbol{u}_{fb} \tag{3.48}$$

kjer sta $\boldsymbol{A}_c(t)$ in $\boldsymbol{B}_c$ matriki zveznega modela prostora stanj in $\boldsymbol{e}$ je sledilni pogrešek v lokalnih koordinatah robota, ki so določene s transformacijo (3.28) in prikazane na sliki 3.21.

Diskretno modelno prediktivno vodenje

Modelno prediktivno vodenje (MPC), predstavljeno v [23], je zasnovano za diskretne čase, zato je potrebno zapisati model (3.48) v diskretni obliki

$$\dot{\boldsymbol{e}}(k+1) = \boldsymbol{A}(k)\boldsymbol{e}(k) + \boldsymbol{B}\boldsymbol{u}_{fb}(k) \tag{3.49}$$

kjer je $\boldsymbol{A}(k) \in \mathbb{R}^n \times \mathbb{R}^n$, n je število spremenljivk stanj, in $\boldsymbol{B} \in \mathbb{R}^n \times \mathbb{R}^m$, m je število vhodnih spremenljivk. Diskretni matriki $\boldsymbol{A}(k)$ in $\boldsymbol{B}$ dobimo na naslednji način

$$\begin{aligned} \boldsymbol{A}(k) &= \boldsymbol{I} + \boldsymbol{A}_c(t)T_s \\ \boldsymbol{B} &= \boldsymbol{B}_c T_s \end{aligned}$$

kar je zadosten približek za kratek računski korak T_s.

Glavna ideja MPC je izračunati optimalne akcije vodenja, ki minimizirajo kriterijsko funkcijo, določeno v intervalu predikcijskega horizonta h. Kriterijska funkcija je kvadratna cenilka

$$J(\boldsymbol{u}_{fb}, k) = \sum_{i=1}^{h} \boldsymbol{\epsilon}^T(k,i)\boldsymbol{Q}\boldsymbol{\epsilon}(k,i) + \boldsymbol{u}_{fb}^T(k+i-1)\boldsymbol{R}\boldsymbol{u}_{fb}(k+i-1) \tag{3.50}$$

sestavljena iz prihodnjega referenčnega sledilnega pogreška $\boldsymbol{e}_r(k+i)$, napovedanega sledilnega pogreška $\boldsymbol{e}(k+i|k)$, razlike med omenjenima pogreškoma $\boldsymbol{\epsilon}(k,i) = \boldsymbol{e_r}(k+i) - \boldsymbol{e}(k+i|k)$ in prihodnje akcije $\boldsymbol{u}_{fb}(k+i-1)$, kjer i označuje i-ti korak napovedi ($i = 1, \ldots, h$); $\boldsymbol{Q}$ in $\boldsymbol{R}$ sta utežnostni matriki.

Za napoved stanja $\boldsymbol{e}(k+i|k)$ se uporabi model pogreška (3.49), kot sledi

$$\begin{aligned} \boldsymbol{e}(k+1|k) &= \boldsymbol{A}(k)\boldsymbol{e}(k) + \boldsymbol{B}\boldsymbol{u}_{fb}(k) \\ \boldsymbol{e}(k+2|k) &= \boldsymbol{A}(k+1)\boldsymbol{e}(k+1|k) + \boldsymbol{B}\boldsymbol{u}_{fb}(k+1) \\ &\vdots \\ \boldsymbol{e}(k+i|k) &= \boldsymbol{A}(k+i-1)\boldsymbol{e}(k+i-1|k) + \boldsymbol{B}\boldsymbol{u}_{fb}(k+i-1) \\ &\vdots \\ \boldsymbol{e}(k+h|k) &= \boldsymbol{A}(k+h-1)\boldsymbol{e}(k+h-1|k) + \boldsymbol{B}\boldsymbol{u}_{fb}(k+h-1) \end{aligned} \tag{3.51}$$

Napovedi $\boldsymbol{e}(k+i|k)$ v (3.51) so preurejene tako, da so odvisne od trenutnega pogreška $\boldsymbol{e}(k)$, trenutnih in prihodnjih vhodov $\boldsymbol{u}_{fb}(k+i-1)$ ter matrik $\boldsymbol{A}(k+i-1)$ in $\boldsymbol{B}$. Napoved izhoda modela v trenutku h lahko potem zapišemo kot

$$\begin{aligned} \boldsymbol{e}(k+h|k) = \Pi_{j=1}^{h-1}\boldsymbol{A}(k+j)\boldsymbol{e}(k) + \\ + \sum_{i=1}^{h} \left(\Pi_{j=i}^{h-1}\boldsymbol{A}(k+j)\right)\boldsymbol{B}u_{fb}(k+i-1) + \boldsymbol{B}\boldsymbol{u}_{fb}(k+h-1) \end{aligned}$$

Prihodnji referenčni pogrešek ($\boldsymbol{e}_r(k+i)$) določa, kako naj se zmanjša sledilni pogrešek, ko robot ni na trajektoriji. Določimo lahko, da naj se prihodnji referenčni pogrešek eksponentno zmanjša od trenutnega sledilnega pogreška $\boldsymbol{e}(k)$ kot

$$\boldsymbol{e}_r(k+i) = \boldsymbol{A}_r^i e(k)$$

za $i = 1, \ldots, h$. Dinamiko referenčnega pogreška določa matrika referenčnega modela $\boldsymbol{A}_r$.

Glede na (3.51) in (3.3.8) je določen vektor predikcijskega pogreška sledenja robota

$$\boldsymbol{E}^*(k) = \begin{bmatrix} \boldsymbol{e}^T(k+1|k) & \boldsymbol{e}^T(k+2|k) & \dots & \boldsymbol{e}^T(k+h|k) \end{bmatrix}^T$$

pri čemer je $\boldsymbol{E}^*$ podan za celoten interval opazovanja (h), kjer je regulirni vektor

$$\boldsymbol{U}_{fb}(k) = \begin{bmatrix} \boldsymbol{u}_{fb}^T(k) & \boldsymbol{u}_{fb}^T(k+1) & \dots & \boldsymbol{u}_{fb}^T(k+h-1) \end{bmatrix}^T \tag{3.52}$$

in

$$\boldsymbol{\Lambda}(k,i) = \Pi_{j=i}^{h-1} \boldsymbol{A}(k+j) = \boldsymbol{A}(k+h-1)\boldsymbol{A}(k+h-2)\dots\boldsymbol{A}(k+i+1)\boldsymbol{A}(k+i)$$

Vektor predikcijskega pogreška sledenja robota lahko zapišemo v strnjeni obliki

$$\boldsymbol{E}^*(k) = \boldsymbol{F}(k)\boldsymbol{e}(k) + \boldsymbol{G}(k)\boldsymbol{U}_{fb}(k)$$

kjer je

$$\boldsymbol{F}(k) = \begin{bmatrix} \boldsymbol{A}^T(k) & \boldsymbol{A}^T(k)\boldsymbol{A}^T(k+1) & \dots & \boldsymbol{\Lambda}^T(k,0) \end{bmatrix}^T$$

in

$$\boldsymbol{G}(k) = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{A}(k+1)\boldsymbol{B} & \boldsymbol{B} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Lambda}(k,1)\boldsymbol{B} & \boldsymbol{\Lambda}(k,2)\boldsymbol{B} & \dots & \boldsymbol{B} \end{bmatrix}$$

pri čemer sta dimenziji $\boldsymbol{F}(k)$ in $\boldsymbol{G}(k)$ enaki $(nh \times n)$ ter $(nh \times mh)$.

Vektor referenčnega sledilnega pogreška je

$$\boldsymbol{E}_r^*(k) = \begin{bmatrix} \boldsymbol{e}_r^T(k+1) & \boldsymbol{e}_r^T(k+2) & \dots & \boldsymbol{e}_r^T(k+h) \end{bmatrix}^T$$

ki se v strnjeni obliki zapiše kot

$$\boldsymbol{E}_r^*(k) = \boldsymbol{F}_r e(k)$$

kjer je

$$\boldsymbol{F}_r = \begin{bmatrix} \boldsymbol{A}_r^T & \left(\boldsymbol{A}_r^2\right)^T & \dots & \left(\boldsymbol{A}_r^h\right)^T \end{bmatrix}^T$$

matrika dimenzije $(nh \times n)$.

Optimalni vhodni vektor (3.52) dobimo z numerično ali analitično optimizacijo funkcije (3.50). V nadaljevanju bo izpeljana analitična rešitev.

Kriterijska funkcija (3.50) se v matrični obliki glasi

$$J(\boldsymbol{U}_{fb}) = \left(\boldsymbol{E}_r^* - \boldsymbol{E}^*\right)^T \overline{\boldsymbol{Q}} \left(\boldsymbol{E}_r^* - \boldsymbol{E}^*\right) + \boldsymbol{U}_{fb}^T \overline{\boldsymbol{R}} \boldsymbol{U}_{fb} \tag{3.53}$$

minimum (3.53) pa je izražen kot

$$\frac{\partial J}{\partial \boldsymbol{U}_{fb}} = -2\overline{\boldsymbol{Q}}\boldsymbol{G}^T\boldsymbol{E}_r^* + 2\boldsymbol{G}^T\overline{\boldsymbol{Q}}\boldsymbol{E}^* + 2\overline{\boldsymbol{R}}\boldsymbol{U}_{fb} = 0$$

Dobimo rešitev za optimalni vhodni vektor kot

$$\boldsymbol{U}_{fb}(k) = \left(\boldsymbol{G}^T\overline{\boldsymbol{Q}}\boldsymbol{G} + \overline{\boldsymbol{R}}\right)^{-1}\boldsymbol{G}^T\overline{\boldsymbol{Q}}\left(\boldsymbol{F}_r - \boldsymbol{F}\right)\boldsymbol{e}(k) \tag{3.54}$$

kjer sta utežnostni matriki naslednji

$$\overline{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{Q} \end{bmatrix} \qquad \overline{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{R} \end{bmatrix}$$

Rešitev (3.54) vsebuje vhodne vektorje $u_{fb}^T(k+i-1)$ za celoten interval napovedi ($i = 1, \dots, h$). Akcijo povratne zanke v trenutku k uveljavimo tako, da na robotu uporabimo samo prvi vektor $u_{fb}^T(k)$ (prve m vrstice od $\boldsymbol{U}_{fb}(k)$). Rešitev je pridobljena analitično, zato omogoča hitre izvedbe v realnem času, kar morda ni možno, če uporabimo numerično optimizacijo funkcije (3.50).

Primer 3.13

Izvedite modelno prediktivno vodenje, podano v (3.54), za vodenje robota z diferencialnim pogonom po trajektoriji. Referenčna trajektorija in robot sta določena v primeru 3.9.

Predikcijski horizont je $h = 4$, matrika referenčnega modela je $\boldsymbol{A}_r = \boldsymbol{I}_{3\times 3} \cdot 0{,}65$, utežnostni matriki pa sta

$$\boldsymbol{Q} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 40 & 0 \\ 0 & 0 & 0{,}1 \end{bmatrix} \qquad \boldsymbol{R} = \boldsymbol{I}_{2\times 2} \cdot 10^{-3}$$

Rešitev

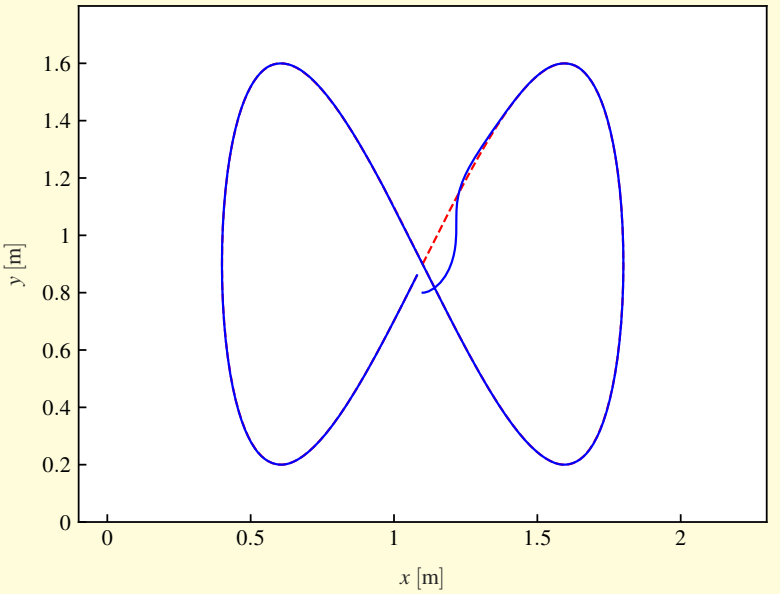
Z uporabo MPC izračunamo povratnozančni del krmilnega signala $\boldsymbol{u}_{fb}(k)$ in ga uporabimo na robotu skupaj s predkrmiljenjem $\boldsymbol{u}_{ff}(k)$. V programu 3.12) so podane možne rešitve in rezultati vodenja po referenčni trajektoriji. Pridobljeni rezultati simulacije so prikazani na slikah 3.30 in 3.31.

Program 3.12

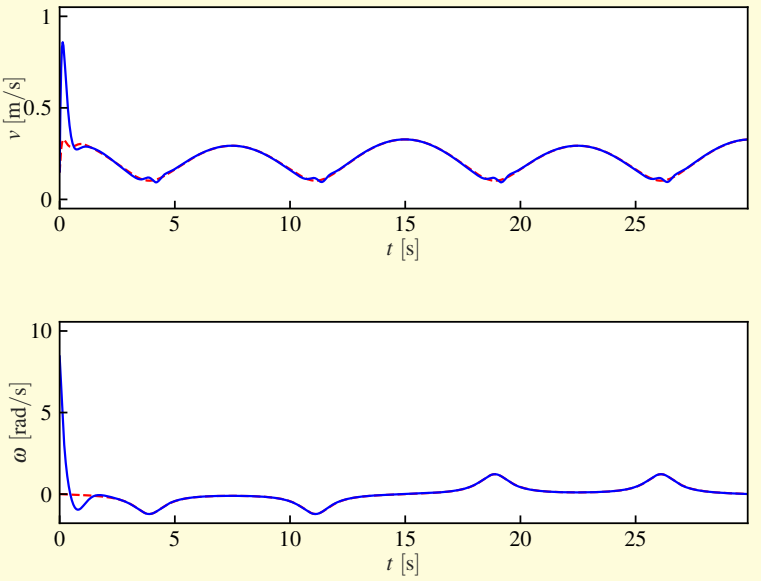
./src/ctr/example_tracking_mpc.m

```
1 Ts = 0.033; % Računski korak
2 t = 0:Ts:30; % Čas simulacije
3 q = [1.1; 0.8; 0]; % Začetna lega
4
5 % Referenca
6 freq = 2*pi/30;
7   xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
```

```
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

for k = 1:length(t)-4
    e = [cos(q(3)), sin(q(3)), 0; ...
        -sin(q(3)), cos(q(3)), 0; ...
         0,         0,         1]*(qRef(:,k) - q); % Vektor pogreška
    e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]

    A1 = [1, Ts*uRef(2,k),   0;-Ts*uRef(2,k),   1, Ts*uRef(1,k);   0,0,1];
    A2 = [1, Ts*uRef(2,k+1), 0;-Ts*uRef(2,k+1), 1, Ts*uRef(1,k+1); 0,0,1];
    A3 = [1, Ts*uRef(2,k+2), 0;-Ts*uRef(2,k+2), 1, Ts*uRef(1,k+2); 0,0,1];
    A4 = [1, Ts*uRef(2,k+3), 0;-Ts*uRef(2,k+3), 1, Ts*uRef(1,k+3); 0,0,1];
    B = [-Ts, 0; 0, 0; 0, -Ts];

    Z = zeros(3,2);
    Hm = [B,           Z,        Z,    Z; ...
          A1*B,        B,        Z,    Z; ...
          A1*A2*B,     A1*B,     B,    Z; ...
          A1*A2*A3*B, A1*A2*B, A1*B, B];
    Fm = [A1, A1*A2, A1*A2*A3, A1*A2*A3*A4].';

    ar = 0.65;
    Ar = eye(3)*ar; % Dinamika referenčnega pogreška
    H = 0;
    Fr = [Ar^(H+1), Ar^(H+2), Ar^(H+3), Ar^(H+4)].';

    % Utežnostne matrike
    Qt = diag(repmat([1; 40; 0.1], 4, 1));
    Rt = diag(repmat([0.001; 0.001], 4, 1));

    % Izračun optimalnih regulirnih signalov
    KKgpc = (Hm.'*Qt*Hm + Rt)\(Hm.'*Qt*(Fr-Fm));
    KK = KKgpc(1:2,:); % Izbira trenutnih ojačenj regulatorja

    v = KK*e;
    uF = [uRef(1,k)*cos(e(3)); uRef(2,k)];
    u = v + uF;

    vMAX = 1; wMAX = 15; % Maksimalni hitrosti
    if abs(u(1))>vMAX, u(1) = sign(u(1))*vMAX; end
    if abs(u(2))>wMAX, u(2) = sign(u(2))*wMAX; end

    % Simulacija gibanja robota
    dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 3.30: Rezultati vodenja, pridobljeni z eksplicitnim regulatorjem MPC (referenca je označena s črtkano krivuljo)



Slika 3.31: Vzbujanje robota izračunano z uporabo eksplicitnega MPC-ja (črtkana krivulja predstavlja le signal predkrmiljenja, brez regulirnega signala)

3.3.9 Vodenje na podlagi optimizacije z rojem delcev

Vodenje mobilnega robota lahko interpretiramo tudi kot optimizacijski problem, kjer je potrebno v vsakem računskem koraku regulacijske zanke najti najboljšo rešitev med vsemi možnimi rešitvami v iskalnem prostoru. Tako vodenje nima eksplicitne strukture, kar pomeni, da regulacijskega zakona ni mogoče podati kot funkcijo preslikave stanj sistema v akcije vodenja. Optimalno rešitev, ki minimalizira nekatere kriterijske funkcije, najdemo z uporabo iterativnega optimizacijskega algoritma, kot so Newtonove metode, metode gradientnega sestopa ali pa stohastične metode, kot so genetski algoritmi, optimizacija z rojem delcev (PSO, angl. *particle swarm optimization*) ipd.

Če kriterijska funkcija ni konveksna za problem minimizacije, je lahko večina optimizacijskih algoritmov ujetih v lokalnem minimumu, kar pa ni optimalna rešitev. Verjetnost pojava take rešitve lahko zmanjšamo z uporabo stohastične optimizacije, kjer je vzorec iskanja do neke mere naključen.

Osnovna ideja PSO izhaja iz družabnega vedenja malih živali, kot so jate ptic ali rib [24, 25]. PSO uporablja skupino (roj) delcev, kjer vsak delec predstavlja svojo hipotetično rešitev. Vsak delec i je opisan s parametričnim vektorjem $\boldsymbol{x}_i$, ki določa njegov položaj v parametričnem prostoru, in inkrementalnim vektorjem $\boldsymbol{v}_i$, ki določa njegovo hitrost v parametričnem prostoru. Med optimizacijo se populacija vseh potencialnih rešitev posodablja glede na kriterijsko funkcijo, ki določa merilo kakovosti. Vsak delec spremlja svoje parametre in si zapomni njihove (doslej) najboljše vrednosti $\boldsymbol{pBest}_i$ skupaj s pripadajočo kriterijsko funkcijo $J_i = f(\boldsymbol{pBest}_i)$. Med optimizacijo je shranjen tudi (doslej) najboljši parametrični vektor za celoten roj $\boldsymbol{pBest}_i$. V lokalni različici PSO pa se za neko okolico delcev spremlja najboljši parametrični vektor $\boldsymbol{gBest}$ vsakega delca (okolica se določi s topologijo obroča, k najbližjimi delci ipd. [26]). Za lokalne različice PSO je manj verjetno, da bodo ujete v lokalnem minimumu.

V nadaljevanju je razložena osnovna (globalna) različica PSO. V vsakem računskem koraku regulacijske zanke delci posodobijo svoje kognitivno in socialno vedenje glede na naslednja pravila

$$\begin{aligned} \boldsymbol{v}_i &\leftarrow \omega \boldsymbol{v}_i + c_1 \mathbf{rand}_{(0,1)}(\boldsymbol{pBest}_i - \boldsymbol{x}_i) + c_2 \mathbf{rand}_{(0,1)}(\boldsymbol{gBest} - \boldsymbol{x}_i) \\ \boldsymbol{x}_i &\leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i \end{aligned} \tag{3.55}$$

kjer je ω faktor vztrajnosti, c_1 samozavedna konstanta in c_2 socialna konstanta. Poleg tega je $\mathbf{rand}_{(0,\ 1)}$ vektor enakomerno razporejenih vrednosti v območju $(0,\ 1)$. Dimenzije vektorjev v (3.55) so enake dimenziji iskalnega prostora.

Parametri ω, c_1 in c_2 so pozitivni nastavitveni parametri. Osnovna koda PSO je podana v algoritmu 1.

Algorithm 1 Optimizacija z rojem delcev

Inicializacija:
for vsak delec $i = 1, \ldots, N$ **do**
  Naključno inicializiraj položaje delcev $\boldsymbol{x}_i$ znotraj meja parametričnega prostora.
  Naključno inicializiraj hitrost delcev $\boldsymbol{v}_i$ ali jih nastavi na nič.
  Nastavi $\boldsymbol{pBest}_i = \boldsymbol{x}_i$.
end for

Optimizacija:
$J_{best} = \infty$
repeat
  for vsak delec $i = 1, \ldots, N$ **do**
    Izračunaj trenutno kriterijsko funkcijo $J_i = f(\boldsymbol{x}_i)$ za vsak delec.
    Shrani najboljše parametre
    if $J_i < f(\boldsymbol{pBest}_i)$ **then**
      $\boldsymbol{pBest}_i = \boldsymbol{x}_i$
    end if
    if $f(\boldsymbol{pBest}_i) < J_{best}$ **then**
      $\boldsymbol{gBest} = \boldsymbol{pBest}_i$
      $J_{best} = f(\boldsymbol{gBest})$
    end if
  end for
  for vsak delec $i = 1, \ldots, N$ **do**
    Posodobi hitrost in položaj delca
    $\boldsymbol{v}_i \leftarrow \omega \boldsymbol{v}_i + c_1 \mathbf{rand}_{(0,\ 1)}(\boldsymbol{pBest}_i - \boldsymbol{x}_i) + c_2 \mathbf{rand}_{(0,\ 1)}(\boldsymbol{gBest} - \boldsymbol{x}_i)$
    Preveri, ali je hitrost $\boldsymbol{v}_i$ izvedljiva:
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$
  end for
until največje število iteracij ali izpolnjen kriterij za konvergenco

Primer 3.14

S pomočjo PSO določite regulator za robota in trajektorijo iz primera 3.9. PSO uporabite v vsakem računskem koraku $t = kT_s$, da najdete najboljše regulirne veličine (translatorna in kotna hitrost), ki pripeljejo robota čim bližje trenutnemu referenčnemu položaju $x_{ref}(t)$, $y_{ref}(t)$. Nato izračunajte predikcijo lege robota glede na njegovo kinematiko in predlagano rešitev delcev (akcija regulatorja). Cenilka je sestavljena iz sledilnega pogreška $\boldsymbol{e}(t) = [e_x(t), e_y(t), e_\varphi(t)]^T$ in povratne zanke $\boldsymbol{u}_{fb} = [v_{fb}, \omega_{fb}]^T$. Optimalno vodenje minimizira cenilko

$J(t) = \boldsymbol{e}(t)^T \boldsymbol{Q} \boldsymbol{e}(t) + \boldsymbol{u}_{fb}^T \boldsymbol{R} \boldsymbol{u}_{fb}$, kjer sta $\boldsymbol{Q}$ in $\boldsymbol{R}$ diagonalni utežnostni matriki, ki se uporabljata za nastavitev delovanja regulatorja.

Rešitev

Regulacijski zakon vključuje podobne akcije predkrmiljenja in povratne zanke kot v primeru 3.9. Akcijo predkrmiljenja izračunamo iz znane trajektorije, medtem ko akcijo povratne zanke določimo z uporabo PSO. Sledilni pogrešek je izražen v lokalnih koordinatah robota (glejte sliko 3.21), saj je s tem optimizacija bolj učinkovita, zaradi bolje razklopljenega zaprtozančnega sistema. Pogrešek v lokalni x koordinati je lahko preprosto kompenziran s translatorno hitrostjo, pogrešek v y in φ pa s kotno hitrostjo. To namreč ne velja pri uporabi globalnega sledilnega pogreška, zaradi nelinearne transformacije rotacije v (3.28).

V programu 3.13 je podana Matlab koda možne rešitve. Pridobljeni rezultati simulacije so prikazani na slikah 3.32 in 3.33. Rezultati vodenja so podobni kot v primeru 3.13, vendar je računska zahtevnost algoritma precej večja. Rezultati primera 3.14 niso deterministični, ker PSO pri optimizaciji uporablja naključno porazdeljene delce, da najde najboljše regulirne veličine v vsakem računskem koraku. Zato se lahko pridobljene trajektorije mobilnega robota do neke mere razlikujejo, še posebej v začetni prehodni fazi dokler robot ne doseže reference. Kljub temu so rezultati večine simulacijskih tekov primerljivi z rezultati drugih predstavljenih deterministčnih regulatorjev (npr. iz primera 3.9).

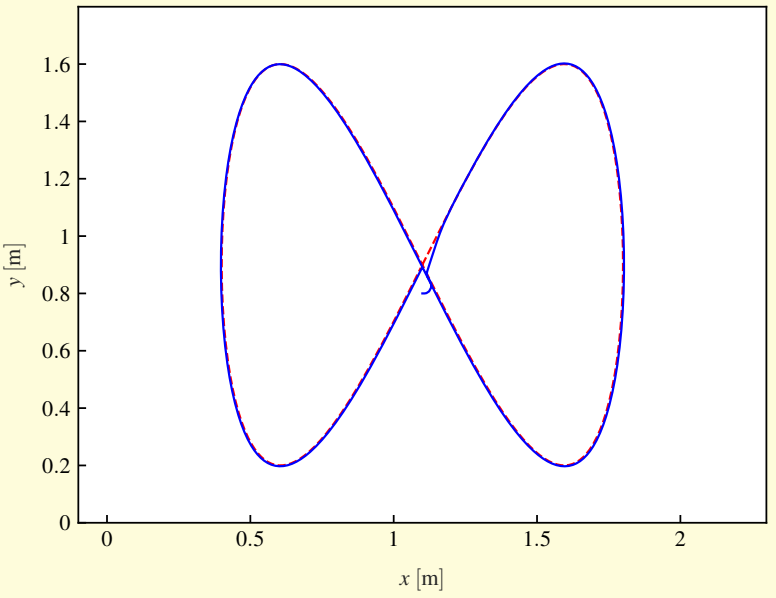
Program 3.13

./src/ctr/example_tracking_pso.m

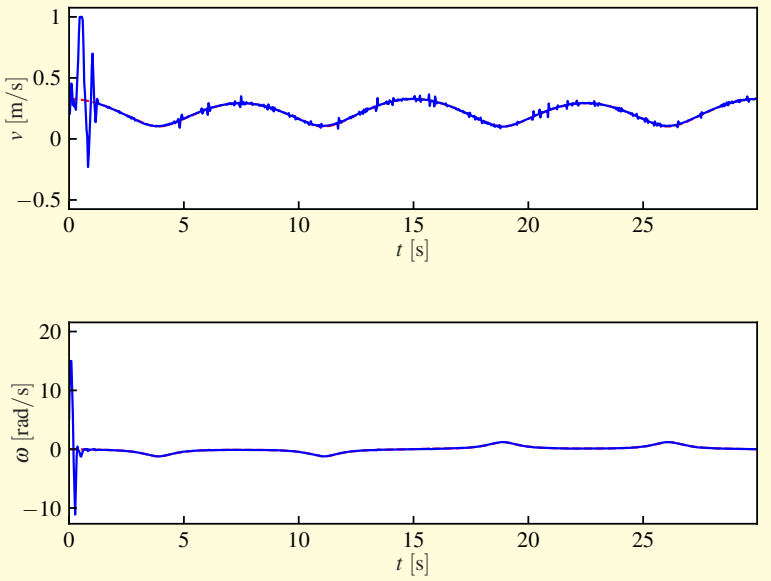
```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

vMax = 1; wMax = 15; % Omejitve hitrosti

% Inicializacija roja delcev
iterations = 20; % Število iteracij
omega = 0.5*0.5; % Faktor vztrajnosti
c1 = 0.5*1; % Samozavedna konstanta
c2 = 0.5*1; % Socialna konstanta
N = 25; % Velikost roja delcev
swarm = zeros([2,N,4]);
```

```
uBest = [0; 0];

for k = 1:length(t)-1
    % Začetni položaji delcev
    swarm(:,:,1) = repmat(uBest, 1, N) + diag([0.1; 3])*randn(2,N);
    swarm(:,:,2) = 0; % Začetne hitrosti delcev
    swarm(1,:,4) = 1000; % Najboljša vrednost kriterijske funkcije

    for iter = 1:iterations % Iterativno iskanje optimalne rešitve s PSO
        % Vrednotenje parametrov delcev
        for i = 1:N
            % Izračun nove predvidene lege robota na podlagi parametrov
            % i-tega delca (vhodnih hitrosti) in primerjava predvidene
            % lege z referenčno lego.
            vwi = swarm(:,i,1);
            ui = vwi + uRef(:,k); % Regulacija in krmiljenje
            qk = q; % Trenutna lega robota
            % Predikcija lege robota na podlagi parametrov delcev (hitrosti)
            qk = qk + Ts*[cos(qk(3)), 0; sin(qk(3)), 0; 0, 1]*ui;
            qk(3) = wrapToPi(qk(3)); % Zapis kota v območju [-pi, pi]
            e = [cos(qk(3)), sin(qk(3)), 0; ...
                -sin(qk(3)), cos(qk(3)), 0; ...
                 0,          0,          1]*(qRef(:,k+1)-qk); % Pogrešek
            e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]
            Qt = diag([4; 80; 0.1]); Rt = diag([1; 1]*0.0001); % Uteži
            J = e.'*Qt*e + vwi.'*Rt*vwi; % Kriterijska funkcija
            if J<swarm(1,i,4) % Če je novi parameter boljši, posodobi:
                swarm(:,i,3) = swarm(:,i,1); % vrednosti parametrov (v in w)
                swarm(1,i,4) = J; % in najboljšo vrednost kriterijske funkcije.
            end
        end
        [~, gBest] = min(swarm(1,:,4)); % Parametri globalno najboljšega delca

        % Posodobitev parametrov s hitrostnimi vektorji
        a = omega*swarm(:,:,2) + ...
            c1*rand(2,N).*(swarm(:,:,3) - swarm(:,:,1)) + ...
            c2*rand(2,N).*(repmat(swarm(:,gBest,3), 1, N) - swarm(:,:,1));
        % Maksimalna sprememba parametrov, pospešek: aMax=3 ==> 3*Ts=0.1
        a(1,a(1,:)>0.1) = 0.1; a(1,a(1,:)<-0.1) = -0.1;
        % Maksimalna sprememba parametrov, kotni pospešek: aMax=60 ==> 60*Ts=2
        a(2,a(1,:)>2) = 2;     a(2,a(1,:)<-2) = -2;

        v = swarm(:,:,1) + a; % Posodobitev hitrosti
        % Omejitev hitrosti z ohranjanjem ukrivljenosti
        [m, ii] = max([v(1,:)/vMax; v(2,:)/wMax; ones(1,N)]);
        i = ii==1; v(1,i) = sign(v(1,i))*vMax;
                   v(2,i) = v(2,i)./m(i);
        i = ii==2; v(2,i) = sign(v(2,i))*wMax;
                   v(1,i) = v(1,i)./m(i);

        swarm(:,:,2) = a; % Posodobitev hitrosti delcev (pospeški)
        swarm(:,:,1) = v; % Posodobitev položajev delcev (hitrosti)
    end

    % Vzmememo najboljši delec za izračun regulirnih signalov
    uBest = swarm(:,gBest,1);
    u = uBest + uRef(:,k); % Regulacija in krmiljenje

    % Omejitve hitrosti
    if abs(u(1))>vMax, u(1) = sign(u(1))*vMax; end
    if abs(u(2))>wMax, u(2) = sign(u(2))*wMax; end

    % Simulacija gibanja robota
```

```
87      dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
88      noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
89      q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
90      q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
91  end
```



Slika 3.32: Pridobljeni rezultati vodenja na podlagi optimizacije z rojem delcev (referenca je označena s črtkano krivuljo)



Slika 3.33: Akcija vodenja, izračunana na podlagi optimizacije z rojem delcev (črtkana krivulja predstavlja le signal predkrmiljenja, brez regulirnega signala)

PSO je splošni algoritem, ki ga je mogoče uporabiti v številnih aplikacijah, dokler računska zahtevnost in čas, potrebna za izračun rešitve, ustrezata dejanskim realno-časovnim zahtevam vodenega sistema.

Modelno prediktivno vodenje z uporabo optimizacije z rojem delcev

Reševanje optimizacije kriterijske funkcije v MPC lahko opravi tudi PSO, če le-ta išče najboljše parametre (akcije regulatorja) za predikcijski interval $(t, t + hT_S)$, kjer je h horizont. Če ima sistem $m = 2$ regulirni veličini, mora optimizacija najti $m \cdot h$ optimalnih parametrov, kar hitro lahko postane računsko zahtevno in posledično problematično za sisteme s kratkim računskim korakom regulacijske zanke. Vseeno pa obstaja nekaj možnosti za zmanjšanje računskega časa. Ena možnost je, da v predikcijskem horizontu predpostavimo konstantne regulirne veličine. Če se akcija regulatorja v relativno kratkem času horizonta bistveno ne spremeni, lahko v intervalu horizonta predpostavimo konstantne regulirne veličine. To pomeni, da je potrebno optimizirati le m parametrov namesto $m \cdot h$. Druga možnost je zmanjšanje števila potrebnih iteracij pri optimizaciji za vsak računski korak regulacijske zanke. To lahko storimo z inicializacijo delcev okoli optimalne rešitve iz prejšnjega časovnega vzorca. Tako bi bilo potrebno manj

iteracij za konvergenco delcev.

V primeru 3.15 je podana možna rešitev za implementacijo MPC na PSO.

Primer 3.15

Razširite primer 3.14 na modelni prediktivni regulator s predikcijskim horizontom $h = 3$.

Rešitev

Predstavljena rešitev predvideva, da je akcija regulatorja (povratnozančni del) v intervalu predikcijskega horizonta konstantna $\boldsymbol{u}_{fb}(t + (i-1)T_S) = \boldsymbol{u}_{fb}$. Akcija predkrmiljenja ($\boldsymbol{u}_{ff}$) je pridobljena iz znane trajektorije, medtem ko se akcija povratne zanke izračuna z uporabo PSO. Iz trenutne lege robota je h-koračna predikcija pridobljena s pomočjo kinematičnega modela robota in akcij regulatorja $\boldsymbol{u}(t + (i-1)T_S) = \boldsymbol{u}_{fb} + \boldsymbol{u}_{ff}(t + (i-1)T_S)$ $(i = 1, \ldots, h)$ kot

$$\hat{\boldsymbol{q}}(t + iT_S) = f\left(\hat{\boldsymbol{q}}\left(t + (i-1)\,T_S\right), \boldsymbol{u}\left(t + (i-1)\,T_S\right)\right)$$

kjer je začetno stanje enako $\hat{\boldsymbol{q}}(t) = \boldsymbol{q}(t)$ in $i = 1, \ldots, h$. Akcijo regulatorja dobimo z optimizacijo cenilke znotraj horizonta

$$J(t + hT_S) = \sum_{i=1}^{h} \boldsymbol{e}(t + iT_S)^T \boldsymbol{Q}\boldsymbol{e}(t + iT_S) + \boldsymbol{u}_{fb}^T \boldsymbol{R}\boldsymbol{u}_{fb}$$

kjer je $\boldsymbol{e}(\cdot)$ sledilni pogrešek v lokalnih koordinatah. Optimalna akcija regulatorja se izračuna z uporabo PSO, kot je predstavljeno v Matlab kodi v programu 3.14. Rezultati simulacije so prikazani na slikah 3.34 in 3.35. Sledilni pogrešek je nekoliko manjši, pa tudi začetni prehodni pojav je boljši kot v primeru 3.14.

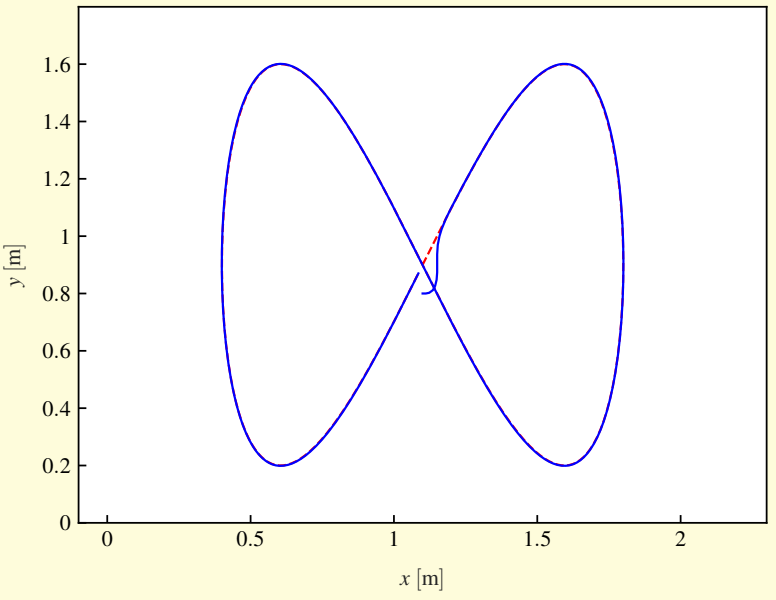
Program 3.14

./src/ctr/example_tracking_pso_mpc.m

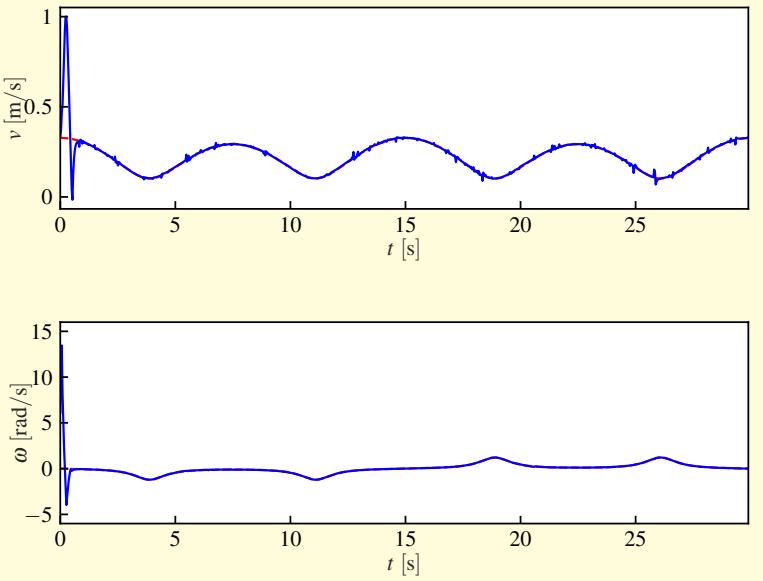
```
Ts = 0.033; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [1.1; 0.8; 0]; % Začetna lega

% Referenca
freq = 2*pi/30;
  xRef = 1.1 + 0.7*sin(freq*t);    yRef = 0.9 + 0.7*sin(2*freq*t);
 dxRef = freq*0.7*cos(freq*t);    dyRef = 2*freq*0.7*cos(2*freq*t);
ddxRef =-freq^2*0.7*sin(freq*t); ddyRef =-4*freq^2*0.7*sin(2*freq*t);
qRef = [xRef; yRef; atan2(dyRef, dxRef)]; % Referenčna trajektorija
vRef = sqrt(dxRef.^2+dyRef.^2);
wRef = (dxRef.*ddyRef-dyRef.*ddxRef)./(dxRef.^2+dyRef.^2);
uRef = [vRef; wRef]; % Referenčni vhodi

vMax = 1; wMax = 15; % Omejitve hitrosti

% Inicializacija roja delcev
iterations = 20; % Število iteracij
```

```
omega = 0.5*0.5; % Faktor vztrajnosti
c1 = 0.5*1; % Samozavedna konstanta
c2 = 0.5*1; % Socialna konstanta
N = 25; % Velikost roja delcev
swarm = zeros([2,N,4]);
uBest = [0; 0];

H = 3; % Dolžina predikcijskega horizonta

for k = 1:length(t)-H
    % Začetni položaji delcev
    swarm(:,:,1) = repmat(uBest, 1, N) + diag([0.1; 3])*randn(2,N);
    swarm(:,:,2) = 0; % Začetne hitrosti delcev
    swarm(1,:,4) = 1000; % Najboljša vrednost kriterijske funkcije

    for iter = 1:iterations % Iterativno iskanje optimalne rešitve s PSO
        % Vrednotenje parametrov delcev
        for i = 1:N
            % Izračun nove predvidene lege robota na podlagi parametrov
            % i-tega delca (vhodnih hitrosti) in primerjava predvidene
            % lege z referenčno lego.
            vwi = swarm(:,i,1);
            ui = vwi + uRef(:,k); % Regulacija in krmiljenje
            qk = q; % Trenutna lega robota
            % Predikcija lege robota na podlagi parametrov delcev (hitrosti)
            J = 0;
            for h = 1:H
                qk = qk + Ts*[cos(qk(3)), 0; sin(qk(3)), 0; 0, 1]*ui;
                qk(3) = wrapToPi(qk(3)); % Zapis kota v območju [-pi, pi]
                e = [cos(qk(3)), sin(qk(3)), 0; ...
                    -sin(qk(3)), cos(qk(3)), 0; ...
                     0,          0,          1]*(qRef(:,k+h)-qk); % Pogrešek
                e(3) = wrapToPi(e(3)); % Zapis kota v območju [-pi, pi]
                Qt = diag([4; 80; 0.1]); Rt = diag([1; 1]*0.0001);% Uteži
                J = J + e.'*Qt*e + vwi.'*Rt*vwi; % Kriterijska funkcija
            end
            if J<swarm(1,i,4) % Če je novi parameter boljši, posodobi:
                swarm(:,i,3) = swarm(:,i,1); % vrednosti parametrov (v in w)
                swarm(1,i,4) = J; % in najboljšo vrednost kriterijske funkcije.
            end
        end
        [~, gBest] = min(swarm(1,:,4)); % Parametri globalno najboljšega delca

        % Posodobitev parametrov s hitrostnimi vektorji
        a = omega*swarm(:,:,2) + ...
            c1*rand(2,N).*(swarm(:,:,3) - swarm(:,:,1)) + ...
            c2*rand(2,N).*(repmat(swarm(:,gBest,3), 1, N) - swarm(:,:,1));
        % Maksimalna sprememba parametrov, pospešek: aMax=3 ==> 3*Ts=0.1
        a(1,a(1,:)>0.1) = 0.1; a(1,a(1,:)<-0.1) = -0.1;
        % Maksimalna sprememba parametrov, kotni pospešek: aMax=60 ==> 60*Ts=2
        a(2,a(1,:)>2) = 2;     a(2,a(1,:)<-2) = -2;

        v = swarm(:,:,1) + a; % Posodobitev hitrosti
        % Omejitev hitrosti z ohranjanjem ukrivljenosti
        [m, ii] = max([v(1,:)/vMax; v(2,:)/wMax; ones(1,N)]);
        i = ii==1; v(1,i) = sign(v(1,i))*vMax;
                   v(2,i) = v(2,i)./m(i);
        i = ii==2; v(2,i) = sign(v(2,i))*wMax;
                   v(1,i) = v(1,i)./m(i);

        swarm(:,:,2) = a; % Posodobitev hitrosti delcev (pospeški)
        swarm(:,:,1) = v; % Posodobitev položajev delcev (hitrosti)
    end
```

```
82
83      % Vzmememo najboljši delec za izračun regulirnih signalov
84      uBest = swarm(:,gBest,1);
85      u = uBest + uRef(:,k); % Regulacija in krmiljenje
86
87      % Omejitve hitrosti
88      if abs(u(1))>vMax, u(1) = sign(u(1))*vMax; end
89      if abs(u(2))>wMax, u(2) = sign(u(2))*wMax; end
90
91      % Simulacija gibanja robota
92      dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
93      noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
94      q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
95      q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
96  end
```



Slika 3.34: Pridobljeni rezultati vodenja z uporabo MPC in PSO (referenca je označena s črtkano krivuljo)



Slika 3.35: Akcija regulatorja, izračunana z uporabo MPC in PSO (črtkana krivulja predstavlja le signal predkrmiljenja, brez regulirnega signala)

3.3.10 Vodenje mobilnega sistema s pristopom vodenja na osnovi slike

V tem poglavju bo predstavljeno *vodenje na osnovi slike* (VS, angl. *visual servoing*), ki se pogosto uporablja v mobilni robotiki. Glavni poudarek je na razvoju mobilnega robotskega sistema, ki lahko avtonomno opravi podano nalogo le na podlagi vizualnih informacij od kamere, nameščene na mobilnem sistemu.

Pri vodenju na osnovi slike se za določitev regulacijskega pogreška uporabljajo t. i. *značilke*. Značilke so neodvisne spremenljivke, ki opisujejo določen vizualni signal, torej lahko z njimi opišemo točke, črte, kroge, območje objekta, kote med črtami itd. Regulacijski pogrešek splošne regulacijske sheme pri vodenju na osnovi slike lahko zapišemo kot razliko med vektorjem želenih značilk $\boldsymbol{x}_{ref}(t)$ in vektorjem trenutnih značilk $\boldsymbol{x}(\boldsymbol{y}(t), \boldsymbol{\zeta})$ [27]

$$\boldsymbol{e}(t) = \boldsymbol{x}_{ref}(t) - \boldsymbol{x}(\boldsymbol{y}(t), \boldsymbol{\zeta}) \tag{3.56}$$

K-ti element vektorja značilk $\boldsymbol{x}(\boldsymbol{y}(t), \boldsymbol{\zeta}) \in \mathbb{R}^K$ v (3.56) je pridobljen iz meritev na podlagi slike $\boldsymbol{y}(t)$ (npr. pozicija, območje ali oblika vzorcev na sliki) in dodatnega časovno nespremenljvega znanja o sistemu $\boldsymbol{\zeta}$ (npr. notranji parametri

kamere, znane lastnosti opazovane scene, omejitve sistema). Eden od ključnih izzivov pri načrtovanju vodenja na osnovi slike je ustrezna opredelitev vektorja značilk, saj se ta vektor uporablja za določanje regulacijskega pogreška. Če je vektor značilk neustrezno določen, lahko obstaja več situacij, kjer je regulacijski pogrešek najmanjši — problem lokalnih minimumov. Nekatere značilke so primerne samo za določene vrste gibanja (npr. samo translacija) in so lahko popolnoma neprimerne za nekatere druge vrste gibanja (npr. translacija z rotacijo). Če značilke niso skrbno izbrane, lahko v nekaterih situacijah pride do neželenih, nepotrebnih ali celo nepričakovanih ukrepov vodenja — težava, znana kot *samovoljnost kamere* (angl. *camera retreat*). Tekom sledenja lahko nekatere značilke zapustijo vidno polje kamere, kar lahko onemogoči dokončanje naloge vodenja na osnovi slike, zato mora algoritem vodenja preprečiti tovrstne situacije. To lahko doseže z uporabo alternativnih značilk, če se nekatere neizogibno izgubijo, ali z oceno lokacije značilk, ki so začasno izven vidnega polja.

Glede na definicijo vektorja značilk $\boldsymbol{x}$ lahko vodenja na osnovi slike razvrstimo v tri glavne kategorije [28]: položajno vodenje na osnovi slike (PBVS, angl. *position-based visual servoing*), direktno vodenje na osnovi slike (IBVS, angl. *image-based visual servoing*) in hibridno vodenje na osnovi slike (HVS, angl. *hybrid visual servoing*). V primeru PBVS je regulacijski pogrešek opredeljen kot razlika med želeno in trenutno lego robota v tridimenzionalnem delovnem prostoru. V strukturi vodenja PBVS se kamera uporablja za oceno tridimenzionalnih položajev objektov in robota iz slike, ki je projekcija realnega okolja. Zato je mogoče tudi tukaj neposredno uporabiti vse do zdaj predstavljene načine vodenja. Običajno je mogoče s PBVS doseči optimalne premike, vendar ta pristop zahteva točno kalibracijo kamere, drugače ni možno odpraviti pravega pogreška vodenja. Po drugi strani je regulacijski pogrešek pri IBVS definiran neposredno v dvodimenzionalnem prostoru slike (npr. kot razlika med slikovnima točkama), zato ima točnost kalibracije kamere manjši vpliv na IBVS. Vendar pa IBVS običajno dosega manj optimalne trajektorije, kot jih je mogoče doseči s pristopom PBVS. IBVS je tudi bolj dovzeten za težave, ki jih povzroča samovoljnost kamere (primer kjer se kamera z rotacijo bliža cilju, ga preseže v normalni smeri in se nato vrača nazaj), če značilke niso pravilno izbrane. Pristopi IBVS so še posebej zanimivi, saj omogočajo opredelitev naloge neposredno na sliki in regulacijski pristop, znan kot *nauči-s-prikazom* (angl. *teach-by-showing*). HVS skuša združiti dobre lastnosti regulacijskih shem PBVS in IBVS. Nekatere regulacijske sheme preklapljajo med PBVS in IBVS v skladu s kriterijem preklopa, ki na podlagi trenutnih stanj sistema izbere najprimernejši način vodenja [29]. V nekaterih drugih HVS je regulacijski pogrešek sestavljen iz značilk dvodimenzionalnega slikovnega prostora kot tudi značilk tridimenzionalnega delovnega prostora [30, 31]. Še posebej zahtevno je vodenje na osnovi slike neholonomičnih sistemov [32–35]. Sheme vodenja na osnovi slike se včasih uporabljajo v kombinaciji z nekaterimi dodatnimi senzorji [36], ki lahko podajo dodatne podatke ali olajšajo obdelavo slike.

V [27, 37] so avtorji predstavili splošni pristop načrtovanja vodenja na osnovi

slike, ki ga je mogoče uporabiti ne glede na njegovo kategorijo. Vzemimo primer, ko je vektor referenčnih značilk v (3.56) časovno nespremenljiv ($\boldsymbol{x}_{ref}(t) = \boldsymbol{x}_{ref} =$ const.). Običajno se uporabi hitrostni regulator. V tem primeru lahko zapišemo vhodni vektor kot kombinacijo vektorja translatorne hitrosti $\boldsymbol{v}(t)$ in vektorja kotne hitrosti $\boldsymbol{\omega}(t)$ v kombiniranem vektorju $\boldsymbol{u}^T(t) = [\boldsymbol{v}^T(t),\ \boldsymbol{\omega}^T(t)] \in \mathbb{R}^M$. V splošnem sta tako vektor translatorne hitrosti kot vektor kotne hitrosti nekega objekta (v tridimenzionalnem prostoru) sestavljena iz treh hitrosti: $\boldsymbol{v}^T(t) = [v_x(t),\ v_y(t),\ v_z(t)]$ in $\boldsymbol{\omega}^T(t) = [\omega_x(t),\ \omega_y(t),\ \omega_z(t)]$. Vendar pa imata v kolesni mobilni robotiki ta dva vhodna vektorja običajno nekaj ničelnih elementov, saj se mobilni robot v normalnih voznih razmerah premika po ravnini in ne pričakujemo prevračanja ali dvigovanja. Razmerje med hitrostjo premika $\boldsymbol{u}^T(t)$ in hitrostjo spreminjanja značilk $\dot{\boldsymbol{s}}^T(t)$ lahko zapišemo kot

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{L}(t)\boldsymbol{u}(t) \tag{3.57}$$

kjer je matrika $\boldsymbol{L}(t) \in \mathbb{R}^K \times \mathbb{R}^M$ znana kot *interakcijska matrika*. Iz relacij (3.56) in (3.57) lahko določimo hitrost spreminjanja pogreška

$$\dot{\boldsymbol{e}}(t) = -\boldsymbol{L}(t)\boldsymbol{u}(t) \tag{3.58}$$

Regulirni signal $\boldsymbol{u}(t)$ mora minimizirati pogrešek $\boldsymbol{e}$. Če je zaželeno eksponentno padanje pogreška $\boldsymbol{e}$ v obliki $\dot{\boldsymbol{e}}(t) = -g\boldsymbol{e}(t)$, $g > 0$, lahko izpeljemo sledeči regulacijski zakon

$$\boldsymbol{u}(t) = g\boldsymbol{L}^{\dagger}(t)\boldsymbol{e}(t) \tag{3.59}$$

kjer je $\boldsymbol{L}^{\dagger}(t)$ Moore-Penroseov psevdoinverz interakcijske matrike $\boldsymbol{L}(t)$. Če ima matrika $\boldsymbol{L}(t)$ poln rang, je njen psevdoinverz enak $\boldsymbol{L}^{\dagger}(t) = (\boldsymbol{L}^T(t)\boldsymbol{L}(t))^{-1}\boldsymbol{L}^T(t)$. V primeru da je interakcijska matrika kvadratna ($K = M$) in ni singularna ($\det(\boldsymbol{L}(t)) \neq 0$), se izračun psevdoinverza v (3.59) poenostavi v običajno inverzno matriko $\boldsymbol{L}^{-1}(t)$.

V praksi je znana le približna vrednost prave interakcijske matrike. Zato lahko v regulacijskem zakonu (3.59) uporabimo samo oceno interakcijske matrike $\widehat{\boldsymbol{L}}(t)$ ali njenega psevdoinverza $\widehat{\boldsymbol{L}^{\dagger}}(t)$

$$\boldsymbol{u}(t) = g\widehat{\boldsymbol{L}^{\dagger}}(t)\boldsymbol{e}(t) \tag{3.60}$$

Oceno psevdoinverzne interakcijske matrike $\widehat{\boldsymbol{L}^{\dagger}}(t)$ lahko določimo z linearizacijo sistema okoli njegovega trenutnega stanja ($\widehat{\boldsymbol{L}^{\dagger}}(t) = \widehat{\boldsymbol{L}}_t^{\dagger}(t)$) ali okoli njegovega želenega (referenčnega) stanja ($\widehat{\boldsymbol{L}^{\dagger}}(t) = \widehat{\boldsymbol{L}}_{ref}^{\dagger}(t)$). Včasih lahko uporabimo tudi kombinacijo v obliki $\widehat{\boldsymbol{L}^{\dagger}}(t) = \frac{1}{2}(\widehat{\boldsymbol{L}}_{ref}(t) + \widehat{\boldsymbol{L}}_t(t))^{\dagger}$ [37].

Če vstavimo (3.60) v (3.58), dobimo diferencialno enačbo zaprtozančnega sistema

$$\dot{\boldsymbol{e}}(t) = -g\boldsymbol{L}(t)\widehat{\boldsymbol{L}^{\dagger}}(t)\boldsymbol{e}(t) \tag{3.61}$$

Funkcijo Ljapunova $V(t) = \frac{1}{2}\boldsymbol{e}^T(t)\boldsymbol{e}(t)$ lahko uporabimo za preverjanje stabilnosti zaprtozančnega sistema (3.61). Odvod funkcije Ljapunova je

$$\dot{V}(t) = -g\boldsymbol{e}^T(t)\boldsymbol{L}(t)\widehat{\boldsymbol{L}^{\dagger}}(t)\boldsymbol{e}(t)$$

Zadosten pogoj za globalno stabilnost sistema (3.61) je izpolnjen, če je matrika $\boldsymbol{L}(t)\widehat{\boldsymbol{L}^{\dagger}}(t)$ pozitivno definitna. V primeru, da je število opazovanih značilk K enako številu regulirnih veličin M, torej $K = M$, ter imata matriki $\boldsymbol{L}(t)$ in $\widehat{\boldsymbol{L}^{\dagger}}(t)$ poln rang, je zaprtozančni sistem (3.61) stabilen, če le ocena matrike $\widehat{\boldsymbol{L}^{\dagger}}(t)$ ni pregroba [27]. Vendar je potrebno opozoriti, da v primeru IBVS ni preprosto izbrati ustreznih značilk v slikovnem prostoru, ki odpravijo problem vodenja v delovnem prostoru.

Primer 3.16

Kamera C je nameščena na kolesnem mobilnem robotu R z diferencialnim pogonom na $\boldsymbol{t}_C^R = [0,\ 0,\ 0{,}5]^T$ z orientacijo $\boldsymbol{R}_C^R = \boldsymbol{R}_x(90°)\boldsymbol{R}_y(-90°)\boldsymbol{R}_x(45°)$. Parametri kamere so (model kamere je opisan v poglavju 5.2.4): $\alpha_x f = \alpha_y f = 300$, $\gamma = 0$ (brez striga), središče slike pa je v središču slike z dimenzijo 1024 krat 768. Načrtajte IBVS, ki vodi mobilnega robota iz začetne lege $[x(0),\ y(0),\ \varphi(0)] = [1\,\text{m},\ 0\,\text{m},\ 100°]$ do ciljne pozicije $x_{ref} = 4\,\text{m}$ in $y_{ref} = 4\,\text{m}$. Zavoljo poenostavitve predpostavimo, da je na sliki vidno središče vrtenja mobilnega robota.

Rešitev

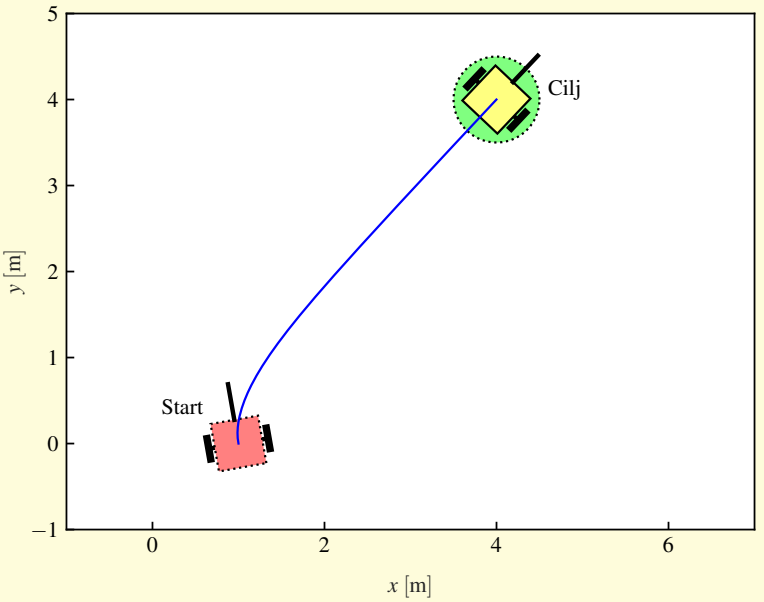
Kamera opazuje prizor pred mobilnim robotom. Ker je kamera nameščena brez možnosti zasuka okoli središča optične osi, uporabimo razklopljeno vodenje (angl. *decoupled control*) neposredno na podlagi slikovnega pogreška med opazovano ciljno pozicijo in sliko središča rotacije robota. Možna implementacija rešitve v Matlabu je prikazana v programu 3.15. Dobljena pot mobilnega robota je prikazana na sliki 3.36, pot opazovanega cilja na sliki pa je prikazana na 3.37. Regulirni signali so prikazani na sliki 3.38. Rezultati potrjujejo uporabnost IBVS. Robot doseže ciljno lego, saj je opazovana značilka (cilj) ves čas vidna (slika 3.37).

Program 3.15

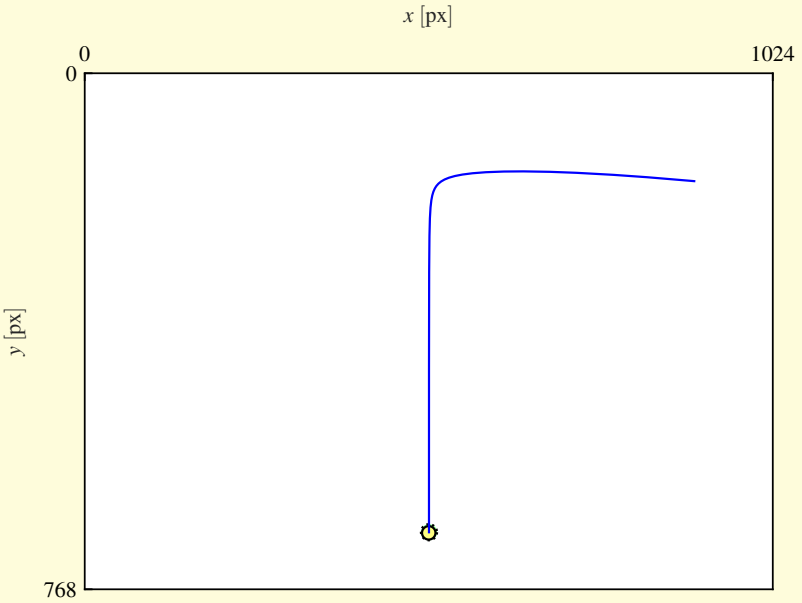
./src/ctr/example_diff_vs_point.m

```
Ts = 0.03; % Računski korak
t = 0:Ts:15; % Čas simulacije
r = 0.5; % Razdalja vmesne točke od cilja
dTol = 1; % Tolerančna razdalja od vmesne točke za preklop
qRef = [4; 4; 0]; % Referenčna lega
q = [1; 0; 100/180*pi]; % Začetna lega

% Kamera
alphaF = 300; % % alpha*f, v px/m
s = [1024; 768]; % Dimenzije zaslona, v px
c = s/2; % Optično središče, v px
S = [alphaF, 0, c(1); 0, alphaF, c(2); 0, 0, 1]; % Notranji model kamere
RL2C = rotX(pi/2)*rotY(-pi/2)*rotX(pi/4); tL2C=[0;0;0.5]; % Lega kamere
% Simulacija kamere
p0P = S*RL2C.'*([0; 0; 0]-tL2C); p0P = p0P/p0P(3);
RW2L = rotZ(-q(3)); tW2L = [q(1:2); 0];
```

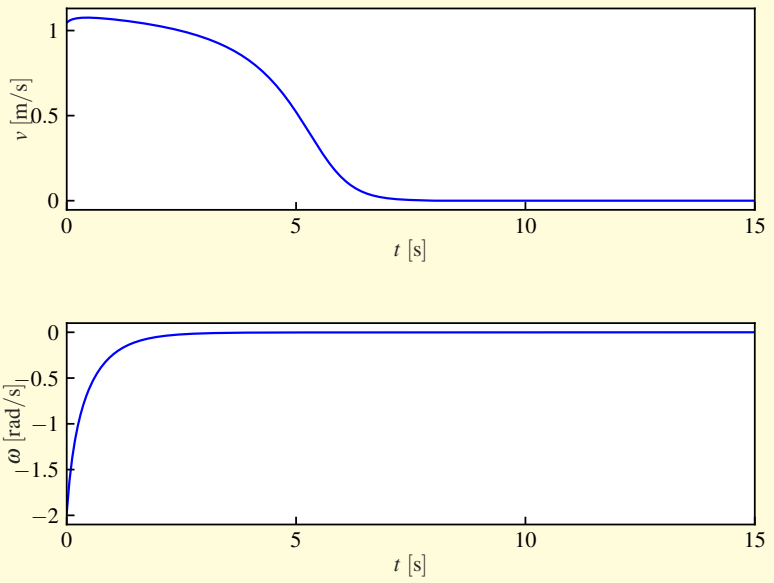
```
pP = S*RL2C.'*(RW2L.'*([qRef(1:2); 0]-tW2L)-tL2C); pP = pP/pP(3);

u = [0; 0];
for k = 1:length(t)
    if pP(1)<0 || pP(2)<0 || pP(1)>s(1) || pP(2)>s(2) % Nevidna značilka
        u = [0; 0]; % Sledena značilka je izgubljena
    else
        D = sqrt(sum((pP(1:2)-p0P(1:2)).^2));
        if D<dTol % Ustavitev v bližini cilja
            u = [0; 0];
        else
            u = [0, 0.002; 0.005, 0]*(p0P(1:2)-pP(1:2));
        end
    end

    % Simulacija gibanja robota
    dq = [u(1)*cos(q(3)); u(1)*sin(q(3)); u(2)];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]

    % Simulacija kamere
    RW2L = rotZ(-q(3)); tW2L = [q(1:2); 0];
    pP = S*RL2C.'*(RW2L.'*([qRef(1:2); 0]-tW2L)-tL2C); pP = pP/pP(3);
end
```



Slika 3.36: Pot pridobljena z IBVS iz primera 3.16



Slika 3.37: Pot opazovane značilke (cilja) v slikovnem prostoru iz primera 3.16



Slika 3.38: Regulirna signala IBVS iz primera 3.16

3.4 Ocena optimalnega profila hitrosti za znano pot

Kolesni mobilni roboti se morajo pogosto voziti po obstoječi vnaprej določeni poti (npr. po cesti ali koridorju), ki je prostorsko določena z nekim parametrom u kot $x_p(u)$, $y_p(u)$, $u \in [u_{SP}, u_{EP}]$. Za vožnjo robota po tej poti je potrebno določiti želeni hitrostni profil, če mora robot v najkrajšem času priti od neke začetne točke (SP) do neke končne točke (EP) in pri tem upoštevati zmožnosti robota ipd. Za vožnjo robota po takšni poti mora biti njegova pozicija odvisna od časa $x(t)$, $y(t)$ z določenim hitrostnim profilom $v(t)$, $\omega(t)$, kot bo prikazano v nadaljevanju.

Predpostavimo poseben primer $u = t$, kjer je referenčna pot v bistvu trajektorija z implicitno podanim hitrostnim profilom. Robot mora voziti po trajektoriji z referenčnima hitrostma $v(t) = v_{ref}(t)$ in $\omega(t) = \omega_{ref}(t)$, izračunani iz referenčne trajektorije z uporabo relacij (3.17) in (3.18).

Pri načrtovanju želenega hitrostnega profila za prostorsko podano pot, je potrebno najti razpored $u = u(t)$. Načrtovani hitrostni profil mora biti skladen z omejitvami robota, kot sta največja hitrost in pospešek, ki ju lahko proizvajajo motorji ali pa ki zagotavljata varno vožnjo brez vzdolžnega in bočnega drsenja koles. Referenčne hitrosti so podobno kot v (3.17) in (3.18), izražene kot

$$v(t) = \sqrt{x_p'\left(u\left(t\right)\right)^2 + y_p'\left(u\left(t\right)\right)^2}\dot{u}\left(t\right) = v_p\left(u\right)\dot{u}\left(t\right) \tag{3.62}$$

$$\omega(t) = \frac{x_p^{'}(u(t))y_p^{''}(u(t)) - y_p^{'}(u(t))x_p^{''}(u(t))}{x_p'(u(t))^2 + y_p'(u(t))^2}\dot{u}(t) = \omega_p\left(u\right)\dot{u}\left(t\right)$$

in ukrivljenost je

$$\kappa(t) = \frac{x_p^{'}(u(t))y_p^{''}(u(t)) - y_p^{'}(u(t))x_p^{''}(u(t))}{\left(x_p'(u(t))^2 + y_p'(u(t))^2\right)^{3/2}} = \kappa_p\left(u\right)$$

kjer črtice označujejo odvode po u, pike pa odvode po t. Časovni odvodi poti upoštevajo razpored $u(t)$ na način $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\mathrm{d}x_p}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}t} = x_p^{'}\dot{u}\left(t\right)$, $\frac{\mathrm{d}y(t)}{\mathrm{d}t} = \frac{\mathrm{d}y_p}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}t} = y_p^{'}\dot{u}\left(t\right)$.

Glavna ideja načrtovanja hitrostnega profila je povzeta iz [38], ki se določi glede na omejitev idealnega kotaljenja. To pomeni, da so regulirne hitrosti enake dejanskim hitrostim robota (brez drsenja koles), kar dosežemo z omejitvijo celotnega dovoljenega pospeška

$$a = \sqrt{a_t^2 + a_r^2}$$

kjer sta

$$a_t = \frac{\mathrm{d}v}{\mathrm{d}t} \qquad a_r = v\omega = v^2\kappa \tag{3.63}$$

tangencialni in radialni pospešek. Največja vrednost pospeška, ki preprečuje drsenje, je določena s silo trenja F_{fric}

$$a_{MAX} = \frac{F_{fric}}{m} = \frac{mgc_{fric}}{m} = gc_{fric}$$

kjer je m masa vozila, g gravitacijski pospešek in c_{fric} koeficient trenja. Največji tangencialni pospešek a_{MAXt} in radialni pospešek a_{MAXr} se zaradi konstrukcije vozila ponavadi razlikujeta in ju je mogoče eksperimentalno oceniti. Pomembno je, da se skupni pospešek nahaja znotraj elipse

$$\frac{a_t^2}{a_{MAXt}^2} + \frac{a_r^2}{a_{MAXr}^2} \leq 1 \tag{3.64}$$

ali pa, v primeru časovno optimalnega načrtovanja, na njenem robu. V zavojih na poti mora robot zaradi večjih radialnih pospeškov voziti počasneje. Zato za oceno razporeda $u(t)$ najprej na poti označimo *prelomne točke* (TP, angl. *turning points*), kjer je absolutna vrednost ukrivljenosti lokalno največja. Parameter u je v intervalu $[u_{SP}, u_{EP}]$. Pozicije TP so označene z $u = u_{TPi}$, kjer je $i = 1, \ldots, n_{TP}$ in n_{TP} je število TP. V TP je translatorna hitrost lokalno najmanjša, tangencialni pospešek naj bi bil 0, radialni pospešek pa je največji. Tangencialno hitrost v TP lahko glede na (3.63) izračunamo kot

$$v_p(u_{TPi}) = \sqrt{\frac{a_{MAXr}}{|\kappa(u_{TPi})|}} \tag{3.65}$$

Pred in po TP se lahko robot premika hitreje, ker je ukrivljenost manjša kot v TP. Zato mora robot pred vsako TP upočasniti ($u < u_{TPi}$) in po njej pospešiti ($u > u_{TPi}$) v skladu z omejitvijo pospeška (3.64).

Iz (3.62) sledi, da sta $v(t)$ in $v_p(u)$ v vsaki fiksni točki na poti sorazmerni s časovno odvisnim proporcionalnim faktorjem $\dot{u}(t)$. Najkrajši izvedljiv hitrostni profil je torej določen z odvodom razporeda $\dot{u}(t)$, kjer minimiziramo največje hitrostne profile, ki izpolnjujejo omejitve pospeška, kot je opisano v nadaljevanju. Radialni in tangencialni pospešek sta izražena iz (3.63) z upoštevanjem (3.62) kot

$$a_r(t) = \left(x_p^{'}(u)^2 + y_p^{'}(u)^2\right) \kappa_p(u)\, \dot{u}^2(t) = v_p^2(u)\, \kappa_p(u)\, \dot{u}^2(t) \tag{3.66}$$

$$\begin{aligned} a_t(t) &= \frac{x_p^{'}(u)x_p^{''}(u) + y_p^{'}(u)y_p^{''}(u)}{\sqrt{x_p^{'}(u)^2 + y_p^{'}(u)^2}}\dot{u}^2(t) + \sqrt{x_p^{'}(u)^2 + y_p^{'}(u)^2}\,\ddot{u}^2(t) \\ &= \frac{\mathrm{d}v_p(u)}{\mathrm{d}u}\dot{u}^2(t) + v_p(u)\,\ddot{u}(t) \end{aligned} \tag{3.67}$$

kjer je zaradi krajšega zapisa izpuščena odvisnost od časa za $u(t)$. Z mejnim primerom (3.64) in (3.66) iz (3.67) dobimo optimalno diferencialno enačbo razporeda

$$\ddot{u} = \pm a_{MAXt}\sqrt{\frac{1}{x_p^{'2} + y_p^{'2}} - \frac{(x_p^{'2} + y_p^{'2})\kappa_p^2\dot{u}^4}{a_{MAXr}^2}} - \frac{x_p^{'}x_p^{''} + y_p^{'}y_p^{''}}{x_p^{'2} + y_p^{'2}}\dot{u}^2 \tag{3.68}$$

Rešitev diferencialne enačbe najdemo z eksplicitno simulacijo z integracijo iz TP naprej in nazaj po času. Pri pospeševanju uporabimo pozitiven predznak, pri upočasnjevanju pa negativen predznak. Začetna pogoja $u(t)$ in $\dot{u}$ sta določena s položajem vseh TP u_{TPi} in z (3.66), ki pove, da je največja dovoljena vrednost radialnega pospeška v TP enaka

$$\dot{u}|_{TPi} = \sqrt{\frac{a_{MAXr}}{\left(x_p'\left(u_{TPi}\right)^2 + y_p'\left(u_{TPi}\right)^2\right)\kappa_p\left(u_{TPi}\right)}} \tag{3.69}$$

Prikazan je le pozitivni začetni pogoj (3.69), ker mora biti $u(t)$ strogo naraščajoča funkcija. Diferencialne enačbe rešujemo, dokler ne najdemo kršitve omejitve pospeška ali pa u zapusti interval $[u_{SP}, u_{EP}]$.

Kršitev se običajno pojavi pri (pospešenem) premikanju iz trenutne TPi proti naslednji TP ($i-1$ ali $i+1$) in vrednost translatorne hitrosti močno preseže največjo dovoljeno vrednost, ki jo določa trajektorija. Rešitev diferencialne enačbe (3.68) je tako sestavljena iz segmentov $\dot{u}$ okoli vsake TP

$$\dot{u}_l = \dot{u}_l(u)\,; \qquad u \in [\underline{u}_l,\ \overline{u}_l]\,, \quad l = 1, \ldots, n_{TP} \tag{3.70}$$

kjer je $\dot{u}_l = \dot{u}_l\left(u\left(t\right)\right)$ odvod razporeda, odvisen od u in $\underline{u}_l$, $\overline{u}_l$ pa so meje l-tega segmenta. Tu so segmenti v (3.70) podani kot funkcije od u, čeprav je simulacija funkcije (3.68) izvedena v času, ker časovni zamik (čas, potreben za prihod v TP) ni znan. Kar je znano na tej točki, je relativni časovni interval, ki ustreza rešitvi vsakega segmenta $\dot{u}_l$.

Rešitev časovno optimalnega hitrostnega profila, pridobljena s pospeševanjem na meji zdrsa (maksimalno pospeševanje) celotne trajektorije, je možna, če unija intervalov pokriva celoten interval zanimanja $[u_{SP}, u_{EP}] \subseteq \bigcup_{l=1}^{n_TP}[\underline{u}_l, \overline{u}_l]$. Dejanski $\dot{u}$ najdemo z minimizacijo posameznih maksimalnih profilov v okolici TP-jev

$$\dot{u} = \min_{1 \le l \le n_{TP}} \dot{u}_l(u) \tag{3.71}$$

Absolutni čas, ki ustreza razporedu $u(t)$, dobimo iz pretvorbe $\dot{u}\left(u\left(t\right)\right) = \frac{\mathrm{d}u}{\mathrm{d}t}$, da dobimo $dt = \dot{u}^{-1}(u)du$ in izvedemo integracijo

$$t = \int_{u_{SP}}^{u_{EP}} \frac{\mathrm{d}u}{\dot{u}(u)} = t(u)$$

kjer $\dot{u}(u)$ ne sme postati 0. V našem primeru je iskana časovno optimalna rešitev, zato je hitrost (tudi $\dot{u}(u)$) vedno večja od 0. Izraz $\dot{u}(u) = 0$ bi namreč pomenil, da je hitrost enaka nič, ko bi se sistem ustavil – to pa ne more voditi do časovno optimalne rešitve.

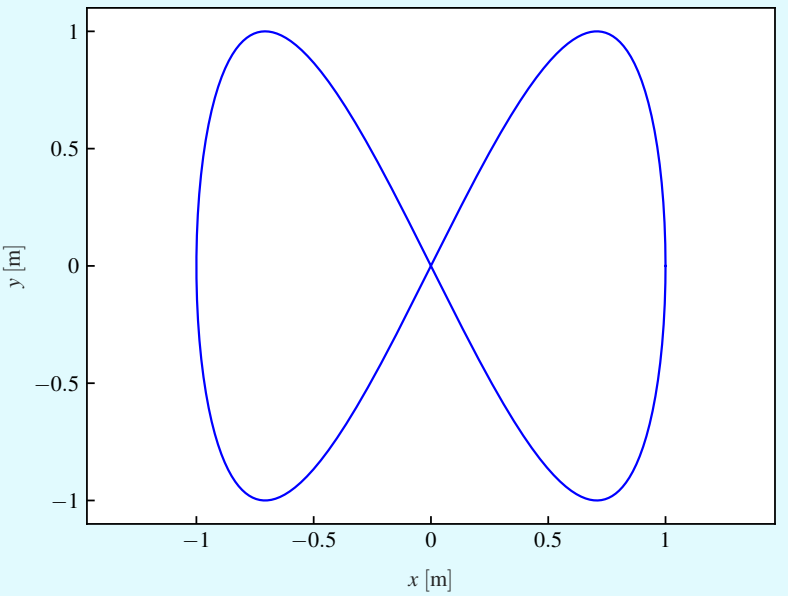
Začetne (v SP) in končne (v EP) hitrosti se lahko obravnavajo na naslednji način. SP in EP obravnavamo kot običajne TP, kjer so začetne hitrosti v_{SP}, $v_p(u_{SP})$, v_{EP}, $v_p(u_{EP})$ znane, zato lahko izračunamo začetne pogoje za $\dot{u}_{SP} = \frac{v_{SP}}{v_p(u_{SP})}$, $\dot{u}_{EP} = \frac{v_{EP}}{v_p(u_{EP})}$. Če so ti začetni pogoji za SP ali EP večji od rešitve za $\dot{u}$ (upoštevajoč vse TP), rešitev ne obstaja, ker bi bilo nemogoče priti skozi prvo

TP, tudi če robot maksimalno (popolnoma) zavira. Če rešitev za dane hitrosti v SP in EP obstaja, vključimo tudi segmente $\dot{u}$, izračunane za SP in EP v (3.70) in (3.71).

Iz izračunanega razporeda $u(t)$, $\dot{u}(t)$, je referenčna trajektorija podana kot $x(t) = x_p\left(u\left(t\right)\right)$ in $y(t) = y_p\left(u\left(t\right)\right)$ ter referenčna hitrost z (3.17) in (3.18).

Primer 3.17

Izračunajte optimalni hitrostni profil, ki bo omogočil najkrajši čas potovanja po poti $x_p(u) = \cos(u)$, $y_p(u) = \sin(2u)$ (na sliki 3.39), kjer je $u \in [0, 2\pi]$, ter upošteval največji tangencialni $a_{MAXt} = 2\,\mathrm{m/s^2}$ in radialni pospešek $a_{MAXr} = 4\,\mathrm{m/s^2}$.



Slika 3.39: Pot

.

Izračunajte razpored $u(t)$ in hitrostna profila $v(u)$, $v(t)$.

Rešitev

Za izračun optimalnega razporeda $u(t)$ uporabimo algoritem, predstavljen v tem poglavju. Najprej moramo izračunati vse TP z začetnimi pogoji, ki so podani v (3.65) in (3.69), nato pa simulirati rešitve za vsako TP z uporabo (3.68). Končno minimiziramo časovni odvod razporeda glede na (3.71).

Možna izvedba rešitve je podana v programu 3.16 (zanemarite dele kode za omejevanje hitrosti, ki bi jih lahko ovrednotili, če bi bila spremenljivka `velCnstr` nastavljena na `true`). Optimalna določitev razporeda za primer 3.17 je prikazana na slikah od 3.40 do 3.42. Iz slike 3.40 je razvidno, da je rešitev za vsako TP integrirana vse do točke, kjer so kršene omejitve pospeška, kar je označeno s tanko linijo. Končna rešitev je označena z odebeljeno krivuljo.

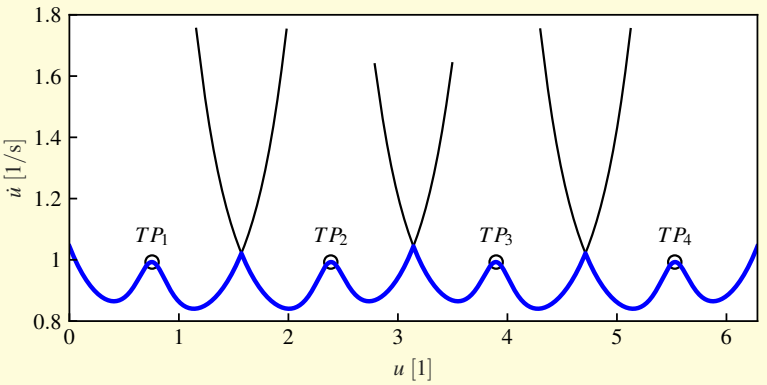
Program 3.16

./src/ctr/example_velocity_profile_planning.m

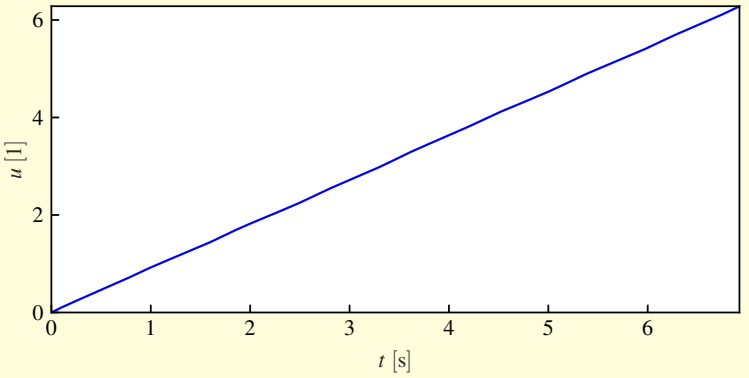
```
% Definicija trajektorije v obliki funkcij
  x = @(u) cos(u);    y = @(u) sin(2*u);    % Pot
 dx = @(u) -sin(u);  dy = @(u) 2*cos(2*u);  % Prvi odvod
ddx = @(u) -cos(u); ddy = @(u) -4*sin(2*u); % Drugi odvod
v = @(u) sqrt(dx(u).^2 + dy(u).^2); % Tangencialna hitrost
w = @(u) (dx(u).*ddy(u)-dy(u).*ddx(u))./(dx(u).^2+dy(u).^2); % Kotna hitrost
kappa = @(u) w(u)./v(u); % Ukrivljenost
dv = @(u) (dx(u).*ddx(u) + dy(u).*ddy(u))./v(u);

u = 0:0.001:2*pi; % Čas
arMax = 4; atMax = 2; % Omejitvi pospeška
vSP = 0.2; vEP = 0.1; % Začetna in končna hitrost
uSP = u(1); uEP = u(end); % Začetni in končni položaj
uTP = []; % Točke zavojev
for i = 2:length(u)-1 % Določitev točk zavojev
    if all(abs(kappa(u(i))) > abs(kappa(u([i-1, i+1]))))
        uTP = [uTP, u(i)];
    end
end
up0 = sqrt(arMax./abs(v(uTP).*w(uTP))) ; % Odvodi v točkah zavojev

velCnstr = false; % Omogoči omejitev hitrosti
if velCnstr
    vMax = 1.5; %Omejitev hitrosti
    for i = 1:length(uTP) % Prilagodi uTP glede na hitrostno omejitev
       vvu = v(uTP(i)); vvt = vvu*up0(i);
       if abs(vvt) > vMax, up0(i) = abs(vMax/vvu); end
    end
    % Dodaj zahtevo za začetno in končno hitrost
    uTP  = [uSP, uTP, uEP]; up0 = [vSP/v(uSP), up0, vEP/v(uEP)];
end

Ts = 0.001; % Računski korak
N = length(uTP); ts = cell(1,N); us = cell(1,N); ups = cell(1,N);
for i = 1:N % Zanka čez vse točke zavojev
    uB = uTP(i); upB = up0(i); tB = 0;
    uF = uTP(i); upF = up0(i); tF = 0;
    uBs =[]; upBs = []; tBs = []; uFs = []; upFs =[]; tFs = []; % Spomin
    goB = true; goF = true;

    while goB || goF
        % Integracija nazaj od točke zavoja
        if uB > uSP && goB
            dxT = dx(uB);   dyT = dy(uB);
            ddxT = ddx(uB); ddyT = ddy(uB);
            vT = v(uB)*upB; wT = w(uB)*upB; kappaT = kappa(uB);
            arT = vT*wT; atT = atMax*sqrt(1 - (arT/arMax)^2);

            if velCnstr && abs(vT) > vMax
```

```
50                  upB = vMax/v(uB); upp = -dv(uB)*upB^2/v(uB);
51              elseif abs(arT)-arMax > 0.001
52                  arT = arMax; atT = 0; upp = 0; goB = false;
53              else
54                  atT = -real(atT);
55                  upp = real(-atMax*sqrt(1/(dxT^2 + dyT^2) - ...
56                      (dxT^2 + dyT^2)*kappaT^2*upB^4/arMax^2) - ...
57                      (dxT*ddxT + dyT*ddyT)/(dxT^2 + dyT^2) * upB^2);
58              end
59
60              uBs = [uBs; uB]; upBs = [upBs; upB]; tBs = [tBs; tB]; % Shranjevanje
61              tB = tB + Ts;
62              uB = uB - upB*Ts; % Eulerjeva integracija
63              upB = upB - upp*Ts; % Eulerjeva integracija
64          else
65              goB = false;
66          end
67
68          % Integracija naprej of točke zavoja
69          if uF < uEP && goF
70              dxT = dx(uF);   dyT = dy(uF);
71              ddxT = ddx(uF); ddyT = ddy(uF);
72              vT = v(uF)*upF; wT = w(uF)*upF; kappaT = kappa(uF);
73              arT = vT*wT; atT = atMax*sqrt(1 - (arT/arMax)^2);
74
75              if velCnstr && abs(vT) > vMax
76                  upF = vMax/v(uF); upp = -dv(uF)*upF^2/v(uF);
77              elseif abs(arT)-arMax > 0.001
78                  arT = arMax; atT = 0; upp = 0; goF = false;
79              else
80                  atT = real(atT);
81                  upp = real(+atMax*sqrt(1/(dxT^2 + dyT^2) - ...
82                      (dxT^2 + dyT^2)*kappaT^2*upF^4/arMax^2) - ...
83                      (dxT*ddxT + dyT*ddyT)/(dxT^2 + dyT^2) * upF^2);
84              end
85
86              uFs = [uFs; uF]; upFs = [upFs; upF]; tFs = [tFs; tF]; % Shranjevanje
87              tF = tF + Ts;
88              uF = uF + upF*Ts; % Eulerjeva integracija
89              upF = upF + upp*Ts; % Eulerjeva integracija
90          else
91              goF = false;
92          end
93      end
94
95      ts{i} = [tBs; tB+tFs(2:end)];
96      us{i} = [flipud(uBs); uFs(2:end)];
97      ups{i} = [flipud(upBs); upFs(2:end)];
98  end
99
100 % Iskanje minimuma med vsemi profili
101 usOrig = us;
102 for i = 1:N-1
103     d = ups{i+1} - interp1(us{i}, ups{i}, us{i+1});
104     j = find(d(1:end-1).*d(2:end)<0, 1); % ups{i} je približno enak ups{i+1}
105     % Iskanje bolj natančnega u-ja, kjer sta profila ups{i} in ups{i+1} enaka
106     uj = us{i+1}(j) + (us{i+1}(j+1)-us{i+1}(j))/(d(j+1)-d(j))*(0-d(j));
107     rob = interp1(us{i}, ups{i}, uj);
108
109     keep = us{i} < uj;
110     us{i} = [us{i}(keep); uj];      ups{i} = [ups{i}(keep); rob];
111     keep = us{i+1} > uj;
112     us{i+1} = [uj; us{i+1}(keep)]; ups{i+1} = [rob; ups{i+1}(keep)];
```

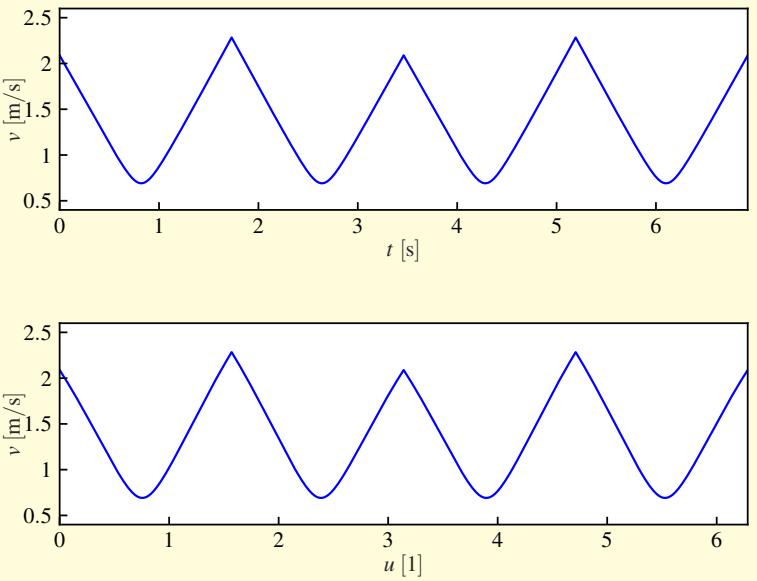
```
113 end
114
115 % Konstrukcija končne rešitve
116 tt = interp1(usOrig{1}, ts{1}, us{1}); uu = us{1}; uup = ups{1};
117 for i = 2:N
118     ti = interp1(usOrig{i},ts{i},us{i});
119     tt = [tt; ti + tt(end) - ti(1)];
120     uu = [uu; us{i} + uu(end) - us{i}(1)];
121     uup = [uup; ups{i}];
122 end
123 vv = v(uu).*uup;
```



Slika 3.40: Optimalno določen razpored, ki poišče najmanjši profil $\dot{u}$ vseh prelomnih točk (TP)



Slika 3.41: Optimalni razpored $u(t)$, ki je nelinearna funkcija, čeprav je v tem primeru videti skoraj linearna



Slika 3.42: Optimalni hitrosti $v(u)$ in $v(t)$

Primer 3.18

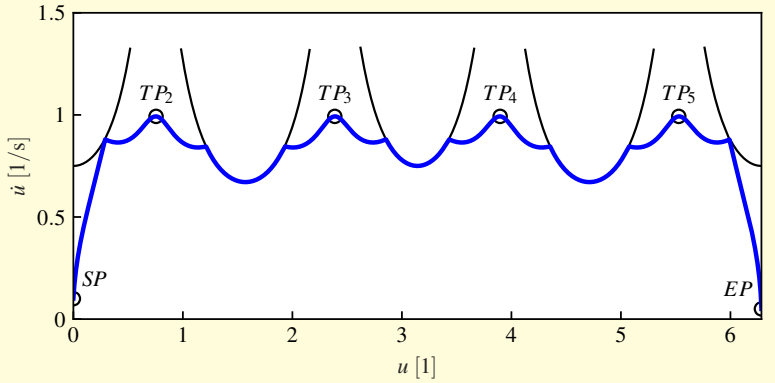
Razširite primer 3.17, da vključuje tudi zahteve za začetno in končno hitrost: $v_{SP} = 0{,}2\,\mathrm{m/s}$ in $v_{EP} = 0{,}1\,\mathrm{m/s}$. Poleg tega upoštevajte, da je največja hitrost omejena na $v_{MAX} = 1{,}5\,\mathrm{m/s}$.

Izračunajte razpored $u(t)$ in profile hitrosti $v(u)$, $v(t)$.

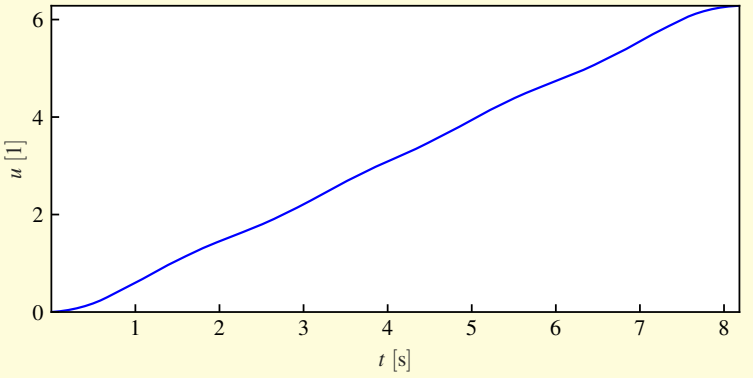
Rešitev

Kodo iz primera 3.17 je mogoče spremeniti tako, da vsebuje dodatne zahteve. Zahteve za začetno in končno hitrost se obravnavajo podobno kot v drugih TP. SP in EP sta obravnavani kot novi TP, katerih začetni pogoji so $u_{SP} = 0$, $u_{EP} = 2\pi$, $\dot{u}_{SP} = \frac{v_{SP}}{v_p(u_{SP})}$ in $\dot{u}_{EP} = \frac{v_{EP}}{v_p(u_{EP})}$.

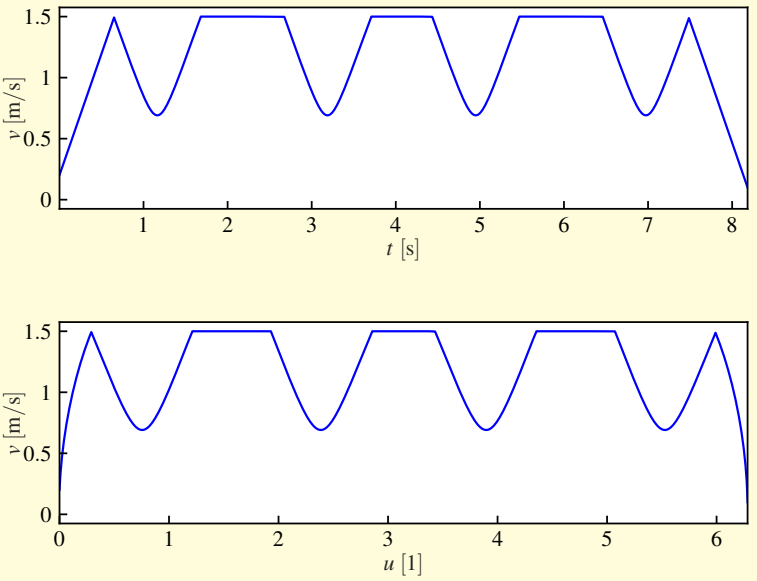
Omejitve hitrosti se upoštevajo, če je spremenljivka `velCnstr` v programu 3.16 nastavljena na `true`. Optimalna določitev razporeda za primer 3.18 je prikazana na slikah 3.43 – 3.45.



Slika 3.43: Optimalno določen razpored, ki poišče najmanjši $\dot{u}$ vseh prelomnih točk (TP)



Slika 3.44: Optimalni razpored $u(t)$, ki je nelinearna funkcija, čeprav je v tem primeru videti skoraj linearna



Slika 3.45: Optimalni hitrosti $v(u)$ in $v(t)$

Literatura

[1] R. W. Brockett. Asymptotic stability and feedback stabilization. V R. S. Millman in H. J. Sussmann (Uredniki), *Differential Geometric Control Theory*, str. 181–191. Birkhuser, Boston, MA, 1983.

[2] M. Bowling in M. Veloso. Motion control in dynamic multi-robot environments. V *IEEE International Symposium on, CIRA '99*, str. 168–173. IEEE, Monterey, CA, 1999.

[3] L. E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, zv. 79, št. 3, str. 497–516, 1957.

[4] J. A. Reeds in L. A. Shepp. Optimal paths for a car that goes both forward and backward. *Pacific Journal of Mathematics*, zv. 145, št. 2, str. 367–393, 1990.

[5] C. C. de Wit in O. J. Sordalen. Exponential stabilization of mobile robots with nonholonomic constraints. *Proceedings of the 30th IEEE Conference on Decision and Control*, zv. 37, št. 11, str. 1791–1797, 1991.

[6] A. De Luca, G. Oriolo in M. Vendittelli. Control of Wheeled Mobile Robots: An Experimental Overview. V S. Nicosia, B. Siciliano, A. Bicchi in P. Valigi (Uredniki), *Ramsete*, *Lecture Notes in Control and Information Sciences*, zv. 270, str. 181–226. Springer Berlin Heidelberg, 2001.

[7] A. Zdešar. *Simulacijsko okolje za analizo in sintezo algoritmov vodenja na osnovi digitalne slike.* Bsc, Univerza v Ljubljani, 2010.

[8] B. Zupančič. *Zvezni regulacijski sistemi.* Fakulteta za elektrotehniko, 2010.

[9] Y. Kanayama, A. Nilipour in C. A. Lelm. A locomotion control method for autonomous vehicles. V *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, zv. 2, str. 1315–1317. 1988.

[10] Y. Kanayama, Y. Kimura in sod. A stable tracking control method for an autonomous mobile robot. V *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, str. 384–389. IEEE, Cincinnati, OH, 1990.

[11] C. Samson. Time-varying Feedback Stabilization of Car-like Wheeled Mobile Robots. *International Journal of Robotics Research*, zv. 12, št. 1, str. 55–64, 1993.

[12] P. A. Ioannou in J. Sun. *Robust Adaptive Control.* št. let. 1 v Control theory. PTR Prentice-Hall, 1996.

[13] S. Blažič. On Periodic Control Laws for Mobile Robots. *IEEE Transactions on Industrial Electronics*, zv. 61, št. 7, str. 3660–3670, 2014.

[14] S. Blažič. A novel trajectory-tracking control law for wheeled mobile robots. *Robotics and Autonomous Systems*, zv. 59, št. 11, str. 1001–1007, 2011.

[15] T. Takagi in M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, zv. SMC-15, št. 1, str. 116–132, 1985.

[16] K. Tanaka in M. Sano. A robust stabilization problem of fuzzy control systems and its application to backing up control of a truck-trailer. *Fuzzy Systems, IEEE Transactions on*, zv. 2, št. 2, str. 119–134, 1994.

[17] H. O. Wang, K. Tanaka in M. F. Griffin. An approach to fuzzy control of nonlinear systems: stability and design issues. *IEEE Transactions on Fuzzy Systems*, zv. 4, št. 1, str. 14–23, 1996.

[18] H. D. Tuan, P. Apkarian in sod. Parameterized linear matrix inequality techniques in

fuzzy control system design. *IEEE Transactions on fuzzy systems*, zv. 9, št. 2, str. 324–332, 2001.

[19] A. Ollero in O. Amidi. Predictive path tracking of mobile robots. application to the CMU navlab. V *Proceedings of 5th International Conference on Advanced Robotics, Robots in Unstructured Environments, ICAR*, zv. 91, str. 1081–1086. 1991.

[20] J. E. Normey-Rico, J. Gómez-Ortega in E. F. Camacho. A Smith-predictor-based generalised predictive controller for mobile robot path-tracking. *Control Engineering Practice*, zv. 7, št. 6, str. 729–740, 1999.

[21] F. Kuhne, W. F. Lages in J. M. G. da Silva Jr. Model predictive control of a mobile robot using linearization. V *Proceedings of mechatronics and robotics*, str. 525–530. 2004.

[22] D. Gu in H. Hu. Neural predictive control for a car-like mobile robot. *Robotics and Autonomous Systems*, zv. 39, št. 2, str. 73–86, 2002.

[23] G. Klančar in I. Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, zv. 55, št. 6, str. 460–469, 2007.

[24] J. Kennedy in R. Eberhart. Particle Swarm Optimization. V *Proceedings of IEEE International Conference on Neural Networks*, str. 1942–1948. IEEE, 1995.

[25] J. Kennedy in R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[26] J. Kennedy in R. Mendes. Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, zv. 36, št. 4, str. 515–519, 2006.

[27] F. Chaumette in S. Hutchinson. Visual servo control, part I: basic approaches. *IEEE Robotics and Automation Magazine*, zv. 13, št. 4, str. 82–90, 2006.

[28] P. I. Corke. *Visual control of robots: high-performance visual servoing*. Wiley, New York, 1996, 353 str.

[29] N. R. Gans in S. A. Hutchinson. Stable visual servoing through hybrid switched-system control. *IEEE Transactions on Robotics*, zv. 23, št. 3, str. 530–540, 2007.

[30] E. Malis, F. Chaumette in S. Boudet. 2-1/2-D visual servoing. *IEEE Transactions on Robotics and Automation*, zv. 15, št. 2, str. 238–250, 1999.

[31] E. Malis in F. Chaumette. 2 1/2 D visual servoing with respect to unknown objects through a new estimation scheme of camera displacement. *International Journal of Computer Vision*, zv. 37, št. 1, str. 79–97, 2000.

[32] D. Fontanelli, P. Salaris in sod. Unicycle-like Robots with Eye-in-Hand Monocular Cameras : From PBVS towards IBVS. V G. Chesi in K. Hashimoto (Uredniki), *Visual Servoing via Advanced Numerical Methods*, Lecture Notes in Control and Information Sciences, str. 335–360. Springer, London, 2010.

[33] A. De Luca, G. Oriolo in P. R. Giordano. Image-based visual servoing schemes for nonholonomic mobile manipulators. *Robotica*, zv. 25, št. 02, str. 131–145, 2007.

[34] G. L. Mariottini. *Image-based Visual Servoing for Mobile Robots: the Multiple View Geometry Approach*. Phd, University of Siena, 2005.

[35] G. Chesi in K. Hashimoto. *Visual Servoing via Advanced Numerical Methods*. Springer-Verlag, Berlin Heidelberg, 2010, 398 str.

[36] A. Cherubini in F. Chaumette. Visual navigation of a mobile robot with laser-based collision avoidance. *The International Journal of Robotics Research*, zv. 32, št. 2, str. 189–205, 2012.

[37] F. Chaumette in S. Hutchinson. Visual servo control, part II: advanced approaches. *IEEE Robotics and Automation Magazine*, zv. 14, št. 1, str. 109–118, 2007.

[38] M. Lepetič, G. Klančar in sod. Time optimal path planning considering acceleration limits. *Robotics and Autonomous Systems*, zv. 45, št. 3, str. 199–210, 2003.

4

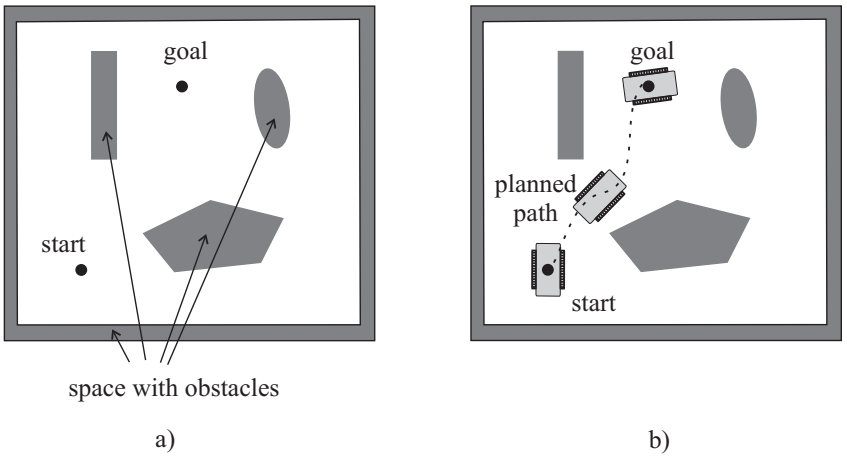
Načrtovanje poti

4.1 Uvod

Načrtovanje poti od točke A do točke B, hkratno izogibanje oviram in upoštevanje sprememb v okolju so za človeka enostavne naloge, medtem ko kolesnemu mobilnemu robotu predstavljajo izziv, ki ga mora (vsaj delno) premagati, da postane avtonomen. Robot s pomočjo senzorjev z določeno negotovostjo zaznava prostor okoli sebe in tako izdeluje ali dopolnjuje svoj zemljevid okolice. Za izračun premikov do cilja se s pomočjo algoritmov odloča in načrtuje potrebne akcije. Pri tem je potrebno upoštevati dinamične in kinematične omejitve robota.

Načrtovanje poti se uporablja za reševanje problemov na različnih področjih, od preprostega načrtovanja poti znotraj znanega okolja do določitve ustreznega zaporedja premikov za doseg cilja. Načrtovanje poti je pogosto omejeno na vnaprej zgrajena okolja in okolja, ki jih lahko vnaprej dovolj dobro opišemo. Načrtovanje poti se lahko uporablja v okoljih, ki so v celoti ali delno poznana, ter v popolnoma neznanih okoljih, kjer zaznane informacije določajo želeno gibanje robota.

Načrtovanje poti v znanih okoljih je aktualno področje raziskovanja, ki je temelj kompleksnejših primerov, kjer okolje ni znano vnaprej. V nadaljevanju so predstavljene najpogosteje uporabljene metode načrtovanja poti za kolesne mobilne robote. Za nadaljnje branje o metodah načrtovanja poti glejte [1–3]. Najprej podajmo definicije nekaj osnovnih pojmov pri načrtovanju poti.



Slika 4.1: (a) Okolica robota z ovirami ter začetno in ciljno konfiguracijo; (b) ena izmed možnih poti od začetne do ciljne konfiguracije

4.1.1 Okolica robota

Robot se giblje v okolju, ki je sestavljeno iz **prostega območja** in **območja z ovirami** (slika 4.1). V prostem območju se nahajata začetna in ciljna konfiguracija – množica parametrov, ki določa robota v prostoru. Parametri običajno vključujejo pozicijo in orientacijo robota, lahko pa tudi zasuke v njegovih sklepih. Število teh parametrov je enako številu prostostnih stopenj robota.

Okolje, ki vsebuje premikajoče se ovire, imenujemo **dinamično okolje**, okolje, ki se časovno ne spreminja, pa **statično okolje**. Pri *znanem okolju* je pozicija ovir vnaprej znana. V nasprotnem primeru govorimo o *neznanem okolju*.

4.1.2 Načrtovanje poti

Načrtovanje poti je proces iskanja zvezne poti, ki bo robota pripeljala od začetne do ciljne konfiguracije, tako da bo njegova celotna pot ležala v *prostem območju*, kot je prikazano na sliki 4.1. Pri načrtovanju poti mobilni sistem uporablja zemljevid okolja, ki je shranjen v njegovem spominu.

Stanje (ali konfiguracija) podaja možno lego mobilnega robota v okolju. Predstavimo ga lahko kot točko v konfiguracijskem prostoru, ki vključuje vsa možna stanja robota. Robot lahko preide iz enega stanja v drugo s pomočjo različnih akcij. Ustrezno pot opišemo z zaporedjem akcij, ki vodijo robota od začetne konfiguracije (oz. stanja) skozi nekaj vmesnih konfiguracij, potrebnih za doseg ciljne konfiguracije (oz. stanja). Izbira akcije v trenutnem ali naslednjem stanju, je odvisna od izbranega algoritma načrtovanja poti. Ta se glede na cenilko (kriterijsko funkcijo) odloči, katero je naslednje najprimernejše stanje iz množice ustreznih stanj. Cenilka je običajno določena z merjenjem razdalje, npr. najkrajša evklidska razdalja do ciljne točke.

Začetno in ciljno stanje pogosto povezuje več poti, ali pa pot sploh ne obstaja. Običajno obstaja več izvedljivih poti, na katerih robot ne trči z ovirami. Izbor zožimo z dodatnimi zahtevami ali kriteriji, ki določajo želeno optimalnost:

- dolžina poti naj bo najkrajša,
- ustrezna pot naj bo tista, ki jo robot lahko prevozi v najkrajšem možnem času,
- pot naj bo čim bolj oddaljena od ovir,
- pot naj bo gladka, brez ostrih zavojev,
- pot naj upošteva omejitve gibanja robota (primer neholonomičnosti, kjer v danem trenutku niso možne vse smeri vožnje).

4.1.3 Konfiguracija in konfiguracijski prostor

Stanje mobilnega sistema v nekem okolju imenujemo **konfiguracija** in jo opišemo z n podatki, ki predstavljajo vektor stanj $\boldsymbol{q} = [q_1, \ldots, q_n]^T$, kjer je n število prostostnih stopenj. Stanje $\boldsymbol{q}$ je točka v n-dimenzionalnem prostoru, ki ga imenujemo **konfiguracijski prostor** Q (angl. *configuration space*) in predstavlja vse možne konfiguracije mobilnega sistema glede na njegov kinematični model.

Del konfiguracijskega prostora, ki predstavlja ovire O_i, označimo z $Q_{obst} = \bigcup_i O_i$. Torej je prosti del okolja brez ovir Q_{free} enak

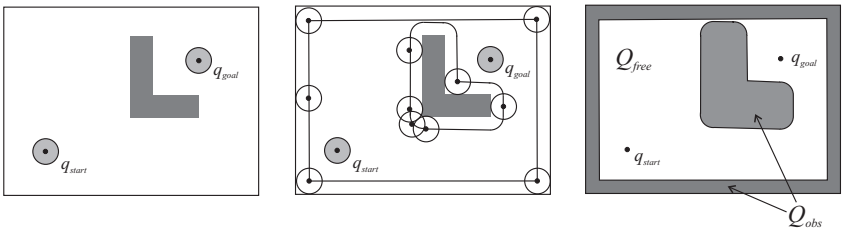
$$Q_{free} = Q - Q_{obst}$$

in predstavlja prostor, kjer lahko mobilni sistem načrtuje svoje gibanje.

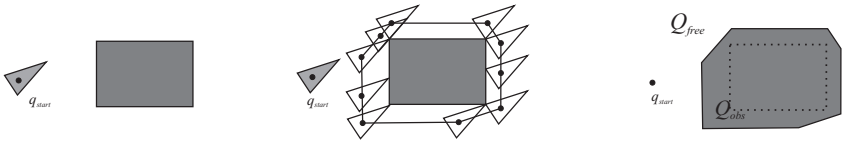
Predpostavimo, da imamo robota krožne oblike, ki je zmožen le translacij v ravnini, torej ima dve prostostni stopnji $\boldsymbol{q} = [x, y]^T$. Njegovo konfiguracijo lahko obravnavamo točkovno in jo enostavno predstavimo s točko njegovega centra x, y. Konfiguracijski prostor Q pa določimo s pomikanjem robota ob oviri, tako da je ves čas v stiku z njo, kar prikazuje slika 4.2. Pri tem točka centra robota, ki določa njegovo pozicijo, opiše mejo med Q_{free} in Q_{obs}. Na ta način razširimo dimenzije ovir za znano dimenzijo robota (njegov radij), da lahko robota obravnavamo kot točko.

Še en primer konfiguracijskega prostora za trikotnega robota in ovire je prikazan na sliki 4.3. Robot se lahko premika samo v smereh x in y ($\boldsymbol{q} = [x, y]^T$).

Če za robota na sliki 4.3 predpostavimo, da je zmožen tudi rotacije, ima njegova konfiguracija tri dimenzije $q = [x, y, \varphi]^T$, njegov konfiguracijski prostor pa je kompleksnejši. Poenostavljeno lahko konfiguracijski prostor določimo tako, da obliki robota očrtamo krog, ki ima središče v točki centra robota. Dobljeni prostor Q_{free} je v tem primeru nekoliko manjši kot dejanski prosti prostor, ker ima očrtan krog večjo površino kot robot. Vendar pa to poenostavi problem načrtovanja poti.



Slika 4.2: (a) Krožni robot v okolju z oviro, z označeno začetno in ciljno konfiguracijo, (b) določitev konfiguracijskega prostora in (c) načrtovanje poti, ki okolje (a) prevede v konfiguracijski prostor, kjer je robot predstavljen s točko svojega centra



Slika 4.3: (a) Pravokotna ovira in trikotni robot s točko, ki določa njegovo konfiguracijo $\boldsymbol{q}$, (b) določitev konfiguracijskega prostora, (c) prosti konfiguracijski prostor Q_{free} in prostor ovir Q_{obs}

4.1.4 Matematični zapis oblike in lege ovire v okolici

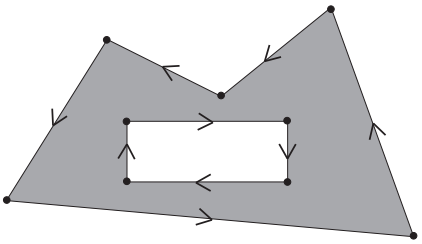
Za izračun konfiguracijskega prostora robota in uporabo nadaljnjih poenostavitev okolja je potreben matematični opis oblike in lege ovir v prostoru. Najpogostejša pristopa za opis ovir sta: predstavitev meje s pomočjo oglišč in predstavitev s polravninami.

Predstavitev ovir z zapisom meje s pomočjo oglišč

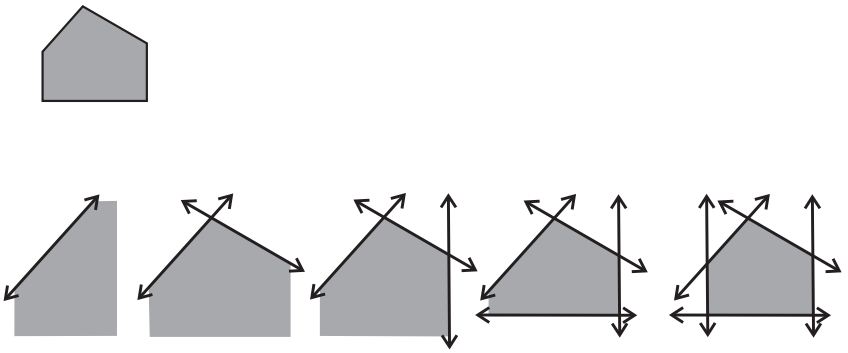
Ovira na tlorisu predstavlja m-strani poligon, ki ima m oglišč. Mejo ovire lahko zapišemo z nizanjem oglišč v obratni smeri urinega kazalca, pri čemer so luknje v ovirah in okoliška stena zapisane v nasprotni smeri, tj. v smeri urinega kazalca (slika 4.4). To velja tako za zapis konveksnih kot tudi nekonveksnih poligonov.

Predstavitev ovir s presekom polravnin

Konveksni poligon z m oglišči lahko zapišemo kot unijo m polravnin, kjer je vsaka določena s svojo enačbo premice $f(x, y) \leq 0$ oz. ravnine $f(x, y, z) \leq 0$ (tridimenzionalne ovire). Slika 4.5 prikazuje primer tovrstnega opisa peterokotnika.



Slika 4.4: Primer opisa ovire z nizom oglišč: luknja ovire je opisana z nizom v smeri urinega kazalca, zunanja meja ovire pa z nizom v nasprotni smeri. Leva stran vsakega usmerjenega linearnega segmenta pripada oviri (zasenčena površina).



Slika 4.5: Primer opisa ovire s polravninami

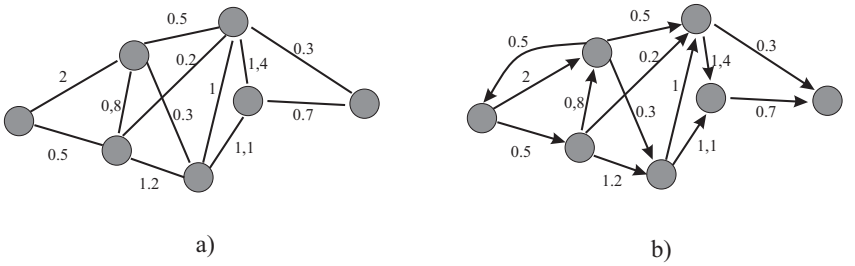
Nekonveksne like in like z luknjami opišemo s pomočjo operacij nad množicami, npr. unija, presek, razlika množic itd.

4.2 Predstavitev okolja za načrtovanje poti

Pred samim načrtovanjem poti moramo okolje predstaviti na poenoten matematičen način, primeren za obdelavo z algoritmi iskanja poti.

4.2.1 Predstavitev z grafi

Konfiguracijski prostor je sestavljen iz prostega območja, ki predstavlja vse možne konfiguracije (stanja) mobilnega sistema, in območja z ovirami. Če prosto območje skrčimo in ga predstavimo s podmnožico konfiguracij (npr. središča področij ali celic), ki vključujejo začetne in ciljne konfiguracije ter želeno število vmesnih konfiguracij s prehodi med njimi, dobimo **graf prehajanja stanj**. Stanja v grafu so prikazana s krogi in jih imenujemo *vozlišča* grafa, povezave



Slika 4.6: Primer (a) uteženega grafa in (b) usmerjenega uteženega grafa

med njimi pa s črtami, ki predstavljajo *akcije*, potrebne za prehod med stanji.

Graf je **utežen**, če vsaki povezavi pripišemo neko utež ali ceno, ki je potrebna za izvršitev akcije pri prehodu med stanjema te povezave. V **usmerjenem grafu** (angl. *directed graph*) pa povezave označimo še s smerjo. V usmerjenem grafu je cena odvisna od smeri prehoda, medtem kot je v *neusmerjenem grafu* možen prehod v obeh smereh. Slika 4.6 prikazuje utežen in usmerjen utežen graf. Iskanje poti v grafu prehajanja stanj je možno z različnimi algoritmi iskanja, kot je A*, Dijkstrov algoritem itd.

4.2.2 Razcep na celice

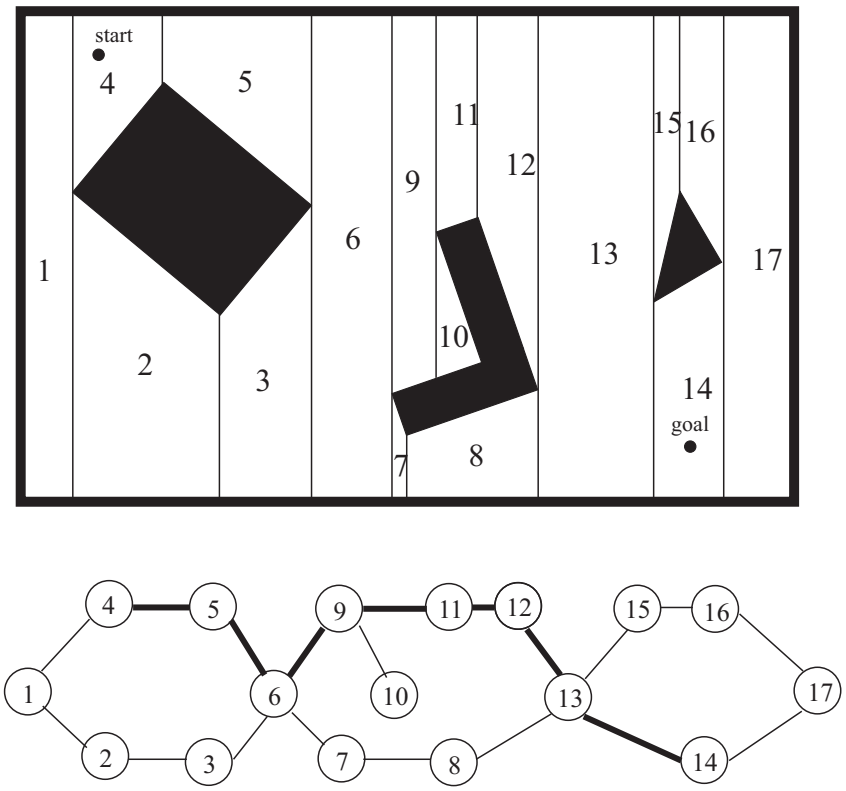
Okolje lahko razdelimo na **celice**, ki predstavljajo enostavne geometrijske like. Celice so konveksne, saj mora vsaka daljica, ki povezuje poljubni konfiguraciji v celici, v celoti ležati znotraj celice. Po razcepu okolja na celice lahko izvedemo graf prehajanja stanj, kjer so stanja določene točke v celici (npr. težišča), povezave med stanji (vozlišča grafa) pa so možne le med sosednjimi celicami s skupnim robom ali ogliščem.

Natančen razcep na celice

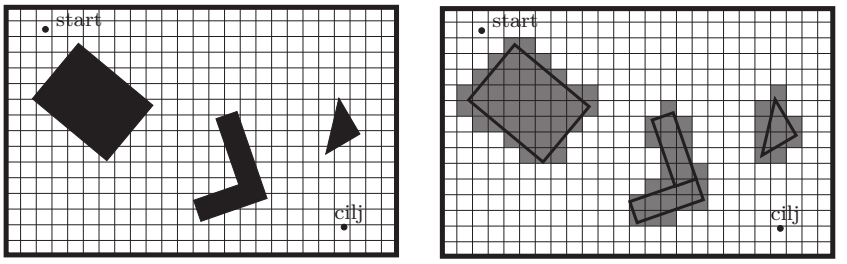
Razcep okolja na celice je natančen, če celice v celoti ležijo ali v prostem območju ali v območju z ovirami. **Natančen razcep** je "brezizguben", saj je unija vseh prostih celic enaka prostemu konfiguracijskemu prostoru Q_{free}.

Primer natančnega razcepa na celice je navpičen razcep, prikazan na sliki 4.7. V tem primeru se z navidezno navpično črto pomikamo čez okolico od leve proti desni (meji). Vsakič, ko prečkamo oglišče katerega izmed večkotnikov, ustvarimo navpično *mejo med celicama*, ki lahko poteka samo navzgor, samo navzdol ali pa gor in dol od oglišča. Kompleksnost tega pristopa je močno odvisna od geometrije okolice. V enostavnih okolicah bo število celic in povezav med njimi majhno. Z večanjem števila poligonov (ovir) in njihovih oglišč, narašča tudi število celic.

Natančen razcep na celice lahko predstavimo z grafom prehajanja stanj, kjer so vozlišča središča celic, prehodi med središči celic pa gredo skozi točke na središču



Slika 4.7: Navpičen razcep na celice (zgoraj) in pripadajoč graf (spodaj) z vrisano potjo med začetno in ciljno konfiguracijo (odebeljena črta)



a) b)

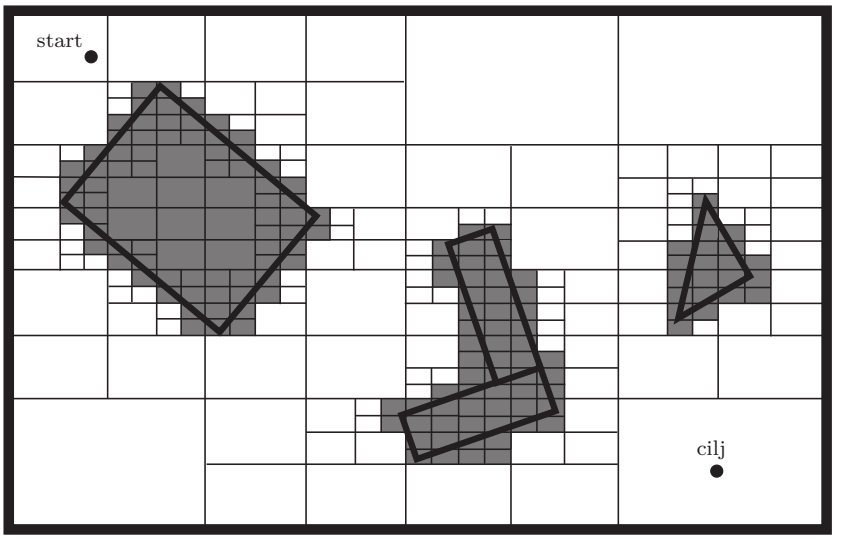
Slika 4.8: Približen razcep na celice je sestavljen iz dveh korakov: (a) na okolico položimo mrežo celic enake velikosti in (b) označimo celice kot proste ali zasedene (zasenčene celice)

mej med celicami.

Približen razcep na celice

Razcep okolja na celice je približen, ko posamezna celica vsebuje tako prosti konfiguracijski prostor kot tudi oviro ali del nje. Celice, ki vsebujejo vsaj del ovire, označimo kot zasedene, ostale pa so proste. Večinoma se uporablja razcep na celice enakih velikosti, kjer dobimo **mrežo zasedenosti** (angl. *occupancy grid*) (slika 4.8). Center vsake celice (na sliki 4.8 so proste celice pobarvane z belo barvo) je v grafu predstavljen kot vozlišče. Povezave med celicami so možne v štirih ali osmih smereh, odvisno od tega, ali je dovoljeno prehajanje v diagonalni smeri. Omenjen pristop je zelo enostaven za uporabo, vendar lahko zaradi konstantne velikosti celic pride do izgube informacij o okolici (ta razcep ni "brezizguben"); npr. ovire se povečajo in obstoječi ozki prehodi med njimi lahko izginejo pri približnem razcepu na celice. Glavna pomanjkljivost tega pristopa je poraba pomnilnika, ki je za večja okolja velika, ne glede na to, ali so okolja enostavna ali kompleksna.

Do manjše izgube informacij pri majhni porabi pomnilnika pride pri uporabi spremenljive velikosti celic. Na okolje položimo eno celico, ki ga popolnoma pokrije. Če je celotna celica v prostem območju ali območju z ovirami, ostane takšna kot je. Če pa je le del nje pokrit z oviro, celico razdelimo na 4 manjše celice. Postopek, imenovan **štiriško drevo** (angl. *quadtree*), ponavljamo, dokler ni dosežena želena resolucija. Dobljeno razdelitev okolja na celice (slika 4.9) lahko prav tako pretvorimo v graf stanj. Približen razcep na celice je enostavnejši od natančnega, vendar lahko zaradi izgube informacij vodi do problema, ko proces načrtovanja poti ne najde rešitve, čeprav le-ta obstaja.



Slika 4.9: Približen razcep na celice z uporabo spremenljive velikosti celic — štiriško drevo. Proste celice so označene z belo barvo, zasedene pa so zasenčene

Primer 4.1

Napišite program, ki z uporabo štiriškega drevesa razgradi okolje s pravokotnimi ovirami. Program 4.1 ustvari naključno okolje z ovirami; v programu je določena tudi funkcija za izračun štiriškega drevesa. Funkcija sprejme ovire, dimenzijo okolja in želeno globino (število delitev) štiriškega drevesa ter vrne štiriško drevo v obliki Matlab strukture.

Program 4.1: Implementacija štiriškega drevesa

./src/pth/example_quad_tree.m

```
bb = [0, 16, 0, 12]; % Dimenzija okolja: xa, xb, ya, yb
N = 10; % Število ovir
minDim = [0.1; 0.1]; % Minimalne dimenzije ovire: xMin in yMin
maxDim = [2; 2]; % Maksimalne dimenzije ovire: xMax in yMax
% Naključni zemljevid ovir, oglišča v stolpcu: x1, y1, x2, y2, x3, y3, x4, y4
obst = zeros(8, N);
for i = 1:N
    p = [bb(1); bb(3)] + [diff(bb(1:2)); diff(bb(3:4))].*rand(2,1);
    phi = 2*pi*rand(); d = minDim/2 + (maxDim-minDim)/2*rand();
    R = [cos(phi),-sin(phi); sin(phi), cos(phi)];
    v = repmat(p, 1, 4) + R*([-1, 1, 1,-1; -1,-1, 1, 1].*repmat(d, 1, 4));
    obst(:,i) = reshape(v, [], 1);
end

tree = quadTree(obst, bb, 4); % Izdelava štiriškega drevesa globine 4
```

Rešitev

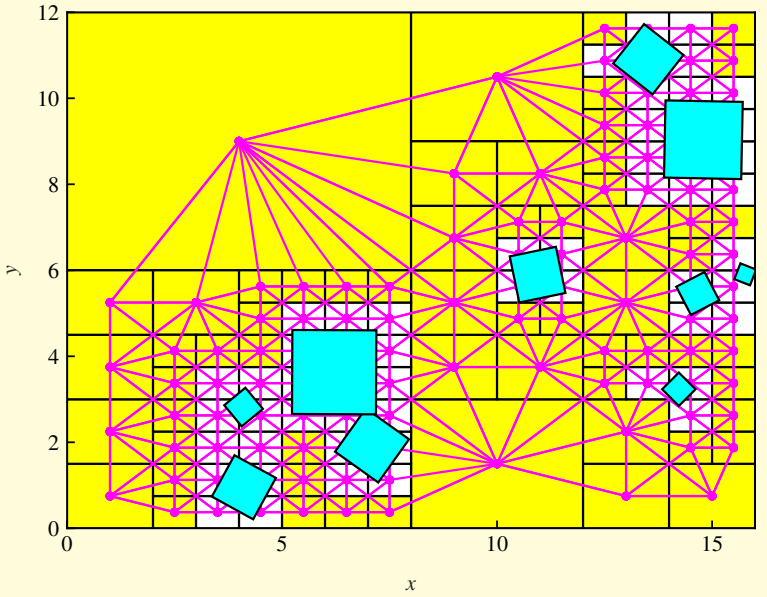
Možna izvedba algoritma s štiriškim drevesom je podana v Matlab kodi v programu 4.2. Rezultat algoritma na naključnem zemljevidu je prikazan na sliki 4.10.

Program 4.2: Razcep na celice s štiriškim drevesom

./src/pth/quadTree.m

```
function tree = quadTree(obst, bb, level)
    % Izdelava štiriškega drevesa $tree$ globine $level>=0$ okoli ovir $obst$
    % (vsak stolpec vsebuje oglišča ovire: x1, y1, x2, y2, ...) in
    % za okolje dimenzij $bb$ (xa, xb, ya, yb)
    minDim = [diff(bb(1:2)); diff(bb(3:4))]/2^level; % Minimalna velikost celice
    % Osnovno vozlišče
    tree(1).leaf = true; % % Ali je celica končno vozlišče?
    tree(1).free = false; % Ali je celica zasedena?
    tree(1).bounds = bb; % Meje celice: xa, xb, ya, yb
    tree(1).center = [mean(bb(1:2)); mean(bb(3:4))]; % Center celice

    id = 1; k = 2;
    while id < k
        occupied = isOccupied(tree(id).bounds, obst);

        d = [diff(tree(id).bounds(1:2)), diff(tree(id).bounds(3:4))]/2;
        if occupied && d(1) > minDim(1)/2 % Razdelitev celice na štiri nove celice
            tree(id).leaf = false;
            tree(id).free = false;

            b = tree(id).bounds;
            bs = [b(1),      b(1)+d(1), b(3),      b(3)+d(2); ...
                  b(1)+d(1), b(2),      b(3),      b(3)+d(2); ...
                  b(1),      b(1)+d(1), b(3)+d(2), b(4); ...
                  b(1)+d(1), b(2),      b(3)+d(2), b(4)];
            for i = 1:4
                tree(k).leaf = true;
                tree(k).free = false;
                tree(k).bounds = bs(i,:);
                tree(k).center = [mean(bs(i,1:2)); mean(bs(i,3:4))];
                k = k + 1;
            end
        elseif ~occupied
            tree(id).free = true;
        end
        id = id + 1;
    end

    % Izdelava vidljivostnega grafa
    a = zeros(2,length(tree)*4); leafs = zeros(1,length(tree));
    for i = 1:length(tree)
        a(:,i*4-3:i*4) = tree(i).bounds([1, 2, 2, 1; 3, 3, 4, 4]);
        leafs(i) = tree(i).leaf;
    end
    offset = [-1, 1, 1,-1; -1,-1, 1, 1]/2.*repmat(minDim, 1, 4);
    for i = 1:length(tree)
        tree(i).neighbours = [];
        if tree(i).leaf
            b = tree(i).bounds([1, 2, 2, 1; 3, 3, 4, 4]) + offset;
            c = reshape(inpolygon(a(1,:), a(2,:), b(1,:), b(2,:)), 4, []);
            tree(i).neighbours = setdiff(find(any(c).*leafs), i);
        end
    end
```

```
end

function occupied = isOccupied(bounds, obst)
    occupied = false;
    pb = bounds([1, 2, 2, 1, 1; 3, 3, 4, 4, 3]);
    for j = 1:size(obst, 2) % Sprehod čez vse ovire
        pa = reshape(obst(:,j), 2, []); N = size(pa, 2);
        ina = inpolygon(pa(1,:), pa(2,:), pb(1,:), pb(2,:));
        inb = inpolygon(pb(1,:), pb(2,:), pa(1,:), pa(2,:));
        if any(ina) || any(inb) % Ali so oglišča v oviri ali celici?
            occupied = true; break;
        else % Ali so kakšna presečišča robov?
            for k = 1:size(pb, 2)-1 % Sprehod čez vse mejne robove
                for i = 1:N % Sprehod čez robove ovir
                    a1 = [pa(:,i); 1]; a2 = [pa(:,mod(i,N)+1); 1];
                    b1 = [pb(:,k); 1]; b2 = [pb(:,k+1); 1];
                    pc = cross(cross(a1, a2), cross(b1, b2));% Presečišče
                    if abs(pc(3))>eps
                        pc = pc/pc(3);
                        da = a2-a1; ca = pc-a1; ea = (ca.'*da)/(da.'*da);
                        db = b2-b1; cb = pc-b1; eb = (cb.'*db)/(db.'*db);
                        if eb>eps && eb<1 && ea>eps && ea<1
                            occupied = true; break;
                        end
                    end
                end
                if occupied, break; end
            end
        end
        if occupied, break; end
    end
end
```



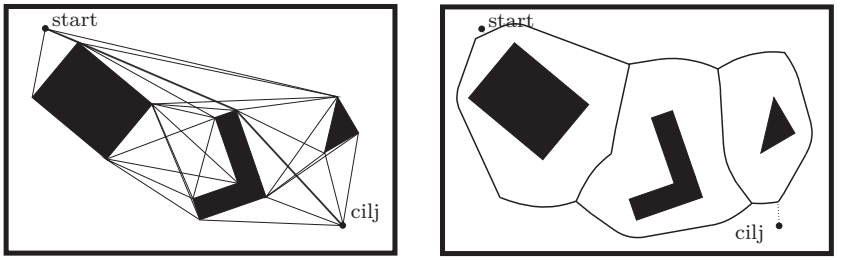
Slika 4.10: Razcep okolja z naključnimi ovirami s pomočjo štiriškega drevesa. Na štiriško drevo je položena mreža, ki povezuje središča vseh sosednjih celic.

4.2.3 Zemljevid cest

Zemljevid cest, sestavljen iz črt, krivulj in njihovih stičišč, podaja povezljivost prostega območja okolja. Proces načrtovanja poti mora povezati začetno in ciljno točko z obstoječimi povezavami na zemljevidu in poiskati povezano zaporedje cest. Postavitev cest je odvisna od geometrije okolice. Cilj je uporabiti najmanjše število cest, ki robotu omogočajo dostop do kateregakoli dela prostega območja. V nadaljevanju so predstavljeni trije načini izdelave zemljevida cest: graf vidljivosti, Voronojev graf in triangulacija prostora.

Graf vidljivosti

Graf vidljivosti je sestavljen iz vseh možnih povezav med dvema ogliščema, ki v celoti ležita v prostem območju. To pomeni, da so za vsako oglišče vzpostavljene povezave z vsemi (drugimi) oglišči, ki so vidna z njegove pozicije. Pri tem začetno in ciljno točko obravnavamo kot dodatni oglišči. Povezave ustvarimo tudi med sosednjima ogliščema istega poligona. Primer grafa vidljivosti je prikazan na sliki 4.11a. Pot, dobljena s pomočjo grafa vidljivosti, je najkrajša možna, saj so ceste speljane kar se da blizu oviram. Da rešimo problem trka robota in



a) b)

Slika 4.11: Zemljevid cest: (a) graf vidljivosti in (b) Voronojev graf

ovir, lahko le-te povečamo vsaj za polmer robota, kot je opisano v podpoglavju 4.1.3. Grafi vidljivosti so dokaj enostavni za uporabo, vendar z večanjem števila ovir in njihove kompleksnosti narašča število cestnih povezav in vozlišč, zato se manjša njihova učinkovitost. Grafe vidljivosti lahko poenostavimo z odstranitvijo redundantnih povezav, ki jih je možno nadomestiti z obstoječo krajšo povezavo.

Voronojev graf

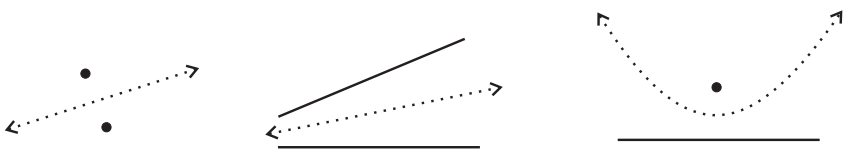
Voronojev graf (slika 4.11b) je sestavljen iz odsekov cest, ki so najbolj oddaljeni od ovir. To pomeni, da je povezava med dvema ovirama enaka razdalji do obeh ovir.

Osnovni Voronojev graf (diagram) je opredeljen za ravnino z n točkami (npr. točkovna ovira). Ravnina se razdeli na n območij, katerih meje sestavljajo zemljevid. Vsako območje ima natanko eno izvorno točko in velja, da so vse točke znotraj določenega območja bližje svoji izvorni točki, kot katerikoli drugi izvorni točki (drugih območij). Na sliki 4.11 je prikazan primer splošnega Voronojevega grafa za ravninsko okolje, v katerem so ovire poljubnih oblik (trikotnik, pravokotnik, mnogokotnik itd.). Tu je prostor razdeljen na območja, kjer ima vsako območje točno eno izvorno oviro. Vsaka točka znotraj določenega območja je bližje izvorni oviri kot katerikoli drugi oviri.

Vožnja po takšni cesti zmanjšuje tveganje za trk robota z ovirami, kar je zaželeno, ko je lega robota znana z neko negotovostjo zaradi merilnega šuma ali vodenja. Zemljevid okolja, zgrajen iz poligonov, vsebuje tri značilne **Voronojeve krivulje**, kot je prikazano na sliki 4.12. Voronojeva krivulja, ki ima enako razdaljo med:

- dvema ogliščema (premica),
- dvema robovoma (ista premica),
- ogliščem in daljico (parabola).

V cestno omrežje povežemo še začetno in ciljno konfiguracijo. Tako dobimo graf, po katerem iščemo rešitev.



Slika 4.12: Tipične Voronojeve krivulje

Kot smo že omenili, ta pristop maksimira oddaljenost robota do ovir, vendar pridobljena dolžina poti še zdaleč ni optimalna (najkrajša). Robot s senzorji oddaljenosti (npr. z ultrazvočnim ali laserskim pregledovalnikom razdalj) lahko z enostavno regulacijo, kjer se giba na enaki oddaljenosti od vseh okoliških ovir, sledi cestam po Voronojevem grafu. Roboti z bližinskimi tipali ali senzorji za kratke razdalje pa imajo pri tem pristopu probleme z lokalizacijo, ki jih pri grafu vidljivosti ne bi imeli.

Primer 4.2

Z uporabo Matlab funkcije `voronoi` izračunajte Voronojev graf za okolje na sliki 4.11. Koordinate okolja (`o`) in objektov (`o1`, `o2`, `o3`) so podane v programu 4.3.

Program 4.3: Oglišča ovir v okolju

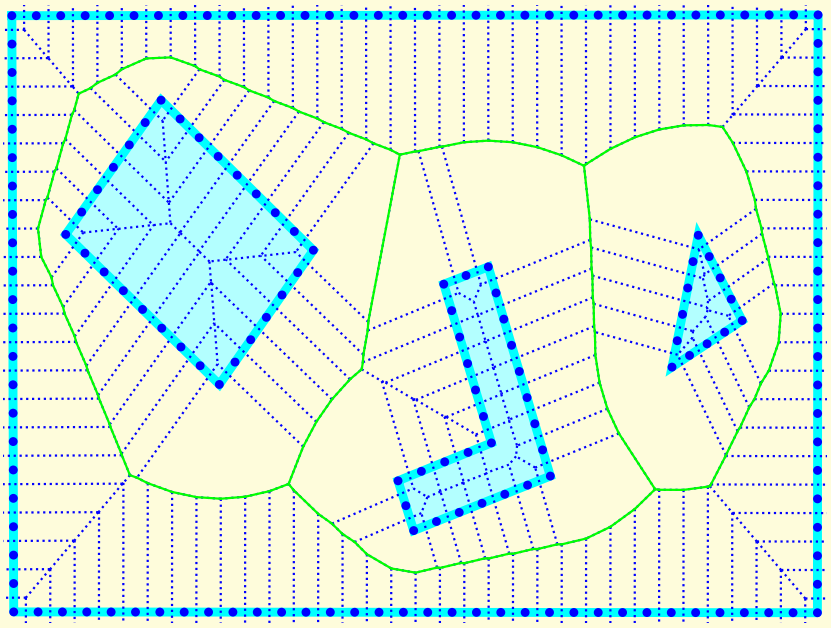
./src/pth/param_map.m

```
o  = 1000*[0.0149, 0.0693; ...
           1.6228, 0.0679; ...
           1.6241, 1.0867; ...
           0.0112, 1.0854];
o1 = 1000*[0.4263, 0.4569; ...
           0.6144, 0.6857; ...
           0.3097, 0.9414; ...
           0.1190, 0.7126];
o2 = 1000*[0.8151, 0.2079; ...
           1.0885, 0.3008; ...
           0.9644, 0.6574; ...
           0.8753, 0.6278; ...
           0.9706, 0.3573; ...
           0.7838, 0.2927];
o3 = 1000*[1.3319, 0.4865; ...
           1.4723, 0.5659; ...
           1.3845, 0.7112];
% Prosto območje je na desni strani od ovire
obstacles = {flipud(o), o1, o2, o3};
```

Rešitev

S pomočjo Matlab funkcije `voronoi` lahko narišemo Voronojev graf za seznam točk. Okolje na sliki 4.11 vključuje tudi like z robovi, ki so opisani z daljicami, zato ni možno neposredno uporabiti funkcije `voronoi`. Vsako daljico lahko poljubno natančno predstavimo z diskretno množico (enakomerno) porazdeljenih pomožnih točk. S pomočjo funkcije `voronoi` lahko nato izračunamo aproksi-

macijo Voronojevega grafa, kot je prikazano na sliki 4.13. Poleg iskanih robov med ovirami dobimo s tem postopkom tudi veliko dodatnih robov, ki so na sliki 4.13 označeni črtkano. Gre za robove, ki ločujejo pomožne točke, niso v prostem prostoru ali pa so posledica nekonveksnosti ovir. Program 4.4 vsebuje filtrirne mehanizme, ki odstranijo vse dodatne robove, tako da dobimo končno aproksimacijo Voronojevega grafa — polno izvlečeni robovi na sliki 4.13.



Slika 4.13: Aproksimacija Voronojevega grafa z uporabo Matlab funkcije `voronoi` in pomožnih točk za predstavitev robov

Program 4.4

./src/pth/example_voronoi.m

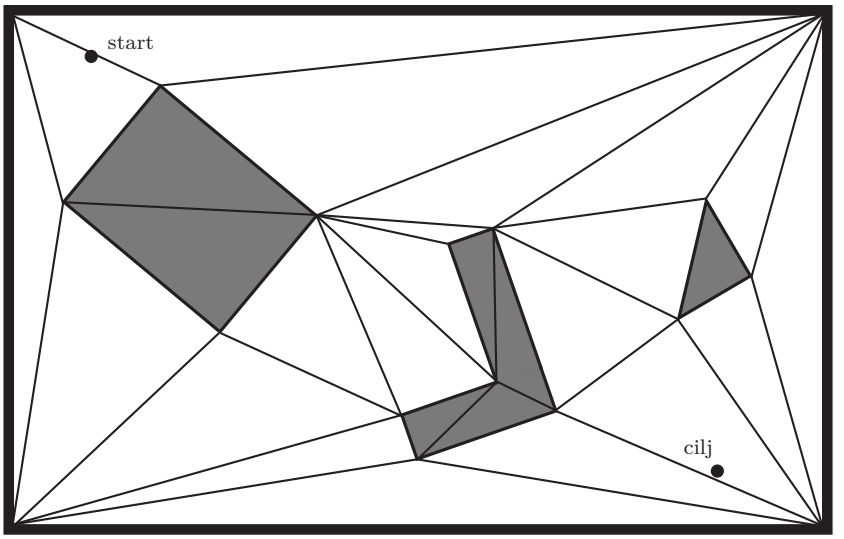
```
1  param_map;
2
3  % Opis daljic z množico pomožni /(vmesnih) točk
4  dMin = 50;
5  points = []; obst = [];
6  B = length(obstacles);
7  for i = 1:B
8      ob = obstacles{i};
9      M = size(ob, 1);
10     for j = 1:M
11         k = mod(j, M)+1; % j+1
12         d = sqrt((ob(j,1)-ob(k,1))^2 + (ob(j,2)-ob(k,2))^2);
13         n = ceil(d/dMin)+1;
14         x = linspace(ob(j,1), ob(k,1), n).';
15         y = linspace(ob(j,2), ob(k,2), n).';
16         points = [points; x(1:end-1) y(1:end-1)];
17     end
```

```
    obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

% Izračun Voronoijevih daljic iz množice pomožnih točk
[vx, vy] = voronoi(points(:,1), points(:,2));

% Odstanjevanje pomožnih (niso v prostem prostoru) Voronoijevih robov
s = false(1, size(vx, 2));
for j = 1:size(vx, 2)
    in = inpolygon(vx(:,j), vy(:,j), obst(:,1), obst(:,2));
    s(j) = all(in==1);
end
ux = vx(:,s); uy = vy(:,s); % Približni Voronoijevi robovi

% Odstranjevanje nazaključenih robov pri konveksnih objektih
r = [];
for j = 1:length(ux(:))
    c = length(find((ux(:)-ux(j)).^2 + (uy(:)-uy(j)).^2 < eps));
    if c==1
        a = j; % Odkrita robna točka na nezaključenem robu
        while true % Iskanje vseh točk na nezaključenem robu
            if mod(a, 2) == 0, b = a-1; else b = a+1; end
            r = [r, max(a,b)/2];
            c = find((ux(:)-ux(b)).^2 + (uy(:)-uy(b)).^2 < eps);
            if length(c)>2, break; end
            a = c(c~=b);
        end
    end
end
ux(:,r) = []; uy(:,r) = [];

% Izris
plot([vx;nan(1,size(vx,2))], [vy;nan(1,size(vy,2))], 'b:'); hold on;
plot(points(:,1), points(:,2), 'b.');
plot([ux;nan(1,size(ux,2))], [uy;nan(1,size(uy,2))], 'g-', 'LineWidth', 1);
axis equal tight;
```

Triangulacija prostora

Pri triangulaciji prostora je okolje razcepljeno na trikotne celice. Čeprav obstajajo različni algoritmi triagulacije, je dober pristop, ki preprečuje ozke trikotnike, še vedno aktualen raziskovalni problem [2]. Eden od možnih algoritmov je *Delaunayeva triangulacija prostora*, ki je dual Voronojevega grafa. V Delaunayevem grafu središče vsakega trikotnika (središče očrtanega kroga) sovpada z vsakim ogliščem Voronojevega poligona. Primer delaunayeve triangulacije prostora je prikazan na sliki 4.14.

Iz pristopov, ki brezizgubno upodobijo okolico, je s pomočjo kasneje predstavljenih algoritmov možno v končnem času dobiti informacijo, ali iskana pot obstaja ali ne. Pravimo, da so ti pristopi popolni (angl. *complete*).



Slika 4.14: Delaunayeva triangulacija prostora, kjer nekateri robovi trikotnika sovpadajo z robovi ovir

Primer 4.3

Uporabite Delaunayevo triangulacijo prostora za okolje na sliki 4.11. Točke, ki določajo meje ovir, so navedene v programu 4.3.

Rešitev

Za znana oglišča lahko v programskem okolju Matlab s pomočjo funkcije `DelaunayTri` izračunamo Delaunayevo triangulacijo prostora, s funkcijo `triplot` pa jo narišemo, kot je prikazano v programu 4.5.

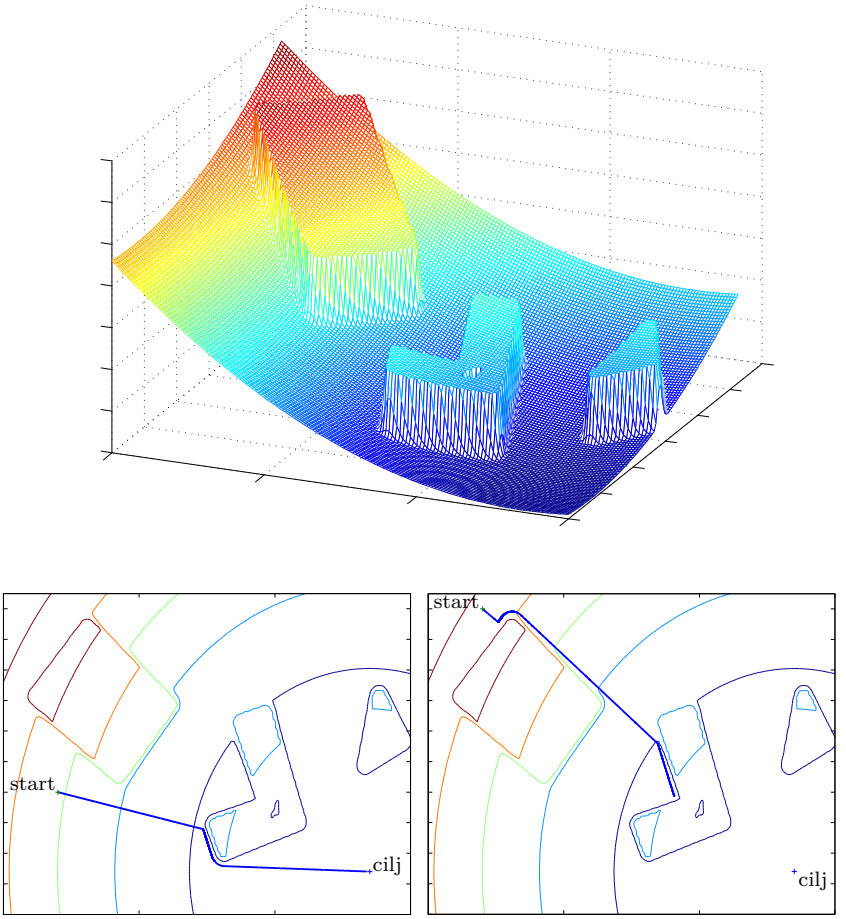
Program 4.5: Delaunayeva triangulacija prostora

./src/pth/example_delaunay.m

```
1 param_map;
2
3 points = cell2mat(obstacles(:));
4 dt = delaunayTriangulation(points);
5 triplot(dt, 'b-'); axis equal tight;
```

4.2.4 Potencialno polje

Okolica je predstavljena s **potencialnim poljem**, ki ga lahko razumemo kot namišljeno višino. Ciljna točka je na dnu, višina polja pa narašča z oddaljenostjo od ciljne točke ter je na ovirah še višja. Postopek načrtovanja poti si lahko



Slika 4.15: Potencialno polje za znano ciljno točko (zgoraj) ter ekvipotencialne krivulje in izračunana pot za dve začetni točki (spodaj), kjer izračunana pot doseže cilj (levo) in kjer je pot ujeta v nekonveksno oviro (desno)

predstavljamo kot premikanje žoge, ki se kotali po hribu navzdol do cilja v dolini, kar je prikazano na sliki 4.15.

Potencialno polje je vsota privlačnega polja ciljne točke $U_{attr}(\boldsymbol{q})$ in odbojnega polja ovir $U_{rep}(\boldsymbol{q})$

$$U(\boldsymbol{q}) = U_{attr}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q}) \tag{4.1}$$

Ciljna točka je globalni minimum potencialnega polja.

Privlačni potencial $U_{atr}(\boldsymbol{q})$ v (4.1) je lahko določen tako, da je sorazmeren kvadratu evklidske razdalje do ciljne točke $D(\boldsymbol{q}, \boldsymbol{q}_{goal}) = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2}$ kot

$$U_{attr}(\boldsymbol{q}) = k_{attr} \frac{1}{2} D^2(\boldsymbol{q}, \boldsymbol{q}_{goal})$$

kjer je k_{atr} pozitivna konstanta.

Odbojni potencial $U_{rep}(\boldsymbol{q})$ naj bo zelo velik v bližini ovir, z oddaljenostjo od ovir $D(\boldsymbol{q}, \boldsymbol{q}_{obst})$ pa se zmanjšuje. Njegova vrednost je nič, ko je $D(\boldsymbol{q}, \boldsymbol{q}_{obst})$ večja od mejne vrednosti D_0. Odbojni potencial je lahko predstavljen kot

$$U_{rep}(\boldsymbol{q}) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{D(\boldsymbol{q}, \boldsymbol{q}_{obst})} - \frac{1}{D_0} \right)^2 & ; \quad D(\boldsymbol{q}) \leq D_0 \\ 0 & ; \quad D(\boldsymbol{q}) > D_0 \end{cases} \tag{4.2}$$

kjer je k_{rep} pozitivna konstanta, $D(\boldsymbol{q}, \boldsymbol{q}_{obst})$ pa razdalja do najbližje točke na najbližji oviri.

Za opis poti od začetne do ciljne točke mora robot slediti negativnemu gradientu potencialnega polja ($-\nabla U(\boldsymbol{q})$).

Primer 4.4

Določite negativni gradient potencialnega polja (4.1).

Rešitev

Negativni gradient privlačnega polja (4.2.4) je

$$-\nabla U_{attr}(\boldsymbol{q}) = -k_{attr} \frac{1}{2} \begin{bmatrix} 2(x - x_{goal}) \\ 2(y - y_{goal}) \end{bmatrix} = k_{attr}(\boldsymbol{q}_{goal} - \boldsymbol{q})$$

in kaže v smeri od lege robota $\boldsymbol{q}$ do cilja $\boldsymbol{q}_{goal}$, njegova dolžina pa je proporcionalna razdalji od $\boldsymbol{q}$ do $\boldsymbol{q}_{goal}$.

Določimo še negativni gradient odbojnega polja 4.2 za primer, ko je $D(\boldsymbol{q}) \leq D_0$

$$\begin{aligned} -\nabla U_{rep}(\boldsymbol{q}) &= -k_{rep} \left(\frac{1}{D_{obst}} - \frac{1}{D_0} \right) \frac{-1}{D_{obst}^2} \nabla D_{obst} = \\ &= k_{rep} \left(\frac{1}{D_{obst}} - \frac{1}{D_0} \right) \frac{1}{D_{obst}^3} (\boldsymbol{q} - \boldsymbol{q}_{obst}) \end{aligned}$$

kjer je $D_{obst} = D(\boldsymbol{q}, \boldsymbol{q}_{obst}) = \sqrt{(x - x_{obst})^2 + (y - y_{obst})^2}$. Vidimo, da smer odbojnega polja vedno kaže stran od ovire, njegova jakost pa se zmanjšuje z oddaljenostjo od ovire. Za primer, ko je $D_{obst} > D_0$, pa je odbojni gradient $-\nabla U_{rep}(\boldsymbol{q}) = 0$.

Pri predstavitvi okolja z uporabo potencialnega polja lahko robot preprosto doseže ciljno točko s sledenjem negativnega gradienta potencialnega polja, ki ga eksplicitno izračunamo iz znane pozicije robota. Glavna pomanjkljivost tega pristopa je možnost ujetosti robota (tresenje – angl. *jittering behaviour*) v lokalnem minimumu. Do tega lahko pride, če okolje vsebuje kakršnokoli nekonveksno oviro (slika 4.15) in začne robot oscilirati med več enako oddaljenimi točkami od ovire. Načrtovanje poti z uporabo potencialnega polja se lahko uporabi

za izračun referenčne poti, ki ji mora robot slediti, ali pa pri vodenju gibanja za usmerjanje robota v trenutni smeri negativnega gradienta.

Primer 4.5

Izračunajte potencialno polje iz slike 4.15 za okolico z ovirami. Koordinate ovir so podane v programu 4.3.

Rešitev

Potencialno polje izračunamo s pomočjo enačb (4.1) – (4.2), kjer je pot izračunana kot integral negativnega gradienta iz primera 4.4.

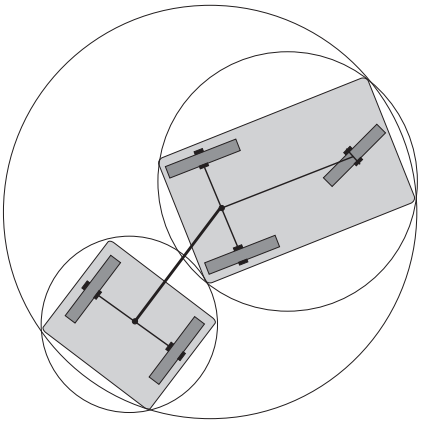
4.2.5 Načrtovanje poti z metodami vzorčenja prostora

Do sedaj predstavljene metode za predstavitev okolja za namen planiranja poti so zahtevale znano eksplicitno predstavitev prostega konfiguracijskega prostora. Z večanjem dimenzije konfiguracijskega prostora postanejo te metode preveč časovno potratne, zato lahko v teh primerih uporabimo metode vzorčenja prostora.

Pri načrtovanju poti z vzorčenjem prostora (angl. *sampling-based path planning*) se naključno zajemajo točke iz okolja (konfiguracije robota), nato se s pomočjo zaznavanja trka preverja, če le-te ležijo v prostem območju [4, 5]. Iz množice zajetih točk in povezav med njimi, ki prav tako v celoti ležijo v prostem območju, poiščemo pot med znano začetno in želeno ciljno točko.

Pri metodah vzorčenja prostora ni potreben izračun prostega konfiguracijskega prostora Q_{free}, ki pri kompleksni postavitvi ovir in visokih prostostnih stopnjah postane časovno zamuden, ampak ga predstavimo z naključnimi vzorci ter neodvisno od geometrije okolice najdemo rešitev za širok spekter problemov. Prav tako se izognemo velikemu številu celic, ki ga dobimo pri opisu z razcepom na celice, ter zamudni implementaciji in računanju, ki spremljata uporabo natančnega razcepa na celice. Zaradi vključitve stohastičnega mehanizma (naključni sprehod, angl. *random walk*) v nekatere algoritme, npr. v RPP (angl. *random path planner*), ko se iz točke, v kateri smo ujeti, rešimo s pomočjo premika po naključnih vzorcih iz prostega območja, močno omilimo problem lokalnih minimumov, ki nas pesti pri uporabi potencialnega polja.

Da bi zaznavanje trka zavzemalo čim manj računskega časa, ga preverjamo samo za ovire, ki so dovolj blizu, da bi lahko trčile z robotom. Robot in ovire so lahko omejeni z enostavnimi liki, tako da kompleksnejše preverjanje trka (med pravo obliko robota in oblikami ovir) izvajamo samo, ko se dva lika prekrivata. V tem primeru lahko trk zaznavamo na hierarhičen način, kjer večji lik, ki obkroža



Slika 4.16: Razdelitev večjega lika na dva manjša pri hierarhičnem zaznavanju trka robota

celotnega robota, zamenjamo z dvema manjšima, ki obkrožata vsak svojo polovico robota, kot je to prikazano na sliki 4.16. Če se kateri izmed likov prekriva z oviro, ga ponovno razdelimo na dva manjša ustrezna lika. S tem nadaljujemo dokler ne izključimo ali potrdimo trka oz. dosežemo želene resolucije.

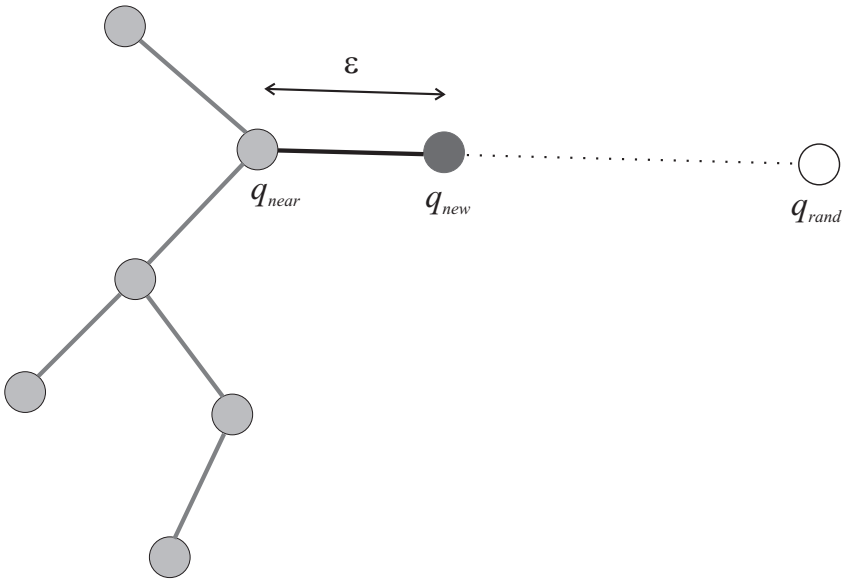
Tovrstne pristope delimo na tiste, ki so primerni za enkratno iskanje poti, in tiste, ki so primerni za večkratno iskanje poti. Pri prvih želimo čim hitreje poiskati pot med eno začetno in eno ciljno točko, zato se osredotočimo na dele okolice, ki obetajo rešitev. V drevesno strukturo sproti dodajajmo nove točke in povezave dokler ne najdemo rešitve. Pri drugih pristopih pa se pred samim načrtovanjem poti izvede enkraten postopek izdelave neusmerjenega grafa oz. zemljevida cest, ki predstavlja povezanost prostega območja in s pomočjo katerega lahko nato rešimo problem načrtovanja poti za več poljubnih parov začetnih in ciljnih točk. V nadaljevanju sta opisana predstavnika iz obeh skupin.

Metoda hitro-rastočega naključnega drevesa

Metoda hitro-rastočega naključnega drevesa (angl. *rapidly-exploring random tree*) je metoda iskanja poti med eno začetno in eno ciljno točko [4]. Metoda v vsaki iteraciji doda nove povezave v smeri od naključnih točk proti najbližjim točkam, ki so že v grafu (drevesu).

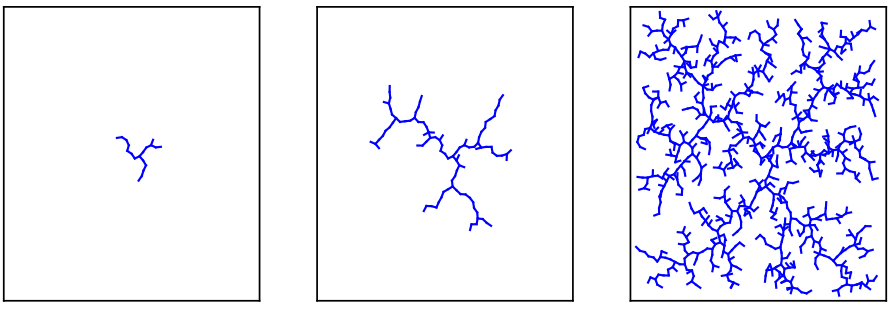
V prvi iteraciji algoritma začetna konfiguracija q_i predstavlja drevo (povezan graf). V vsaki naslednji iteraciji se naključno izbere konfiguracija q_{rand} in iz obstoječega grafa poišče najbližje vozlišče q_{near}. V smeri od q_{near} proti q_{rand} se na vnaprej določeni razdalji ε izračuna kandidat za novo vozlišče q_{new}. Če sta q_{new} in povezava od q_{near} do q_{new} v prostem območju, je q_{new} novo vozlišče in njegova povezava z q_{near} je dodana v graf. Postopek je prikazan na sliki 4.17.

Iskanje se zaključi po določenem številu iteracij (npr. sto iteracij) ali ko je dosežena določena verjetnost (npr. 10%). Takrat se namesto zajema novega



Slika 4.17: Metoda hitro-rastočega naključnega drevesa — razširitev grafa z novo točko q_{new} v smeri naključno vzorčene točke q_{rand}

naključnega vzorca izbere ciljna točka za katero se preveri, ali jo je mogoče povezati z grafom [6]. Takšno drevo se hitro razširi na neraziskana območja, kar lahko vidimo na sliki 4.18. Ta metoda ima samo dva parametra: velikost koraka ε in želena ločljivost ali število iteracij, ki določata pogoje za zaključek algoritma. Zato je vedenje algoritma hitro-rastočega naključnega drevesa dosledno in njegova analiza preprosta.



(a) (b) (c)

Slika 4.18: Drevo, zgrajeno z metodo hitro-rastočega naključnega drevesa, hitro napreduje v neraziskano prosto območje. Slike z leve proti desni prikazujejo drevesa z 20, 100 in 1000 vozlišči.

Primer 4.6

Izvedite algoritem hitro-rastočega naključnega drevesa, ki bo ustvaril drevo za dvodimenzionalno prosto območje, velikosti $10\,\mathrm{m} \times 10\,\mathrm{m}$, kjer je parameter $\varepsilon = 0{,}2\,\mathrm{m}$.

Rešitev

Za podobne rezultate, kot na sliki 4.18, lahko uporabite Matlab kodo iz programa 4.6.

Program 4.6

./src/pth/example_rrt.m

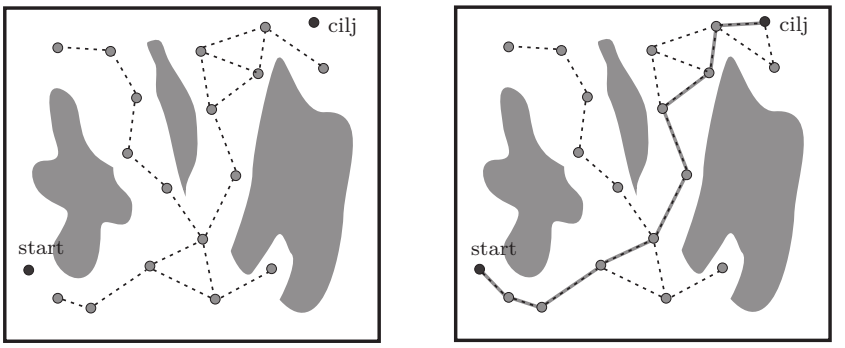
```
xi = [5, 5]; % Začetna konfiguracija
D = 0.2; % Razdalja do novega vozlišča
maxIter = 1000;
M = [xi]; % Zemljevid

j = 1;
while j < maxIter
    xRand = 10*rand(1,2); % Naključna konfiguracija
    dMin = 100; iMin = 1; % Iskanje najbližje točke v zemljevidu M
    for i = 1:size(M,1)
        d = norm(M(i,:)-xRand);
        if d<dMin
            dMin = d;
            iMin = i;
        end
    end

    xNear = M(iMin,:);
    v = xRand - xNear;
    xNew = xNear + v/norm(v)*D; % Izračun nove točke

    con = [xNear; xNew];
    M = [M; xNew];
    j = j + 1;

    line(con(:,1), con(:,2), 'Color', 'b');
end
```

Primer 4.7

Razširite primer 4.6, da vključuje tudi preproste ovire (npr. krožne ovire).

Rešitev

Predpostavite okolje z enostavnimi ovirami v obliki krogov z znanimi položaji in premeri. Preverite, ali v prostem območju ležita kandidat za novo vozlišče q_{new}



a) b)

Slika 4.19: Metoda verjetnostnega zemljevida poti: (a) faza učenja in (b) faza iskanja poti

in daljica, ki povezuje q_{near} in q_{new}.

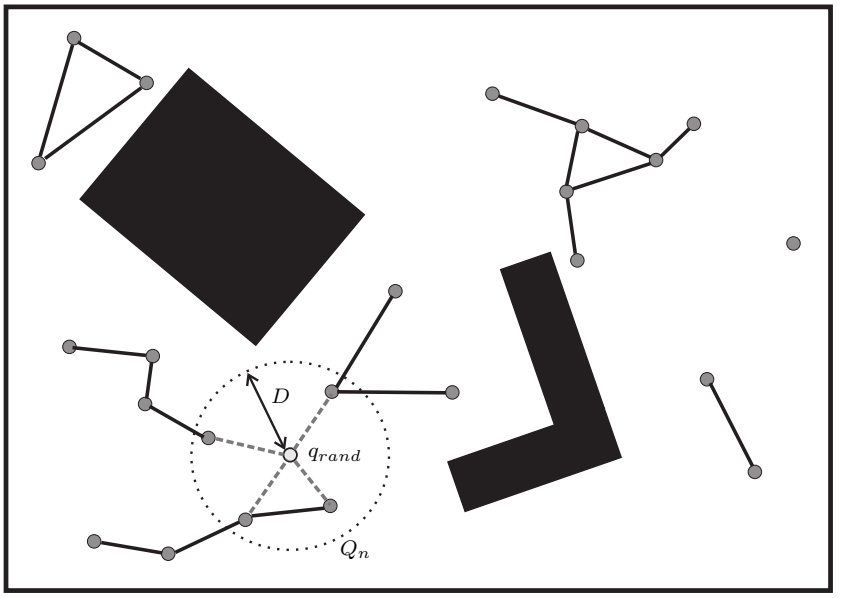
Metoda verjetnostnega zemljevida poti

Metoda verjetnostnega zemljevida poti (angl. *probabilistic roadmap*) je način iskanja poti med več začetnimi in več ciljnimi točkami, ki poteka v dveh fazah [7]. Prva je *faza učenja*, v kateri se izdela povezan zemljevid cest ali neusmerjen graf prostega območja (slika 4.19a), druga pa *faza iskanja*, v kateri se trenutni par začetne in ciljne točke poveže z grafom in se s pomočjo iskalnih algoritmov poišče pot (slika 4.19b).

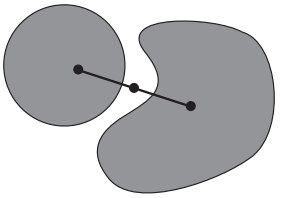
V fazi učenja izdelamo zemljevid cest, ki je na začetku prazna množica, potem pa se napolni z vozlišči s ponavljanjem v nadaljevanju naštetih korakov. Naključno izbrano konfiguracijo q_{rand}, ki leži v prostem območju, dodamo v zemljevid in določimo vozlišča Q_n za razširitev zemljevida. To lahko storimo tako, da izberemo K najbližjih sosednjih vozlišč (Q_n) ali pa vsa sosednja vozlišča, katerih oddaljenost od q_{rand} je manjša od vnaprej določenega parametra D (slika 4.20). V prvem oz. prvih korakih morda ne najdemo sosednjih vozlišč. Nato dodamo v zemljevid vse enostavne povezave od q_{rand} do vozlišč iz Q_n, ki v celoti ležijo v prostem območju. S tem postopkom nadaljujemo, dokler zemljevid ne vsebuje želenega števila vozlišč N.

V fazi iskanja začetno in ciljno točko preko prostega območja povežemo s čim bližjima možnima vozliščema iz zemljevida in nato z iskalnim algoritmom poiščemo pot med njima.

Za ti dve fazi ni nujno, da ju izvedemo ločeno. Lahko ju ponavljamo, dokler nimamo dovolj vozlišč za odkritje rešitve. Če ni možno najti rešitve, zemljevid razširimo z novimi vozlišči in povezavami, dokler ne dobimo izvedljive rešitve.



Slika 4.20: Pri metodi verjetnostnega zemljevida poti dodamo v zemljevid vse možne povezave od naključnih točk q_{rand} do sosednjih vozlišč Q_n



Slika 4.21: Pri preizkusu mostu sta izbrani dve naključni bližnji točki, ki določata daljico. Če se srednja točka nahaja v prostem območju, zunanji dve pa znotraj ovir, je srednja točka dodana kot vozliče na zemljevidu.

Na ta način se iterativno približujemo čim bolj ustrezni predstavitvi prostega območja.

Metoda je zelo učinkovita pri robotih z velikim številom prostostnih stopenj, vendar ima težave pri iskanju povezave med dvema območjema preko ozkih prehodov. To lahko premagamo z dodajanjem vozlišč s pomočjo **preizkusa mostu** (angl. *bridge test*), v katerem izberemo tri naključne "blizuležeče" točke na daljici (slika 4.21). Če sta krajni točki v trku z ovirami, srednja pa ne, potem srednjo točko vključimo v zemljevid kot vozlišče ter jo nato skušamo na enak način kot ostale povezati s sosednjimi vozlišči. Z združitvijo preizkusa mostu in enakomernega vzorčenja [8] v "hibridno strategijo vzorčenja" lahko dobimo manjše zemljevide poti, ki bolj učinkovito pokrivajo prosto območje ter ohranjajo dobre povezave preko ozkih prehodov.

Primer 4.8

Izvedite algoritem verjetnostnega zemljevida poti za dvodimenzionalno prosto območje, velikosti 10 m × 10 m. Poskusite razširiti algoritem za okolje z ovirami.

Rešitev

Rešitev je podana v programu 4.7.

Program 4.7

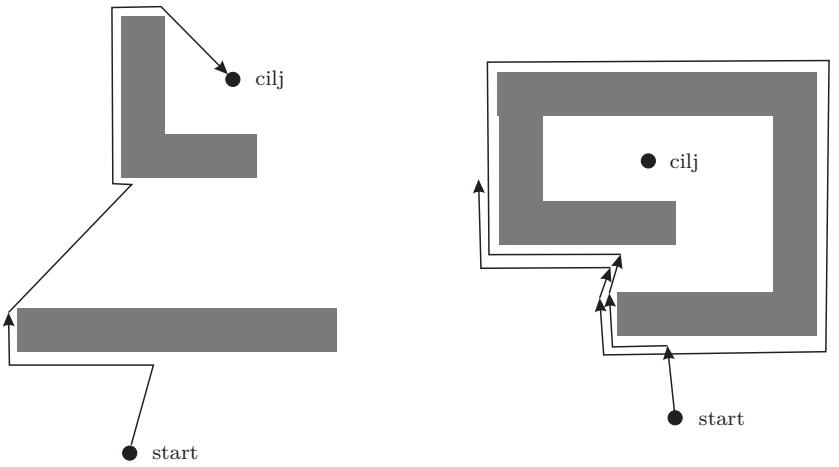
./src/pth/example_prm.m

```
D = 1; % Parameter razdalje
maxIter = 200;
M = []; % Zemljevid

j = 1;
while j <= maxIter
    xRand = 10*rand(1,2); % Naključna konfiguracija
    M = [M; xRand];
    con = []; % Povezave
    for i = 1:size(M,1) % Iskanje povezav do sosednjih vozlišč
        d = norm(M(i,:)-xRand);
        if d<D && d>eps % Dodajanje povezav od xRand do soseda
            con = [con; xRand, M(i,:)];
        end
    end
    j = j + 1;

    line(xRand(1), xRand(2), 'Color', 'r', 'Marker', '.');
    for i = 1:size(con,1)
        line(con(i,[1,3]), con(i,[2,4]), 'Color', 'b');
    end
end
```

4.3 Preprosti algoritmi načrtovanja poti — algoritmi tipa hrošč

Algoritmi tipa *hrošč* (angl. *bug algorithm*) so najbolj enostavni algoritmi planiranja poti. Za planiranje ne potrebujejo zemljevida okolice, zato so primerni, ko zemljevid okolice ni znan ali pa se okolica stalno spreminja in tudi ko ima mobilna platforma zelo omejeno moč računanja. Ti algoritmi uporabljajo le lokalno informacijo o okolju, pridobljeno iz senzorjev (npr. senzor razdalje), in globalno podan cilj, ne potrebujejo pa globalnega znanja v obliki zemljevida okolja. Njihovo delovanje sestoji iz dveh enostavnih vzorcev obnašanja: gibanje po ravni liniji proti cilju in sledenje obrisu ovire.

Mobilni roboti, ki uporabljajo te algoritme, se lahko izogibajo oviram in pre-



Slika 4.22: Algoritem Hroščo (varianta vedno zavij levo) uspešno najde pot do cilja (leva slika) in neuspešno (desna slika)

mikajo proti cilju. Tovrstni algoritmi imajo majhno porabo spomina, vendar je lahko najdena pot daleč od optimalne. Algoritmi tipa hrošč so bili najprej implementirani v [9], temu pa so sledile številne izboljšave kot v [10–12].

V nadaljevanju so predstavljeni trije osnovni algoritmi tipa hrošč.

4.3.1 Algoritem Hroščo

Algoritem Hroščo deluje v naslednjih dveh korakih:

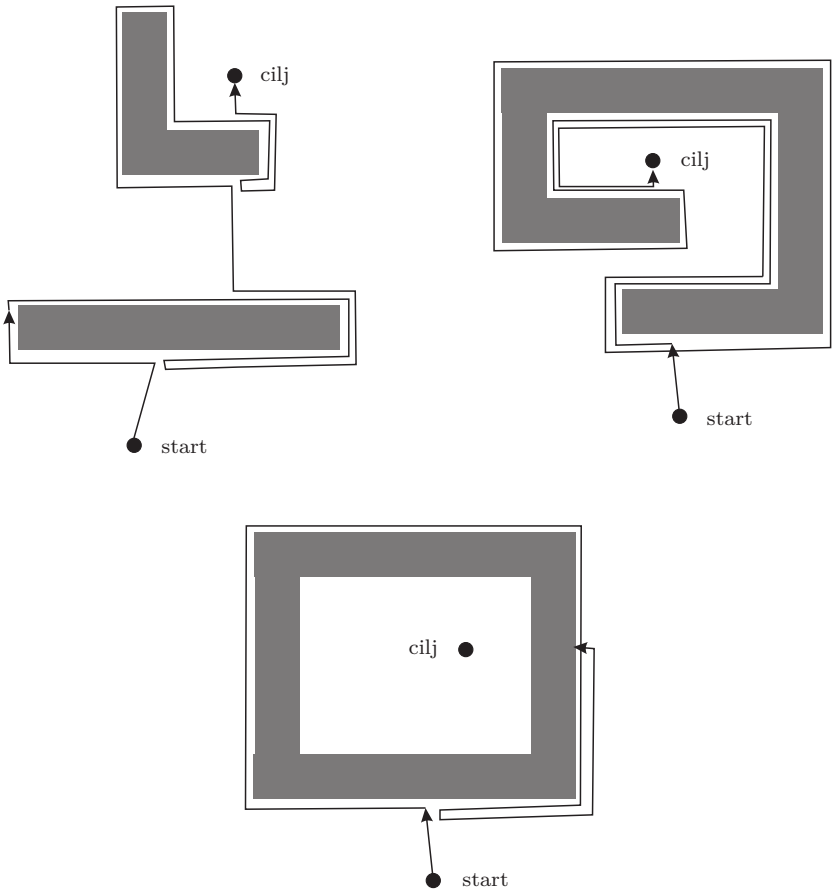
1. V ravni liniji se premika proti cilju, dokler ne naleti na oviro ali cilj.
2. Če naleti na oviro, vedno zavije levo (oz. vedno desno, če je tako določeno v algoritmu) in sledi obrisu ovire, dokler ni možno ponovno nadaljevati proti cilju.

Primer delovanja algoritma Hroščo je prikazan na sliki 4.22.

4.3.2 Algoritem Hroščı

Algoritem Hroščı uporablja glede na Hroščo nekaj več spomina in zahteva malo več računanja, saj v vsaki iteraciji izračuna evklidsko razdaljo do cilja in si zapomni najbližjo točko na obodu ovire do cilja. Njegovo delovanje podajata koraka:

1. V ravni liniji se premika proti cilju, dokler ne naleti na oviro ali cilj.
2. Če naleti na oviro, ob oviri zavije levo in sledi celotnemu obrisu ovire ter ves čas meri evklidsko razdaljo do cilja. Ko ponovno prispe do točke, kjer

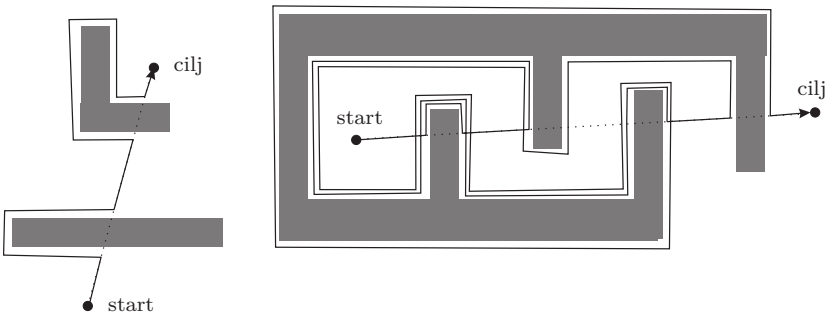


Slika 4.23: Algoritem Hrošč1 najde pot do cilja v obeh primerih na zgornjih slikah. V najslabšem primeru je njegova pot za $\frac{3}{2}$ obsega vseh ovir daljša od evklidske razdalje med začetno in ciljno točko. Algoritem zna ugotoviti, kdaj cilj ni dosegljiv (primer na spodnji sliki).

je naletel na oviro, gre po krajši poti ob obodu ovire do točke, ki je bila najbliže cilju. Nato gre po ravni liniji proti cilju.

Primer delovanja algoritma Hrošč1 je prikazan na sliki 4.23.

Dobljena pot ni optimalna; v najslabšem primeru je za $\frac{3}{2}$ obsega vseh ovir do cilja daljša kot evklidska razdalja med začetno in ciljno točko. Algoritem za vsako oviro, na katero naleti na poti do ciljne točke, najde samo eno točko naleta na oviro in samo eno točko, v kateri zapusti obod ovire. Tako na nobeno oviro ne naleti več kot enkrat in zaradi tega nikoli ne ustvari ciklov med istimi ovirami. Ko algoritem naleti na isto oviro več kot enkrat, je to znak, da je znotraj ovire ujeta ali začetna ali ciljna točka. Takrat se algoritem konča, saj ne obstaja nobena izvedljiva pot do cilja (spodnji primer na sliki 4.23).



Slika 4.24: Algoritem Hrošč2 sledi glavni liniji do cilja. Na isto oviro lahko naleti večkrat, zato pride do kroženja. Algoritem Hrošč2 zna prepoznati nedosegljiv cilj.

4.3.3 Algoritem Hrošč2

Algoritem Hrošč2 se vedno poskuša premikati po glavni liniji, tj. daljici, ki povezuje začetno in ciljno točko. Deluje s ponavljanjem naslednjih korakov:

1. Robot naj se premika po glavni liniji, dokler ne naleti na oviro ali ciljno točko. V ciljni točki se iskanje zaključi.

2. Če je robot naletel na oviro, sledi obodu ovire toliko časa, da doseže glavno linijo, kjer je evklidska razdalja do cilja manjša kot evklidska razdalja do točke, kjer je (zadnjič) naletel na oviro.

Čeprav se zdi algoritem Hrošč2 veliko bolj učinkovit kot Hrošč1 (leva slika 4.24), ne zagotavlja, da bo robot samo enkrat naletel na določeno oviro. Pri nekaterih postavitvah in oblikah ovir po prostoru lahko Hrošč2 dolgo časa po nepotrebnem kroži, preden prispe do ciljne točke, kar je prikazano na desni sliki 4.24. Algoritem lahko razbere, da ciljne točke ni možno doseči, če večkrat v isti točki naleti na isto oviro.

Iz primerjave algoritmov Hrošč1 in Hrošč2 lahko zaključimo sledeče:

- Hrošč1 je bolj temeljit algoritem iskanja, saj preišče vse možnosti pred izvršitvijo naslednjega koraka,

- Hrošč2 je požrešen algoritem, saj izbere prvo obetavno opcijo,

- v večini primerov je Hrošč2 bolj učinkovit kot Hrošč1, toda

- Hrošč1 ima bolj predvidljivo delovanje.

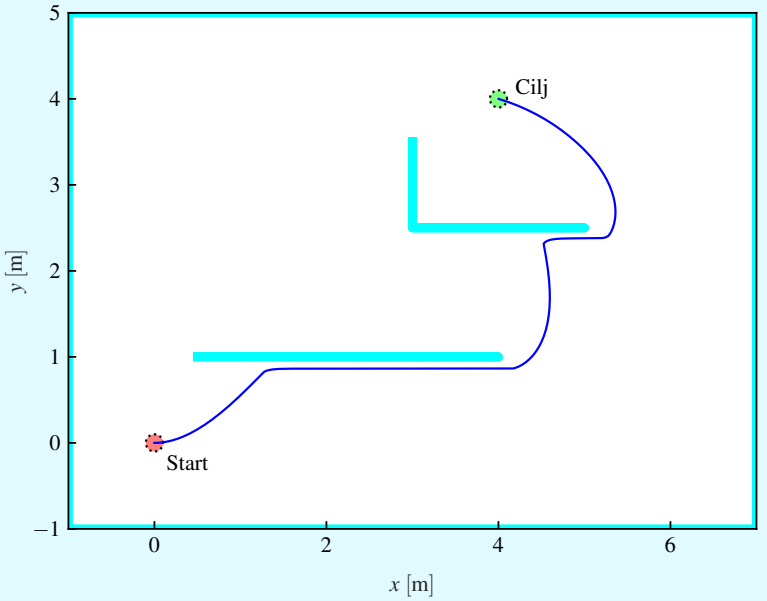
Primer 4.9

Izvedite načrtovanje poti z algoritmom Hrošč0 za mobilnega robota z diferencialnim pogonom. Predpostavite, da zemljevid okolja ni na voljo in robot pozna samo svojo trenutno lego, ciljno točko in trenutno razdaljo do cilja (meritev senzorja).

Glede na algoritem Hrošč0 bi moral robot zapeljati proti cilju, če je dovolj oddaljen od katerekoli ovire (npr. več kot 0.2 m), in slediti oviri, če je blizu ovire. Napišite kodo svoje implementacije algoritma s pomočjo programa 4.8, ki že omogoča simulacijo gibanja robota in meritev senzorjev. Okolje in primer pridobljene poti robota sta prikazana na sliki 4.25.

Program 4.8
./src/pth/example_bug0.m

```
Ts = 0.03; % Računski korak
t = 0:Ts:30; % Čas simulacije
q = [0; 0; 0];% Začetna lega
goal = [4; 4]; % Ciljna lega
% Ovire
obstacles{1} = flipud([-1 -1; 7 -1; 7 5; -1 5]);
obstacles{2} = [0.5 1; 4 1];
obstacles{3} = [3 3.5; 3 2.5; 5 2.5; 3 2.5];
obst = [];
for i = 1:length(obstacles)
    obst = [obst; obstacles{i}([1:end,1],:); nan(1,2)];
end

for k = 1:length(t)
    % Razdalja do najbližje ovire in usmeritev daljice
    [dObst, ~, z] = nearestSegment(q(1:2).', obst);
    phiObst = atan2(obst(z+1,2)-obst(z,2), obst(z+1,1)-obst(z,1));

    % Sem pride regulacijski algoritem ...

    % Simulacija gibanja robota
    dq = [v*cos(q(3)); v*sin(q(3)); w];
    noise = 0.00; % Parameter za nastavljanje šuma (npr. 0.001)
    q = q + Ts*dq + randn(3,1)*noise; % Eulerjeva integracija
    q(3) = wrapToPi(q(3)); % Zapis kota v območju [-pi, pi]
end
```



Slika 4.25: Načrtovanje poti in vodenje z algoritmom Hrošč0 za vožnjo robota do cilja ob izogibanju oviram

Rešitev

Možna implementacija rešitve, ki ustvari pot na sliki 4.25, je podana v programu 4.9 — kodo vstavite v označeno vrstico programa 4.8. Oblika dobljene poti je odvisna tudi od uporabljenega regulacijskega algoritma za vodenje robota.

Program 4.9

./src/pth/script_bug0.m

```
% Regulacija na podlagi razdalje do ovire
if dObst>0.2 % Vožnja proti cilju
    phiRef = atan2(goal(2)-q(2), goal(1)-q(1));
    ePhi = wrapToPi(phiRef - q(3));
    dGoal = sqrt(sum((goal-q(1:2)).^2));
    g = [dGoal/2, 1]; % Ojačenji regulatorja
else % Vožnja okoli ovire po desni
    phiRef = wrapToPi(phiObst + pi*0);  % Prištejte pi za vožnjo po levi
    ePhi = wrapToPi(phiRef-q(3));
    g = [0.4, 5]; % Ojačenji regulatorja
end
% Enostavni regulator za diferencialni pogon
v = g(1)*abs(cos(ePhi));
w = g(2)*ePhi;
v = min([v, 0.5]);
```

Primer 4.10

Z razširitvijo primera 4.9 izvedite tudi algoritma Hrošč1 in Hrošč2.

Rešitev

Za implementacijo algoritma Hrošč1 lahko prilagodite program 4.9 iz primera 4.9, kjer je glavno vedenje sestavljeno iz dveh delov. Prvo vedenje (vožnja proti cilju) ostaja nespremenjeno, drugo pa je treba spremeniti. Shranite začetni položaj, kjer robot najprej zazna oviro. Vozite se okoli ovire in izmerite razdaljo do cilja ter si zapomnite najbližjo točko. To izvajajte, dokler robot ne pride v shranjeni začetni položaj ali vsaj dovolj blizu. Vrnite se na najbližjo zapomnjeno točko.

Podobno lahko program 4.9 iz primera 4.25 prilagodite algoritmu Hrošč2. Shranite začetni položaj, kjer robot najprej zazna oviro. Robot naj kroži okoli ovire, dokler ne prečka glavne linije. Če je točka prečkanja bližje cilju kot izhodiščni točki, pelje proti cilju; v nasprotnem še naprej kroži okoli ovire.

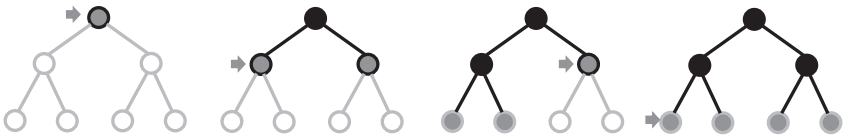
4.4 Metode iskanja poti v grafu

Ko imamo okolje z ovirami ustrezno predstavljeno z grafom (npr. prostor stanj, razcep na celice, zemljevid cest), lahko uporabimo enega izmed algoritmov, ki poiščejo pot od začetne do ciljne konfiguracije. V nadaljevanju je podanih nekaj znanih algoritmov iskanja poti v grafu.

V splošnem začnemo iskanje tako, da preverimo, ali je začetno vozlišče hkrati tudi ciljno vozlišče. Ponavadi to ne drži, zato razširimo iskanje na vozlišča, ki sledijo sedanjemu vozlišču. Na podlagi izbranega algoritma (in vrednosti cenilke) izberemo enega izmed sosednjih vozlišč. Če izbrano vozlišče ni ciljno, raziskujemo naslednja vozlišča, ki sledijo temu novemu vozlišču. Postopek nadaljujemo, dokler ne najdemo rešitve ali dokler ne preiščemo celotnega grafa.

Pri iskanju v grafu vodimo sezname vozlišč, ki smo jih med iskanjem že obiskali, s čimer preprečimo, da bi večkrat obiskali isto vozlišče. Tako imenovana živa vozlišča, iz katerih lahko nadaljujemo iskanje, shranimo na **seznam odprtih vozlišč**. Mrtva vozlišča nimajo naslednikov ali pa smo jih že preverili in jih shranimo na **seznam zaprtih vozlišč**.

Od strategije algoritma je odvisno, po kakšnem zaporedju izbiramo vozlišča za razširitev območja iskanja. Seznam odprtih vozlišč razvrstimo glede na določen kriterij in ob izbiranju naslednjega vozlišča za razširitev iskanja vzamemo prvo s seznama, torej tisto, ki najbolj ustreza razvrščevalnemu kriteriju (ima najmanjšo vrednost glede na kriterij).



Slika 4.26: Algoritem iskanja v širino najprej razišče najbližja vozlišča. Trenutno vozlišče je označeno s puščico, odprta (živa) vozlišča so označena s svetlo sivimi, zaprta (mrtva) vozlišča s črnimi in nepreverjena vozlišča z belimi krogi.

Na začetku je na seznamu odprtih vozlišč Q samo začetno vozlišče. Izračunamo vozlišča, ki sledijo začetnemu, in jih zapišemo na seznam odprtih vozlišč, začetno vozlišče pa damo na seznam zaprtih vozlišč. Nato iskanje razširimo s prvim vozliščem na odprtem seznamu tako, da izračunamo njegove naslednike. Tako so na odprtem seznamu preostali nasledniki (razen prvega že raziskanega vozlišča) in pravkar izračunani nasledniki prej izbranega vozlišča. Postopek je prikazan na sliki 4.26, kjer so odprta vozlišča prikazana z svetlo sivimi krogi in lahko vidimo, zakaj se ta vozlišča med iskanjem v drevesni strukturi imenujejo listi. Zaprta vozlišča so prikazana s črnimi krogi, nepreverjena vozlišča pa z belimi.

Ločimo **neinformirane** in **informirane** algoritme iskanja po grafu. Neinformirani algoritmi posedujejo samo informacije, ki so podane z definicijo problema (slepo iskanje po stanjih oz. vozliščih). Graf pregledujejo sistematično in ne razlikujejo med bolj ali manj obetavnimi vozlišči. Informirano ali hevristično iskanje vsebuje dodatne informacije o vozliščih, zato je zmožen razlikovati med bolj in manj obetavnimi vozlišči in posledično je lahko algoritem učinkovitejši.

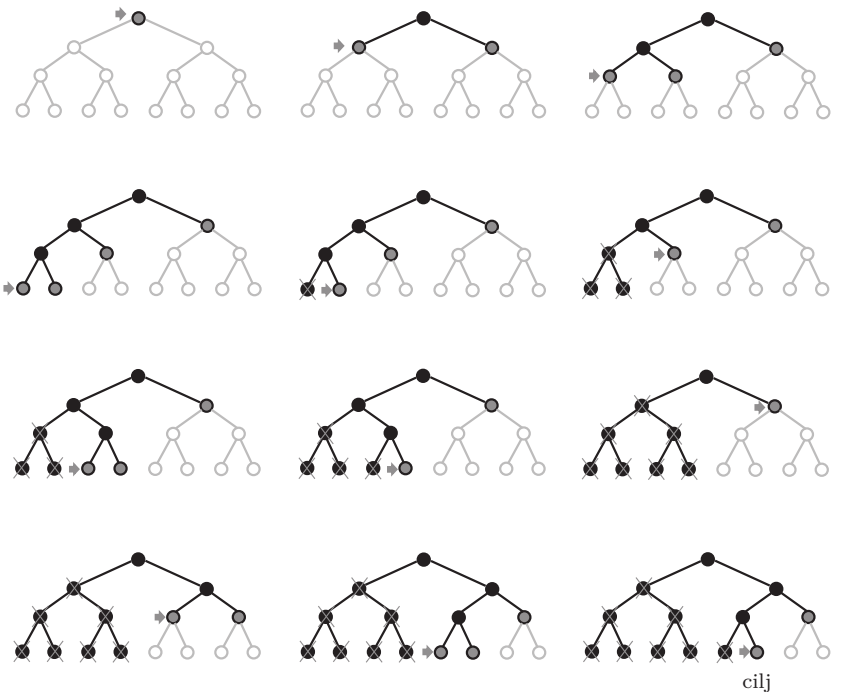
4.4.1 Iskanje v širino

Iskanje v širino (BFS, angl. *breadth-first search*) je algoritem neinformiranega iskanja. Najprej raziščemo najbolj plitva vozlišča, torej vozlišča, ki so najbliže začetnemu vozlišču. Vsa vozlišča, do katerih lahko dostopamo v k korakih, obiščemo prej kot katerokoli vozlišče, do katerega pridemo v $k + 1$ korakih, kar je prikazano na sliki 4.26.

Seznam odprtih vozlišč Q razvrščamo po metodi *prvi noter, prvi ven* (FIFO, angl. *first in, first out*): na novo odprta vozlišča dodajamo na konec seznama Q, vozlišča za razširjanje iskanja pa jemljemo z začetka seznama.

Algoritem je **popoln**, saj pri končnem faktorju razvejitve najde rešitev, če le-ta obstaja. Če pa je možnih več rešitev, najde tisto, ki je najmanj korakov oddaljena od začetnega vozlišča. To ne pomeni, da je najdena rešitev hkrati tudi optimalna, saj ni nujno, da imajo vsi prehodi med vozlišči enako ceno.

Poraba spomina in računski čas sta pri tej metodi velika, saj z razvejanostjo drevesa eksponentno naraščata.



Slika 4.27: Iskanje v globino ima majhno porabo spomina, saj hranimo le liste (svetlo siva) in na poti razširjena vozlišča (temno siva). Prečrtana vozlišča so odstranjena iz spomina.

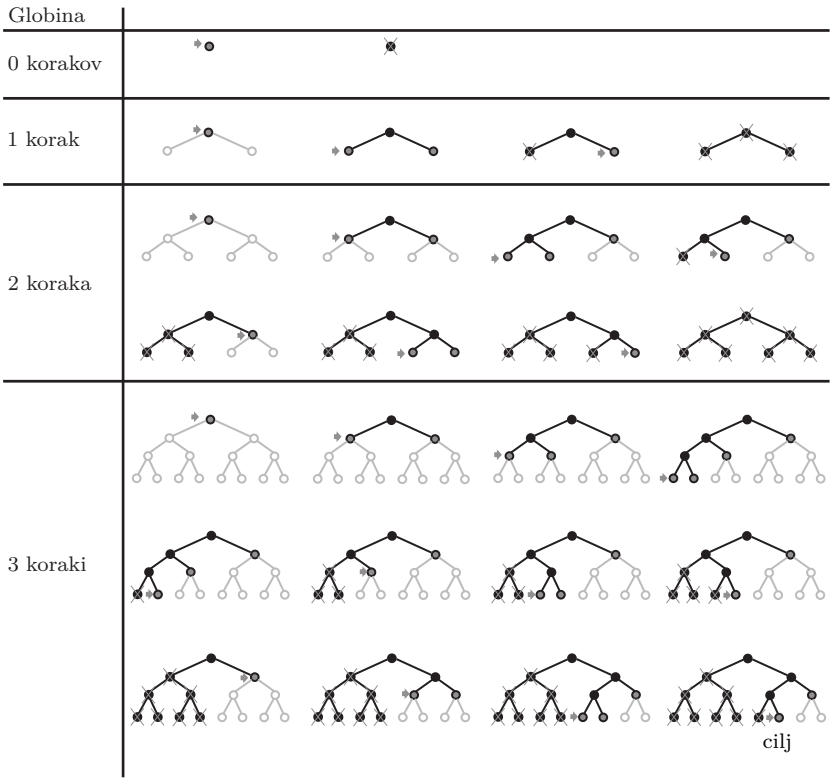
4.4.2 Iskanje v globino

Iskanje v globino (DFS, angl. *depth-first search*) je algoritem neinformiranega iskanja, kjer iščemo v globino. Vozlišče, ki je najbolj oddaljeno od začetnega, razširimo in iskanje na ta način nadaljujemo v globino, dokler neko vozlišče nima več nobenih naslednikov. Takrat iskanje nadaljujemo z naslednjim najglobljim vozliščem, katerega nasledniki še niso bili raziskani, kot je to prikazano na sliki 4.27

Seznam odprtih vozlišč Q obravnavamo kot *sklad* (angl. *stack*), ki ga razvrščamo po metodi *zadnji noter, prvi ven* (angl. *last in, first out – LIFO*): na novo odprta vozlišča dodajamo na začetek seznama Q, od koder tudi jemljemo vozlišča za razširjanje iskanja.

Algoritem iskanja v globino ni popoln, saj bi v primeru neomejene globine (neskončno število vej, ki se ne končajo) neskončno časa raziskoval samo eno vejo grafa. Temu se lahko izognemo tako, da iskanje omejimo do določene globine, kjer pa je možno, da ima rešitev večjo globino in jo zato algoritem ne najde. Prav tako algoritem ni optimalen, saj najdena pot ni nujno tudi najkrajša.

Iskanje v globino ima majhno porabo spomina, saj mora hraniti samo pot od začetnega do trenutnega vozlišča in vmesna neraziskana vozlišča, s katerimi še



Slika 4.28: Prikaz iterativnega poglabljanja iskanja v globino do globine treh korakov. Prečrtana vozlišča so odstranjena iz spomina. Algoritem se zaključi v tretjem koraku, kjer najde ciljno vozlišče.

nismo nadaljevali iskanja. Ko raziščemo neko vozlišče in vse njegove naslednike, lahko to vozlišče prenehamo hraniti v spominu.

4.4.3 Iterativno iskanje v globino

Iterativno iskanje v globino (IDDFS, angl. *iterative deepening depth-first search*) združuje prednosti iskanja v širino in iskanja v globino. Algoritem po korakih veča globino, do katere raziskujemo z iskanjem v globino, dokler ne najdemo ciljnega vozlišča. Najprej izvedemo iskanje v globino za vozlišča oddaljena nič korakov od začetnega. V primeru, da ne najdemo ciljnega vozlišča, iskanje v globino ponovimo za vozlišča, oddaljena en korak od začetnega vozlišča itd. Tako se iskanje izvaja tudi v širino.

Algoritem ima majhno porabo spomina, je popoln, saj najde rešitev, če le-ta obstaja, in optimalen, ker najde najkrajšo pot, če so vse cene prehodov enake ali ne naraščajo z globino vozlišča. Če imajo vsa vozlišča približno enak faktor razvejitve tudi večkratno računanje stanj ni pretirano potratno, saj je večina vozlišč na dnu drevesa in jih algoritem obišče le občasno.

4.4.4 Dijkstrov algoritem

Dijkstrov algoritem je neinformiran algoritem za iskanje najkrajše poti od enega začetnega vozlišča do vseh preostalih vozlišč v grafu [13]. Prvotno ga je zasnoval Edsger Dijkstra [14], kasneje pa so ga razširili z raznimi prilagoditvami. V primeru iskanja poti med eno začetno in eno ciljno točko deluje neučinkovito, zaradi izračunavanja optimalnih poti do vseh vozlišč, zato lahko določimo, da se konča, ko izračuna najkrajšo želeno pot.

Algoritem zagotovo najde najkrajšo pot od začetnega do ciljnega vozlišča, saj temelji na računanju cene poti od začetnega do trenutnega vozlišča, ki jo imenujmo *cena-do-sem*.

Ceno poti do trenutno obravnavanega vozlišča izračunamo kot vsoto cene celotne poti do vozlišča, iz katerega smo prišli do trenutnega vozlišča, in cene povezave med njima. V primeru več najkrajših poti (z isto ceno) algoritem vrne eno in ni pomembno katero.

Za izvajanje algoritma je potrebno označiti povezave med vozlišči in jim določiti ceno. Za vsako obiskano vozlišče shranjujemo ceno trenutno najkrajše poti do njega (*cena-do-sem*) in povezave, po kateri smo prišli do trenutnega vozlišča. Pri iskanju vodimo tudi seznam odprtih in zaprtih vozlišč.

Na začetku je v seznamu odprtih vozlišč samo začetno vozlišče, katerega *cena-do-sem* je nič in brez predhodne povezave (ni prejšnjega vozlišča). Seznam zaprtih vozlišč je prazen. Nato ponavljamo naslednje korake, ki so ilustrirani na sliki 4.29

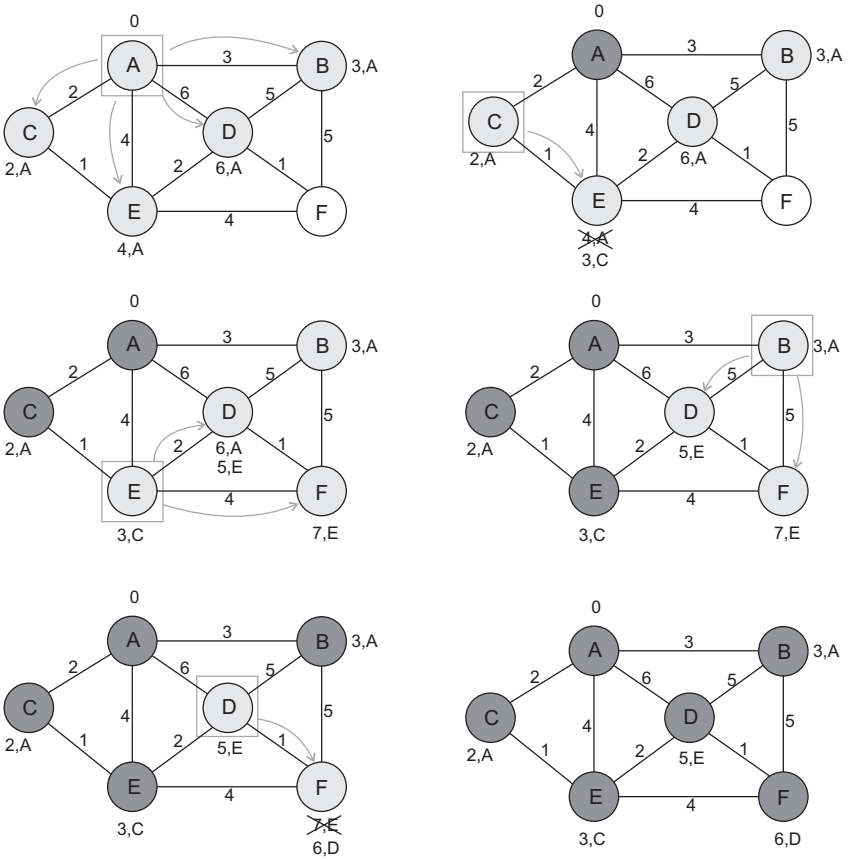
1. Iz seznama odprtih vozlišč vzamemo prvo vozlišče; to naj bo trenutno vozlišče. Seznam naj bo urejen naraščajoče glede na *ceno-do-sem*, kjer je prvo vozlišče tisto z najmanjšo *ceno-do-sem*.

2. Vsem vozliščem, do katerih lahko pridemo iz trenutnega vozlišča in niso na seznamu zaprtih vozlišč, izračunamo *ceno-do-sem* kot vsoto *cene-do-sem* trenutnega vozlišča in cene vmesne povezave.

3. Izračunamo in shranimo *ceno-do-sem* ter povezavo do trenutnega vozlišča za vsa vozlišča, ki še nimajo shranjenih teh informacij.

4. Če je v prejšnjem koraku katero izmed teh vozlišč že imelo shranjeno *ceno-do-sem* in ustrezno povezavo v grafu iz katere od prejšnjih iteracij, ti dve ceni primerjamo in kot končen podatek shranimo manjšo ceno in ustrezno povezavo.

5. Vozlišča dodamo na seznam odprtih vozlišč in ga uredimo po naraščajoči vrednosti *cene-do-sem*. Takšen seznam, ki ga imenujemo *vrsta s prednostjo* (angl. *priority queue*), omogoča, da hitreje najdemo vozlišče z najmanjšo vrednostjo *cene-do-sem* kot pa z iskanjem po neurejenem seznamu. Trenutno vozlišče premaknemo na seznam zaprtih vozlišč.



Slika 4.29: Dijkstrov algoritem za iskanje najkrajše poti med začetnim vozliščem A in ostalimi vozlišči. Trenutno oglišče je označeno s sivim kvadratom, njegovi nasledniki pa s puščicami. Cene poti so označene ob povezavah. Ob vozliščih je označena *cena-do-sem* in povezava do predhodnega vozlišča. Odprta vozlišča so označena s svetlo sivo, zaprta vozlišča pa s temno sivo. **Primer:** Zanima nas najkrajša pot med vozliščema A in F. Vidimo, da je $Cena_{F-D-E-C-A} = 6$, pot pa gre skozi vozlišča $A \to C \to E \to D \to F$.

V osnovni verziji naj bi se Dijkstrov algoritem končal, ko je seznam odprtih vozlišč prazen in rezultat vsebuje najkrajše poti iz začetnega v vsa ostala vozlišča. Če potrebujemo samo najkrajšo pot od začetnega do končnega vozlišča, iskanje zaključimo, ko dodamo ciljno vozlišče na seznam zaprtih vozlišč.

Ustrezno pot dobimo tako, da vzvratno sledimo in izpišemo po katerih povezavah smo prišli do ciljnega vozlišča, gledamo torej povezave v smeri od ciljnega do začetnega vozlišča. Najprej pogledamo povezavo od ciljnega vozlišča do predhodnega, nato sledimo povezavi do predhodega vozlišča, kjer prav tako preberemo in izpišemo po kateri povezavi smo prišli do tja. S tem nadaljujemo, dokler ne dosežemo začetnega vozlišča. Na koncu seznam izpisanih povezav le še obrnemo.

Dijkstrov algoritem je popoln (če pot obstaja, jo najde) in optimalen (najdena pot je najkrajša), če so vse uteži (cene) povezav večje od nič.

4.4.5 Algoritem A*

A* (prebrano kot *a zvezdica* ali *a star*) je informiran algoritem iskanja, saj vsebuje dodatno informacijo ali hevristiko. *Hevristika* je ocena cene poti od trenutnega vozlišča do cilja, zaradi česar lahko algoritem razlikuje med bolj ali manj obetavnimi vozlišči in je učinkovitejši pri iskanju rešitve. Algoritem za vsako vozlišče izračuna vrednost izbrane hevristične funkcije, ki predstavlja oceno cene, potrebne za pot od tega vozlišča do cilja, imenovana *cena-do-cilja*. Hevristična funkcija je lahko evklidska razdalja, razdalja Manhattan (vsota premikov v navpični in vodoravni smeri) ali kakšna druga primerna funkcija.

Tekom izvajanja algoritem za vsako vozlišče računa *ceno-celotne-poti* tako, da za določeno vozlišče sešteje *ceno-do-sem* in *ceno-do-cilja*. Hkrati vodi tudi seznama odprtih in zaprtih vozlišč.

Algoritem deluje tako, da je na začetku v seznamu odprtih vozlišč samo začetno vozlišče, katerega *cena-do-sem* je nič in nima predhodne povezave, seznam zaprtih vozlišč pa je prazen. V nadaljevanju se ponavljajo naslednji koraki, ki so ilustrirani na sliki 4.30:

1. Iz seznama odprtih vozlišč vzamemo prvo vozlišče, imenovano trenutno vozlišče. Seznam je urejen naraščajoče glede na *ceno-celotne-poti*, kjer ima prvo vozlišče najmanjšo *ceno-celotne-poti*.
2. Vsem vozliščem, do katerih lahko pridemo iz trenutnega vozlišča, izračunamo
   - *ceno-do-cilja*,
   - *ceno-do-sem* kot vsoto *cene-do-sem* trenutnega vozlišča in vmesne povezave ter
   - *ceno-celotne-poti* kot vsoto *cene-do-sem* in *cene-do-cilja*.

3. Za vsako izmed teh vozlišč, ki še nima shranjene izračunane *cene-do-sem*, ustrezne vmesne povezave od trenutnega vozlišča, *cene-do-cilja* in *cene-celotne-poti*, shranimo vse naštete vrednosti.

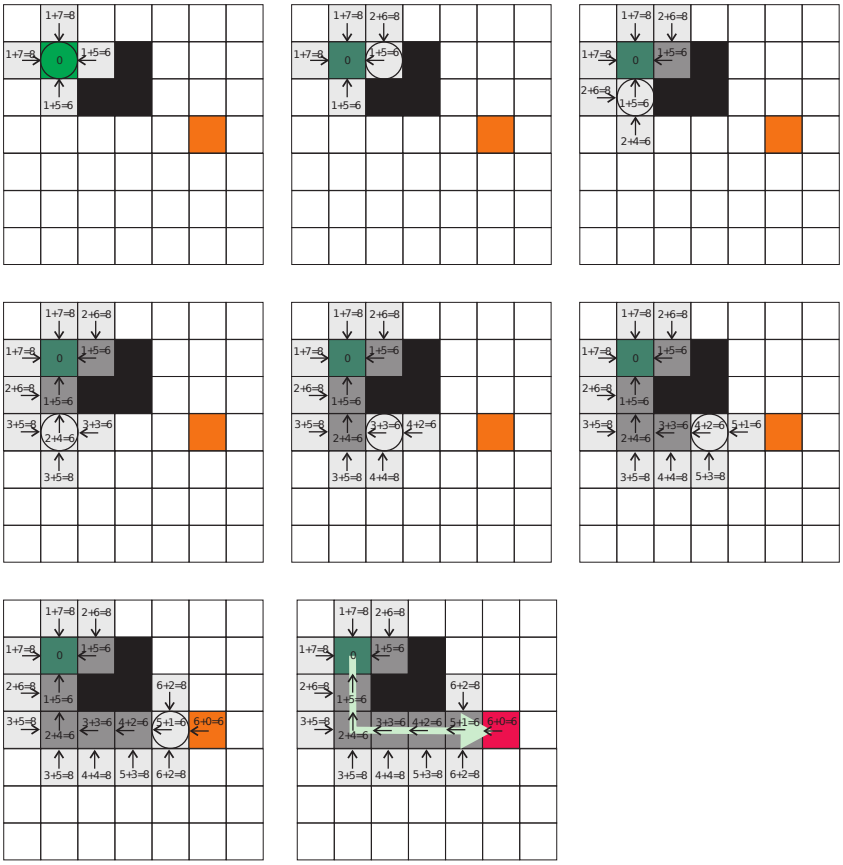
4. Če je v prejšnjem koraku katero izmed teh vozlišč že imelo shranjene izračunane vrednosti iz katere od prejšnjih iteracij, primerjamo obe *ceni-do-sem* in kot končen podatek shranimo manjšo skupaj s pripadajočo povezavo in ustrezno *ceno-celotne-poti*.

5. Vozlišča, katerih vrednosti smo računali prvič, dodamo na seznam odprtih vozlišč. Vozlišča, ki smo jim posodobili vrednosti in so že bila na seznamu odprtih vozlišč, jih tam tudi obdržimo. Vozlišča, ki so bila na seznamu zaprtih vozlišč in katerih vrednosti smo posodobili (do njega smo našli pot z manjšo *ceno-do-sem*), premaknemo na seznam odprtih vozlišč, ki ga nato uredimo po naraščajoči vrednosti *cene-celotne-poti*. Trenutno vozlišče premaknemo na seznam zaprtih vozlišč.

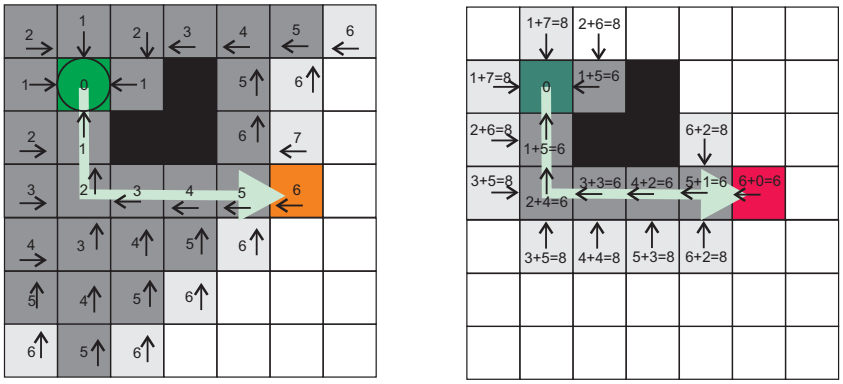
Na prvi sliki 4.30 je trenutno vozlišče hkrati tudi začetno ter so izbrana njegova naslednja vozlišča, ki so dosegljiva iz štirih smeri: levo, desno, gor in dol. Za vsa ta naslednja vozlišča je *cena-do-sem* enaka 1, saj so samo en korak stran od začetnega vozlišča, *ceno-do-cilja* pa določa razdalja Manhattan (hevristika) iz naslednjega do ciljnega vozlišča, ki jo lahko izmerimo preko ovire. Vsota obeh cen je *cena-celotne-poti*. Vozlišča, ki sledijo, imajo s puščico označeno povezavo do trenutnega vozlišča (označeno s krogom). Odprti seznam vozlišč tako vsebuje ta štiri naslednja vozlišča, zaprti seznam vozlišč pa vsebuje samo začetno vozlišče. Na drugi sliki 4.30 iz odprtega seznama izberemo za trenutno vozlišče tisto, ki ima najmanjšo *ceno-celotne-poti* (v tem primeru je enaka 6). Trenutno vozlišče ima samo enega naslednika, saj druge celice blokira ovira ali pa se nahajajo na zaprtem seznamu. *Cena-do-sem* za naslednje vozlišče je 2, ker se nahaja dva koraka stran (razdalja Manhattan) od začetnega vozlišča, in *cena-do-cilja* je 8. Trenutno vozlišče premaknemo na zaprti seznam, naslednje vozlišče pa na odprti seznam. Na tretji sliki 4.30 izberemo trenutno vozlišče kot vozlišče iz odprtega seznama, ki ima najmanjšo *ceno-celotne-poti* (v našem primeru je 6). Algoritem ponavlja korake dokler ne doseže ciljnega vozlišča.

Algoritem A* zagotavlja optimalnost najdene poti v grafu, v kolikor je hevristika (*cena-do-cilja*) optimistična, kar pomeni, da je *cena-do-cilja* za vsako vozlišče manjša ali kvečjemu enaka pravi *ceni-do-cilja*. Algoritem zaključimo, ko dodamo ciljno vozlišče na seznam zaprtih vozlišč.

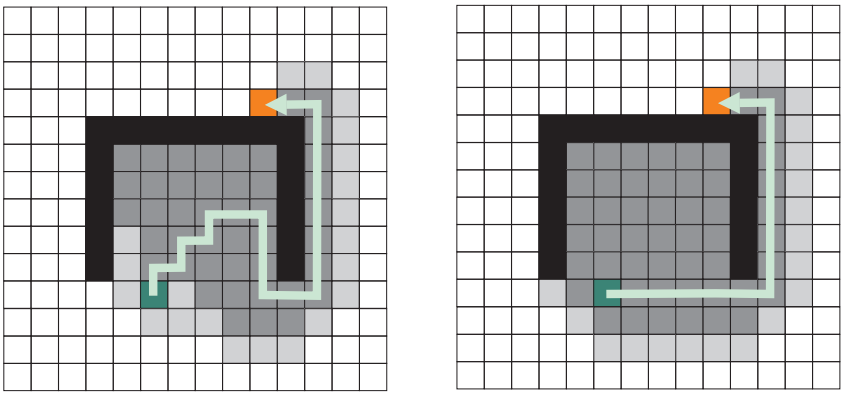
Algoritem A* je popoln, saj vedno najde pot, če le-ta obstaja, pri uporabi optimistične hevristike pa je tudi optimalen. Slabost A* je velika poraba spomina. V primeru, da so vse *cene-do-cilja* enake nič, je delovanje algoritma A* enako Dijkstrovemu. Na sliki 4.31 je prikazana primerjava delovanja Dijkstrovega algoritma in A*.



Slika 4.30: Trenutno vozlišče je označeno s krogom. Začetno vozlišče je zeleno, končno vozlišče oranžno, odprta vozlišča so svetlo siva, zaprta vozlišča temno siva in ovire so črne. Možne so štiri smeri prehodov: levo, desno, gor in dol. V vsaki celici (vozlišče) je označena smer do trenutnega vozlišča in cena poti, ki je vsota *cene-do-sem* in *cene-do-cilja*. Za izračun cene je uporabljena razdalja Manhattan. Najdeno pot razberemo s sledenjem povezavam, označene so s puščicami.



Slika 4.31: Primerjava algoritmov iskanja poti: Dijkstrov (levo) in A* (desno). Oba najdeta najkrajšo pot do cilja, vendar A* porabi precej manj iteracij zaradi uporabe dodatne informacije (hevristike) pri iskanju poti.



Slika 4.32: Primerjava algoritmov iskanja poti: Pohlepno iskanje *najprej-najboljši* (levo) in A* (desno). Pot, ki jo najdemo s pohlepnim iskanjem na način najprej najboljši, ni vedno optimalna. Odprta vozlišča so označena s svetlo sivo, zaprta vozlišča pa s temno sivo.

4.4.6 Pohlepno iskanje *najprej najboljši*

Pohlepno iskanje na način *najprej najboljši* (GBFS, angl. *greedy best-first search*) je informiran oz. hevristično algoritem. Seznam odprtih vozlišč razvrščamo po naraščajoči *ceni-do-cilja*. Tako v vsaki iteraciji razširimo iskanje na tisto odprto vozlišče, ki je najbliže cilju (ima najmanjšo *ceno do cilja*), saj predvidevamo, da bomo tako najhitreje dosegli cilj. Vendar najdena celotna pot ni nujno optimalna (najkrajša), kot prikazuje slika 4.32, saj algoritem upošteva le ceno od trenutnega do ciljnega vozlišča in ga ne zanima cena do trenutnega vozlišča. Posledično tudi ni pomembno, ali je hevristika optimistična ali ne, kot je bilo pomembno pri algoritmu A*.

Primer 4.11

Izvedite načrtovanje poti z algoritmom A* v okolju na sliki 4.32. Primerjajte dobljene rezultate z rezultati na sliki 4.32. Za izračun razdalje uporabite razdaljo Manhattan, medtem ko za hevristiko (*cena-do-cilja*) uporabite razdaljo Manhattan ali evklidsko razdaljo ter tudi primerjajte njune rezultate.

Nato prilagodite kodo za pohlepno iskanje *najprej najboljši*.

Rešitev

V splošnem je zaželena uporaba algoritma A*. Možna izvedba algoritma je prikazana v programu 4.10, kjer smo za izračun *cene-do-sem* uporabili razdaljo Manhattan, za izračun hevristike pa evklidsko razdaljo.

Program 4.10: Implementacija algoritma A*

./src/pth/AStarBase.m

```
classdef AStarBase < handle
properties
    map = []; % Zemljevid: 0 - prosto, 1 - ovira
    open = []; closed = []; start = []; goal = []; act = []; path = [];
end

methods
    function path = find(obj, start, goal) % start=[is; js], goal=[ig; jg]
        obj.start = start; obj.goal = goal; obj.path = [];
        obj.closed = []; % Prazen zaprti seznam
        obj.open = struct('id', start, 'src', [0; 0], 'g', 0, ...
                          'h', obj.heuristic(start)); % Začetni odprti seznam

        if obj.map(start(1), start(2))~=0 || obj.map(goal(1), goal(2))~=0
            path = []; return; % Pot ne obstaja!
        end

        while true % Iskanje
            if isempty(obj.open), break; end % Pot ni bila najdena :(

            obj.act = obj.open(1); % Vozlišče z urejenega odprtega seznama
            obj.closed = [obj.closed, obj.act]; % damo na zaprti seznam
            obj.open = obj.open(2:end); % in ga odstranimo z odprtega seznama.

            if obj.act.id(1)==obj.goal(1) && obj.act.id(2)==obj.goal(2)
                % Pot obstaja :) Poiščemo pot s pomočjo zaprtega seznama ...
                p = obj.act.id; obj.path = [p]; ids = [obj.closed.id];
                while sum(abs(p-start))~=0 % Sledimo staršem do starta
                    p = obj.closed(ids(1,:)==p(1) & ids(2,:)==p(2)).src;
                    obj.path = [p, obj.path];
                end
                break;
            end

            neighbours = obj.getNodeNeighbours(obj.act.id);
            for i = 1:size(neighbours, 2)
                n = neighbours(:,i);
                % Vozlišče dodamo na odprti seznam, če ni že na zaprtem
                % seznamu in ni ovira
                ids = [obj.closed.id]; z = ids(1,:)==n(1) & ids(2,:)==n(2);
                if isempty(find(z, 1)) && ~obj.map(n(1), n(2))
                    obj.addNodeToOpenList(n);
                end
            end
        end
        path = obj.path;
    end

    function addNodeToOpenList(obj, i)
        g = obj.act.g + obj.cost(i); % Cena poti
        % Preverimo, če je vozlišče že na odprtem seznamu
        ids = [obj.open.id]; s = [];
        if ~isempty(ids)
            s = sum(abs(ids-repmat(i, 1, size(ids, 2))))==0;
        end
        if isempty(find(s, 1)) % Dodamo vozlišče na odprti seznam
            node = struct('id', i, 'src', obj.act.id, ...
                          'g', g, 'h', obj.heuristic(i));
            obj.open = [obj.open, node];
        else % Posodobimo vozlišče na odprtem seznamu, če ima boljšo ceno
```

```
61              if g<obj.open(s).g
62                  obj.open(s).g = g;
63                  obj.open(s).src = obj.act.id;
64              end
65          end
66          % Uredimo odprti seznam
67          [~,i] = sortrows([[obj.open.g]+[obj.open.h]; obj.open.h].', [1,2]);
68          obj.open = obj.open(i);
69      end
70
71      function n = getNodeNeighbours(obj, a)
72          n = [a(1)-1, a(1), a(1), a(1)+1; a(2), a(2)-1, a(2)+1, a(2)];
73          [h, w] = size(obj.map);
74          n = n(:, n(1,:)>=1 & n(1,:)<=h & n(2,:)>=1 & n(2,:)<=w); % Meje
75      end
76
77      function g = cost(obj, a)
78          g = sum(abs(a-obj.act.id)); % Manhattanska razdalja
79      end
80
81      function h = heuristic(obj, a)
82          h = sqrt(sum((a-obj.goal).^2)); % Evklidska razdalja
83      end
84  end
85  end
```

V programu 4.11 je prikazana uporaba algoritma A* iz programa 4.10. Stolpci izhodne spremenljivke `path` predstavljajo urejen seznam celic (vozlišč), ki vodijo od začetka do cilja. Za uporabo različnih hevristik je potrebna minimalna sprememba algoritma (npr. razdalja Manhattan, ki je bila uporabljena za pridobitev poti, prikazana na sliki 4.32).

Program 4.11: Uporaba algoritma A*

./src/pth/example_astar_base.m

```
map = zeros(14, 14); % Zemljevid
map(5:10,[4 11]) = 1; map(5,4:11) = 1; % Ovire

astar = AStarBase();
astar.map = map;
path = astar.find([11; 6], [4; 10])
```

```
path =
  Columns 1 through 13
    11    11    11    11    11    11    10     9     8     7     6     5
     6     7     8     9    10    11    12    12    12    12    12    12
  Columns 14 through 16
     4     4     4
    12    11    10
```

Algoritem A* se lahko prevede v pohlepno iskanje *najprej najboljši*, če je *cena-do-sem* nastavljena na nič.

Primer 4.12

Razširite primer 4.11 za razcep okolja s pomočjo štiriškega drevesa in poiščite optimalno pot z algoritmom A*.

Rešitev

Za razširitev algoritma A* iz programa 4.10 s štiriškim drevesom je potrebnih le nekaj manjših sprememb. V tem primeru niso vse celice enako velike, zato so med njimi različne razdalje in vsaka celica ima drugačno število sosedov. Torej moramo spremeniti način določanja sosednjih celic. Izvedba algoritma za razgradnjo s pomočjo štiriškega drevesa v programu 4.2 ustvari graf vidljivosti tako, da za vsako celico v štiriškem drevesu poišče vse sosednje celice. Za določitev *cene-do-sem* lahko izračunamo razdaljo med celicami kot evklidsko razdaljo med središči celic (glejte sliko 4.10).

Literatura

[1] H. Choset, K. Lynch in sod. *Principles of robot motion: theory, algorithms, and implementations*. MIT Press, illustrate izd., 2005, 603 str.

[2] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006, 1007 str.

[3] B. Siciliano in O. Khatib. *Springer Handbook of Robotics*. Springer, 2008, 1611 str.

[4] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11. Tehn. por., Computer Science Dept., Iowa State Univ., Iowa, 2013.

[5] S. M. LaValle in J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, zv. 20, št. 5, str. 278–400, 2001.

[6] J. J. Kuffner in S. M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. V *IEEE International Conference on Robotics and Automation (ICRA 2000)*, str. 1–7. IEEE, 2000.

[7] L. E. Kavraki, P. Svestka in sod. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, zv. 12, št. 4, str. 566–580, 1996.

[8] Z. Sun, D. Hsu in sod. Narrow Passage Sampling for Probabilistic Roadmap Planning. *IEEE Transactions on Robotics*, zv. 21, št. 6, str. 1105–1115, 2005.

[9] V. Lumelsky in P. Stepanov. Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment. *IEEE Transactions on Automatic Control*, zv. 31, št. 11, str. 1058–1063, 1986.

[10] A. Sankaranarayanan in M. Vidyasagar. A new path planning algorithm for moving a point object amidst unknown obstacles in a plane. V *IEEE Conference on Robotics and Automation*, str. 1930–1936. IEEE, 1990.

[11] I. Kamon in E. Rivlin. Sensory-Based Motion Planning with Global Proofs. *IEEE Transactions on robotics*, zv. 13, št. 6, str. 814–821, 1997.

[12] S. Laubach in J. Burdick. An autonomous Sensor-Based Path-Planner for Planetary Microrovers. V *IEEE Conference on Robotics and Automation*, str. 347–354. IEEE, 1999.

[13] K. Mehlhorn in P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008, 300 str.

[14] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, zv. 1, str. 269–271, 1959.

5

Senzorji v mobilnih sistemih

5.1 Uvod

Kolesni mobilni roboti zaznavajo okolje s pomočjo senzorjev, kar jim omogoča avtonomno delovanje v okolju. Senzorji se uporabljajo za obvladovanje negotovosti in motenj, ki so vedno prisotne v okolju in v vseh robotskih podsistemih, kot je negotovost zemljevida okolice, neznani model gibanja, neznana dinamika itd. Negotovi so tudi izidi akcij (premiki mobilnega robota), zaradi neidealnih aktuatorjev. Primarni namen senzorjev torej je, da zmanjšajo ali odpravijo te negotovosti in s tem omogočijo ocenjevanje stanj mobilnega robota kot tudi izgleda okolice.

Običajno z enim senzorjem ni možna elegantna rešitev, zlasti za uporabo v zaprtih prostorih, za (dovolj) natančno in robustno merjenje želenih informacij, npr. leg robota. Lega je potrebna za lokalizacijo robota, kar pa predstavlja enega izmed največjih izzivov v mobilni robotiki. Zato je potrebna uporaba več senzorjev, kjer s pomočjo metod zlivanja (fuzije) njihovih informacij dobimo bolj kakovostne in robustne informacije. Ocena lege robota običajno združuje relativne in absolutne senzorje. **Relativni senzorji** podajajo informacije relativno glede na koordinate mobilnega robota, medtem ko so meritve **absolutnih senzorjev** podane v nekem globalnem koordinatnem sistemu (npr. zemeljske koordinate).

S pomočjo senzorjev lahko mobilni robot zaznava stanje okolice, pri čemer je potrebno informacije senzorjev predhodno ustrezno analizirati in interpretirati. Meritve v realnem okolju se spreminjajo dinamično (npr. spremembe osvetlitve,

različna absorpcija zvoka ali svetlobe na površinah). Pogrešek merjenja pogosto statistično modeliramo z neko gostoto verjetnosti, za katero običajno predpostavimo, da je simetrična ali celo normalna. Vendar ta predpostavka ni vedno pravilna; npr. ultrazvočni merilnik razdalje lahko zaradi večkratnih odbojev (od oddajnika do sprejemnika) izmeri preveliko razdaljo.

V nadaljevanju so na kratko opisane transformacije koordinatnih sistemov, ki so potrebne za pravilno predstavitev senzorskih meritev robotu in za oceno relevantnih informacij v koordinatnem sistemu robota. Sledi poglavje, v katerem so pojasnjene glavne metode za lokalizacijo, ki se uporabljajo za oceno lege robota v okolici z uporabo izbranih senzorjev. Na koncu je podan kratek pregled pogosto uporabljenih senzorjev v mobilni robotiki.

5.2 Transformacije koordinatnih sistemov

Senzorji običajno niso nameščeni v središču robota ali izhodišču njegovega koordinatnega sistema. Njihova pozicija in orientacija na robotu sta opisani s translacijskim vektorjem in rotacijo glede na koordinatni sistem robota. S pomočjo transformacij lahko izmerjene veličine v koordinatnem sistemu senzorja prevedemo v koordinatni sistem robota.

S transformacijami lahko izrazimo izmerjeni smerni vektor (npr. pospeškometer, magnetometer) ali izmerjene pozicijske koordinate (npr. laserski pregledovalnik razdalj, kamera) v koordinatnem sistemu robota. Poleg tega se mobilni roboti premikajo po prostoru, zato lahko njihovo lego ali premike opišemo z ustreznimi transformacijami.

V nadaljevanju je podan kratek splošni pregled transformacij za tridimenzionalni prostor, čeprav sta v mobilni robotiki običajno dovolj dve dimenziji (npr. ravninsko gibanje, ki ga opisujeta dve translaciji in ena rotacija). Najprej bo opisana transformacija rotacije, nato pa še translacije.

5.2.1 Orientacija in rotacija

Orientacijo nekega lokalnega koordinatnega sistema (npr. senzor) glede na referenčni koordinatni sistem (npr. robot) opisuje rotacijska matrika $\boldsymbol{R}$

$$\boldsymbol{R} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

kjer so enotski vektorji lokalnega koordinatnega sistema $\boldsymbol{u}$, $\boldsymbol{v}$ in $\boldsymbol{w}$ izraženi v referenčnem koordinatnem sistemu kot $\boldsymbol{u} = [u_1,\ u_2,\ u_3]^T$, $\boldsymbol{v} = [v_1,\ v_2,\ v_3]^T$, $\boldsymbol{w} = [w_1,\ w_2,\ w_3]^T$ in velja $\boldsymbol{u} \times \boldsymbol{v} = \boldsymbol{w}$. Vrstice matrike $\boldsymbol{R}$ so komponente lokalnih enotskih vektorjev vzdolž referenčnih enotskih vektorjev $\boldsymbol{x}$, $\boldsymbol{y}$ in $\boldsymbol{z}$. Elementi

matrike $\boldsymbol{R}$ so kosinusi kotov med posameznimi osmi obeh koordinatnih sistemov, zato jo imenujemo **matrika smernih kosinusov** (DCM, angl. *direction cosine matrix*) ali *rotacijska matrika*. Ker so vektorji $\boldsymbol{u}$, $\boldsymbol{v}$ in $\boldsymbol{w}$ ortonormalni, sta inverz in transponiranka matrike $\boldsymbol{R}$ enaki ($\det \boldsymbol{R} = 1$ in $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$). Rotacijska matrika ima 9 parametrov za opis treh stopenj prostosti, vendar ti parametri niso medsebojno neodvisni, ampak so definirani s šestimi omejitvenimi relacijami (vsota kvadratov elementov poljubne vrstice ali stolpca matrike $\boldsymbol{R}$ je 1 in skalarni produkt poljubnih dveh vrstic ali stolpcev matrike $\boldsymbol{R}$ je 0).

Vektor $\boldsymbol{v}_L$ v lokalnem koordinatnem sistemu izrazimo z vektorjem $\boldsymbol{v}_G$ v globalnem koordinatnem sistemu s pomočjo rotacije

$$\boldsymbol{v}_L = \boldsymbol{R}_G^L \boldsymbol{v}_G$$

Rotacijska matrika $\boldsymbol{R}_G^L$ torej transformira vektor iz globalnega G v lokalni L koordinatni sistem.

Osnovne transformacije rotacije dobimo z vrtenjem okoli osi x, y in z z osnovnimi rotacijskimi matrikami

$$\boldsymbol{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \tag{5.1}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{5.2}$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

kjer $\boldsymbol{R}_{os}(kot)$ označuje rotacijo okoli osi za dani kot. Zaporedne rotacije opišemo s produktom rotacijskih matrik, kjer je pomemben vrstni red rotacij. Z rotacijsko matriko opišemo orientacijo togega telesa, vendar so v nekaterih primerih primernejše druge predstavitve, zato bomo v nadaljevanju opisali še dve dodatni obliki: Eulerjeve kote in kvaternione.

Eulerjevi koti

Eulerjevi koti opisujejo orientacijo togega telesa z vrtenjem okoli osi x, y in z. Ti koti so označeni kot:

- φ – nagib ali kot valjanja (angl. *roll*), okoli osi x,
- θ – naklon ali kot prevračanja (angl. *pitch*), okoli osi y,
- ψ – zasuk ali kot sukanja (angl. *yaw* ali *heading*), okoli osi z.

Možnih je 12 različnih kombinacij treh osnovnih rotacij okoli osi x, y in z [1]. Najpogosteje se uporablja kombinacija *3-2-1*, kjer dobimo orientacijo lokalnega koordinatnega sistema glede na referenčni koordinatni sistem tako, da iz začetne lege, kjer sta oba koordinatna sistema poravnana, izvedemo rotacije lokalnega koordinatnega sistema v naslednjem zaporedju:

1. rotacija okoli osi z za kot sukanja ψ,
2. rotacija okoli na novo pridobljene osi y za kot prevračanja θ,
3. rotacija okoli na novo pridobljene osi x za kot valjanja φ.

Rotacijska matrika te transformacije je

$$\begin{aligned}\boldsymbol{R} &= \boldsymbol{R}_x(\varphi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi) = \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi & \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi & \sin\varphi\cos\theta \\ \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi & \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi & \cos\varphi\cos\theta \end{bmatrix}\end{aligned}$$

in predstavitev z Eulerjevimi koti je

$$\begin{aligned}\varphi &= \arctan\left(\frac{R_{23}}{R_{33}}\right) \\ \theta &= -\arcsin(R_{13}) \\ \psi &= \arctan\left(\frac{R_{12}}{R_{11}}\right)\end{aligned} \tag{5.4}$$

Parametrizacija z uporabo Eulerjevih kotov ni redundantna (trije parametri za tri prostostne stopnje). Njena pomanjkljivost pa je singularnost pri $\theta = \pi/2$, kjer imata rotaciji okoli osi z in x enak učinek (sovpadeta). Ta učinek izgube prostostne stopnje je v klasičnih letalskih žiroskopih znan kot problem blokade kardanskega sklopa (angl. *gimbal lock*). Pri zapisu rotacije z Eulerjevimi koti se ta singularnost pojavi zaradi deljenja s $\cos\theta$ (glejte (5.23) v poglavju 5.2.3).

Kvaternioni

Kvaternioni predstavljajo orientacijo v tridimenzionalnem prostoru z uporabo štirih parametrov in ene omejitvene enačbe. So brez problema singularnosti, ki se je pojavil pri predstavitvi z Eulerjevimi koti. Matematično gledano so kvaternioni *nekomutativna* razširitev kompleksnih števil, ki jih zapišemo kot

$$\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$$

kjer za kompleksne elemente i, j in k velja $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$; q_0 je skalarni del kvaterniona in $q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$ je vektorski del. Norma kvaterniona je določena z

$$\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q}\boldsymbol{q}^*} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

kjer je $\boldsymbol{q}^*$ konjugiran kvaternion, ki ga izračunamo kot $\boldsymbol{q}^* = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k}$. Inverzno vrednost kvaterniona določimo s pomočjo njegove konjugirane vrednosti in norme

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|^2}$$

Z enotskimi kvaternioni lahko parametriziramo rotacijo v prostoru

$$\begin{aligned} q_0 &= \cos \Delta\varphi/2 \\ q_1 &= e_1 \sin \Delta\varphi/2 \\ q_2 &= e_2 \sin \Delta\varphi/2 \\ q_3 &= e_3 \sin \Delta\varphi/2 \end{aligned} \tag{5.5}$$

kjer je $\boldsymbol{e}^T = [e_1,\ e_2,\ e_3]$ enotski vektor osi vrtenja in $\Delta\varphi$ kot zasuka okoli te osi. Za enotske kvaternione velja $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$.

Transformacija

$$\boldsymbol{q}_{v'} = \boldsymbol{q}^{-1} \circ \boldsymbol{q}_v \circ \boldsymbol{q}$$

zavrti vektor $\boldsymbol{v} = [x,\ y,\ z]^T$, podan s kvaternionom

$$\boldsymbol{q}_v = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$$

(ali enakovredno $\boldsymbol{q}_v = [0,\ x,\ y,\ z]^T$), okoli osi $\boldsymbol{e}$ za kot $\Delta\varphi$ v vektor $\boldsymbol{v}' = [x',\ y',\ z']^T$, izražen s kvaternionom

$$\boldsymbol{q}_{v'} = x'\boldsymbol{i} + y'\boldsymbol{j} + z'\boldsymbol{k}$$

kjer $\circ$ označuje produkt kvaternionov, definiran v (5.6) in (5.7).

Druga prednost kvaternionov je relativno enostavna kombinacija zaporednih rotacij. Kvaternion, ki ustreza produktu dveh rotacijskih matrik, je enak produktu obeh kvaternionov [1]. Imamo kvaterniona

$$\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$$

in

$$\boldsymbol{q}' = q_0' + q_1'\boldsymbol{i} + q_2'\boldsymbol{j} + q_3'\boldsymbol{k}$$

Če nek vektor zavrtimo iz njegove začetne orientacije za zasuk $\boldsymbol{q}'$, nato pa še za zasuk $\boldsymbol{q}$, je celoten zasuk vektorja

$$\begin{aligned} \boldsymbol{q}'' = \boldsymbol{q}' \circ \boldsymbol{q} = {} & (q_0'q_0 - q_1'q_1 - q_2'q_2 - q_3'q_3) \\ & + \boldsymbol{i}(q_1'q_0 + q_2'q_3 - q_3'q_2 + q_0'q_1) \\ & + \boldsymbol{j}(-q_1'q_3 + q_2'q_0 + q_3'q_1 + q_0'q_2) \\ & + \boldsymbol{k}(q_1'q_2 - q_2'q_1 + q_3'q_0 + q_0'q_3) \end{aligned} \tag{5.6}$$

ali v vektorsko-matrični obliki

$$\begin{bmatrix} q_0'' \\ q_1'' \\ q_2'' \\ q_3'' \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} q_0' \\ q_1' \\ q_2' \\ q_3' \end{bmatrix} \tag{5.7}$$

Povezava med kvaternionom in predstavitvijo z rotacijsko matriko je podana z

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_0q_3+q_1q_2) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_3q_0) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_0q_1+q_2q_3) \\ 2(q_0q_2+q_1q_3) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{5.8}$$

ali v obratni smeri

$$q_0 = \frac{1}{2}\sqrt{1+R_{11}+R_{22}+R_{33}} \tag{5.9}$$

$$q_1 = \frac{1}{4q_0}(R_{23}-R_{32})$$

$$q_2 = \frac{1}{4q_0}(R_{31}-R_{13})$$

$$q_3 = \frac{1}{4q_0}(R_{12}-R_{21}) \tag{5.10}$$

Če je q_0 v enačbi (5.9) blizu nič, je pretvorba iz rotacijske matrike v kvaternion (enačbe (5.9) – (5.10)) singularna. V tem primeru lahko izračunamo kvaternion z uporabo ekvivalentne oblike (enačbe (5.11) – (5.12)) brez numeričnih problemov

$$q_0 = \frac{1}{4T}(R_{32}-R_{23}) \tag{5.11}$$

$$q_1 = T$$

$$q_2 = \frac{1}{4T}(R_{12}+R_{21})$$

$$q_3 = \frac{1}{4T}(R_{13}+R_{31}) \tag{5.12}$$

kjer je $T = \frac{1}{2}\sqrt{1+R_{11}-R_{22}-R_{33}}$.

Povezavo med kvaternioni in Eulerjevimi koti (notacija *3-2-1*) dobimo z matrikami $\boldsymbol{R}_x(\varphi)$, $\boldsymbol{R}_y(\theta)$ in $\boldsymbol{R}_z(\psi)$, kar ustreza kvaternionom $[\cos(\varphi/2)+\boldsymbol{i}\sin(\varphi/2)]$, $[\cos(\theta/2)+\boldsymbol{j}\sin(\theta/2)]$ in $[\cos(\psi/2)+\boldsymbol{k}\sin(\psi/2)]$. Kvaternion za rotacijo *3-2-1* je

$$\boldsymbol{q} = [\cos(\psi/2)+\boldsymbol{k}\sin(\psi/2)][\cos(\theta/2)+\boldsymbol{j}\sin(\theta/2)][\cos(\varphi/2)+\boldsymbol{i}\sin(\varphi/2)]$$

ali v vektorski obliki

$$\boldsymbol{q} = \begin{bmatrix} \cos(\varphi/2)\cos(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\varphi/2)\cos(\theta/2)\cos(\psi/2)-\cos(\varphi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\sin(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\cos(\theta/2)\sin(\psi/2)-\sin(\varphi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}$$

Obratna transformacija se glasi

$$\varphi = \arctan\left(\frac{2(q_1q_0+q_2q_3)}{q_0^2-q_1^2-q_2^2+q_3^2}\right)$$

$$\theta = -\arcsin\left(2(q_1q_3-q_2q_0)\right)$$

$$\psi = \arctan\left(\frac{2(q_3q_0+q_1q_2)}{q_0^2+q_1^2-q_2^2-q_3^2}\right)$$

Primer 5.1

Koordinatni sistem senzorja (magnetometer) je glede na koordinatni sistem robota zasukan. Naj bo koordinatni sistem senzorja lokalen (L), koordinatni sistem robota pa globalen (G). Orientacija senzorja glede na robota je opisana z dvema rotacijama. Na začetku sta oba koordinatna sistema poravnana, nato (L) zavrtimo okoli osi x za kot $\alpha_x = 90°$ in nato še enkrat okoli nove osi y za kot $\alpha_y = 45°$.

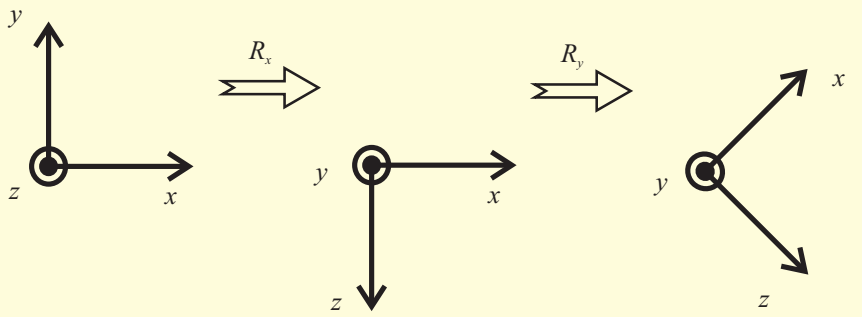
1. Kakšna je orientacija senzorja, izražena z rotacijsko matriko $\boldsymbol{R}_G^L$, glede na koordinatni sistem robota? Določite Eulerjeve kote (notacija *3-2-1*) in kvaternion $\boldsymbol{q}_G^L$, ki opisujejo to transformacijo.
2. Magnetometer meri smerni vektor $\boldsymbol{v} = [0,\ 0,\ 1]^T$ magnetnega polja. Kakšna je predstavitev tega vektorja v koordinatah robota?

Rešitev

1. Končno orientacijo opisuje skupna rotacijska matrika, kjer je pomemben vrstni red množenja

$$\begin{aligned}\boldsymbol{R}_G^L &= \boldsymbol{R}_y(\alpha_y)\boldsymbol{R}_x(\alpha_x) = \\ &= \begin{bmatrix}\cos(\alpha_y) & 0 & -\sin(\alpha_y)\\ 0 & 1 & 0\\ \sin(\alpha_y) & 0 & \cos(\alpha_y)\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\alpha_x) & \sin(\alpha_x)\\ 0 & -\sin(\alpha_x) & \cos(\alpha_x)\end{bmatrix} \\ &= \begin{bmatrix}0{,}7071 & 0{,}7071 & 0\\ 0 & 0 & 1\\ 0{,}7071 & -0{,}7071 & 0\end{bmatrix}\end{aligned} \tag{5.13}$$

   Vrstice v matriki $\boldsymbol{R}_G^L$ predstavljajo komponente novih osi senzorja, izražene v koordinatnem sistemu robota, kar je prikazano grafično na sliki 5.1.

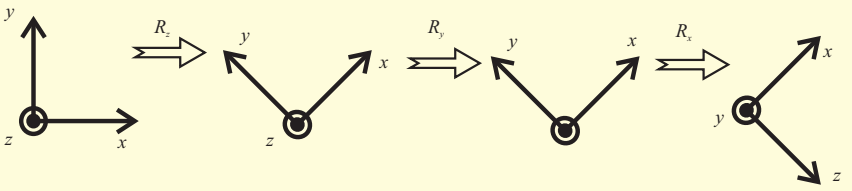


Slika 5.1: Prikaz rotacij v enačbi (5.13)

2. Eulerjeve kote za notacijo *3-2-1* izračunamo iz matrike $\boldsymbol{R} = \boldsymbol{R}_G^L$

$$\begin{aligned} \varphi &= \arctan(\frac{R_{23}}{R_{33}}) = 90° \\ \theta &= -\arcsin(R_{13}) = 0° \\ \psi &= \arctan(\frac{R_{12}}{R_{11}}) = 45° \end{aligned} \tag{5.14}$$

kjer dobimo rotacijsko matriko z $\boldsymbol{R}_G^L = \boldsymbol{R}_x(\varphi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)$, katere osnovne rotacije $\boldsymbol{R}_x(\varphi)$, $\boldsymbol{R}_y(\theta)$ in $\boldsymbol{R}_z(\psi)$ so določene v enačbi (5.3) — glejte sliko 5.2.



Slika 5.2: Prikaz rotacij z Eulerjevimi koti (5.14)

3. Kvaternion $\boldsymbol{q}_G^L$ dobimo s produktom kvaternionov zaporednih rotacij

$$\boldsymbol{q}_G^L = \boldsymbol{q}_x \circ \boldsymbol{q}_y$$

kjer je $\boldsymbol{q}_x$, glede na enačbo (5.5), definiran s kotom zasuka $\Delta\varphi_x = 90°$ okoli rotacijske osi $\boldsymbol{e}_x = [1,\ 0,\ 0]^T$ in $\boldsymbol{q}_y$ s kotom zasuka $\Delta\varphi_y = 45°$ okoli rotacijske osi $\boldsymbol{e}_y = [0,\ 1,\ 0]^T$

$$\boldsymbol{q}_x = \begin{bmatrix} \cos\Delta\varphi_x/2 \\ e_{x_1}\sin\Delta\varphi_x/2 \\ e_{x_2}\sin\Delta\varphi_x/2 \\ e_{x_3}\sin\Delta\varphi_x/2 \end{bmatrix} = \begin{bmatrix} 0{,}7071 \\ 0{,}7071 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{q}_y = \begin{bmatrix} \cos\Delta\varphi_z/2 \\ e_{y_1}\sin\Delta\varphi_y/2 \\ e_{y_2}\sin\Delta\varphi_y/2 \\ e_{y_3}\sin\Delta\varphi_y/2 \end{bmatrix} = \begin{bmatrix} 0{,}9239 \\ 0 \\ 0{,}3827 \\ 0 \end{bmatrix}$$

Končni kvaternion (glejte definicijo produkta kvaternionov v (5.7)) ima obliko

$$\boldsymbol{q}_G^L = \boldsymbol{q}_x \circ \boldsymbol{q}_y = \begin{bmatrix} 0{,}6533 \\ 0{,}6533 \\ 0{,}2706 \\ 0{,}2706 \end{bmatrix}$$

kar ustreza kotu zasuka $\Delta\varphi = 2\arccos(q_0) = 2\arccos(0{,}6533) = 98{,}41°$ okoli rotacijske osi $\boldsymbol{e} = \frac{1}{\sin\frac{\Delta\varphi}{2}}[q_1, q_2, q_3]^T = [0{,}8630,\ 0{,}3574,\ 0{,}3574]^T$.

4. Merjeni smerni vektor $\boldsymbol{v}_L = [0,\ 0,\ 1]^T$ zapišemo v koordinatah robota kot

$$\boldsymbol{v}_G = \boldsymbol{R}_L^G \boldsymbol{v}_L = [0{,}707\,11,\ -0{,}707\,11,\ 0]^T$$

kjer je $\boldsymbol{R}_L^G = (\boldsymbol{R}_G^L)^{-1} = (\boldsymbol{R}_G^L)^T$.

Enako dobimo s kvaternioni, kjer je vektor po rotaciji enak

$$\boldsymbol{q}_{v_G} = (\boldsymbol{q}_L^G)^{-1} \circ \boldsymbol{q}_{v_L} \circ \boldsymbol{q}_L^G$$

ter velja $\boldsymbol{q}_{v_L} = [0,\ \boldsymbol{v}_L^T]^T$ in $\boldsymbol{q}_L^G = (\boldsymbol{q}_G^L)^{-1}$ (glejte 5.2.1). Produkt kvaternionov je definiran v (5.7). Dobimo

$$\boldsymbol{q}_{v_G} = \begin{bmatrix} 0{,}6533 \\ 0{,}6533 \\ 0{,}2706 \\ 0{,}2706 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0{,}6533 \\ -0{,}6533 \\ -0{,}2706 \\ -0{,}2706 \end{bmatrix} = \begin{bmatrix} 0 \\ 0{,}7071 \\ -0{,}7071 \\ 0 \end{bmatrix}$$

kar ustreza vektorju $\boldsymbol{v}_G = [0{,}7071,\ -0{,}7071,\ 0]^T$ (upoštevamo le vektorski del kvaterniona).

Primer 5.2

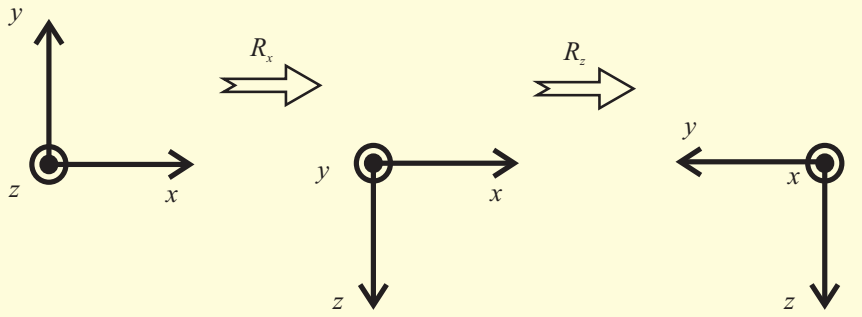
Na začetku sta globalni koordinatni sistem G in lokalni koordinatni sistem L poravnana, nato pa se L zavrti okoli osi x za kot $\alpha_x = 90°$ in nato ponovno okoli nove osi z za kot $\alpha_z = 90°$.

1. Kakšna je orientacija koordinatnega sistema L, izražena z rotacijsko matriko ($\boldsymbol{R}_G^L$), glede na koordinatni sistem G? Določite Eulerjeve kote (notacija *3-2-1*) in kvaternion $\boldsymbol{q}_G^L$, ki opisujejo to transformacijo.
2. Vektor v globalnih koordinatah je $\boldsymbol{v}_G = [0,\ 0,\ 1]^T$. Kakšna je predstavitev tega vektorja v lokalnih koordinatah?

Rešitev

1. Za rotacijo zapišemo rotacijsko matriko (na sliki 5.3 je še grafična predstavitev te rotacije)

$$\boldsymbol{R}_G^L = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.15}$$



Slika 5.3: Prikaz rotacije (5.15)

2. Eulerjeve kote (notacija *3-2-1*) ocenimo iz matrike $\boldsymbol{R}_G^L$

$$\varphi = \arctan\left(\frac{R_{23}}{R_{33}}\right) = \text{nedefinirano}$$
$$\theta = -\arcsin\left(R_{13}\right) = -90°$$
$$\psi = \arctan\left(\frac{R_{12}}{R_{11}}\right) = \text{nedefinirano}$$

Opazimo, da je $\theta = -90°$, kar pomeni, da je parametrizacija z Eulerjevimi koti singularna in sta zato φ ter ψ nedefinirana. Z uporabo Eulerjevih kotov torej ne moremo zapisati te orientacije (rotacije).

3. Kvaternion $\boldsymbol{q}_G^L$ je

$$\boldsymbol{q}_G^L = \boldsymbol{q}_x \circ \boldsymbol{q}_z$$

kjer je

$$\boldsymbol{q}_x = \begin{bmatrix} 0{,}7071 \\ 0{,}7071 \\ 0 \\ 0 \end{bmatrix}$$
$$\boldsymbol{q}_z = \begin{bmatrix} 0{,}7071 \\ 0 \\ 0 \\ 0{,}7071 \end{bmatrix}$$

Torej je

$$\boldsymbol{q}_G^L = \boldsymbol{q}_x \circ \boldsymbol{q}_z = \begin{bmatrix} 0{,}5 \\ 0{,}5 \\ -0{,}5 \\ 0{,}5 \end{bmatrix}$$

kar ustreza zasuku za kot $\Delta\varphi = 2\arccos(0{,}5) = 120°$ okoli rotacijske osi $\boldsymbol{e} = \frac{1}{\sin\frac{\Delta\varphi}{2}}[q_1, q_2, q_3]^T = [0{,}5774,\ -0{,}5774,\ 0{,}5774]^T$.

4. Vektor $\boldsymbol{v}_G = [0,\ 0,\ 1]^T$ je v lokalnih koordinatah zapisan kot

$$\boldsymbol{v}_L = \boldsymbol{R}_G^L \boldsymbol{v}_G = [1,\ 0,\ 0]^T$$

ali s kvaternioni

$$\boldsymbol{q}_{v_L} = \left(\boldsymbol{q}_G^L\right)^{-1} \circ \boldsymbol{q}_{v_G} \circ \boldsymbol{q}_G^L = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

kjer je vektorski del enak $\boldsymbol{v}_L = [1,\ 0,\ 0]^T$.

5.2.2 Translacija in rotacija

Za posplošeno predstavitev bomo označili koordinatni sistem senzorja kot L (lokalne koordinate) in koordinatni sistem robota kot G (globalne koordinate). Lokacija senzorja glede na koordinatni sistem robota je opisana s translacijskim vektorjem $\boldsymbol{t}_L^G = [t_x,\ t_y,\ t_z]$ in rotacijsko matriko $\boldsymbol{R}_G^L$. Translacija $\boldsymbol{t}_L^G$ opisuje pozicijo izhodišča lokalnega koordinatnega sistema v globalnih koordinatah in rotacijska matrika $\boldsymbol{R}_G^L$ opisuje orientacijo lokalnega koordinatnega sistema glede na globalni koordinatni sistem (robota). Točko $\boldsymbol{p}_G$, podano v globalnih koordinatah, lahko opišemo z lokalnimi koordinatami z uporabo transformacije

$$\boldsymbol{p}_L = \boldsymbol{R}_G^L \left(\boldsymbol{p}_G - \boldsymbol{t}_L^G\right)$$

njena inverzna transformacija pa je podana z

$$\boldsymbol{p}_G = (\boldsymbol{R}_G^L)^{-1} \boldsymbol{p}_L + \boldsymbol{t}_L^G = \left(\boldsymbol{R}_G^L\right)^T \boldsymbol{p}_L + \boldsymbol{t}_L^G$$

Primer 5.3

Robot ima laserski pregledovalnik razdalj (LPR), ki izmeri položaj najbližje točke ovire $\boldsymbol{p}_L = [1,\ 0{,}5,\ 0{,}4]^T$ m v koordinatah senzorja. Ima tudi magnetometer, ki ob istem času izmeri vektor zemeljskega magnetnega polja $\boldsymbol{v}_L^T = [22,\ 1,\ 42]$ nT.

LPR je glede na (globalni) koordinatni sistem robota premaknjen za $\boldsymbol{t}_1 = [0{,}1,\ 0,\ 0{,}25]^T$ in zasukan za 30° okoli osi z. Magnetometer pa je premaknjen za $\boldsymbol{t}_2 = [0,\ 0{,}1,\ 0{,}2]$ in zasukan za Eulerjeve kote $\varphi = 0°$, $\theta = 10°$,$\psi = 20°$ (notacija *3-2-1*).

1. Katere so najbližje koordinate točke ovire v koordinatnem sistemu robota?
2. Kakšen je vektor magnetnega polja v koordinatnem sistemu robota?

Rešitev

1. Rotacijska matrika je

$$\boldsymbol{R}_G^L = \begin{bmatrix} \cos(30°) & \sin(30°) & 0 \\ -\sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0{,}866 & 0{,}5 & 0 \\ -0{,}5 & 0{,}8660 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   Točka, izražena v koordinatah robota, pa je

$$\boldsymbol{p}_G = (\boldsymbol{R}_G^L)^T \boldsymbol{p}_L + \boldsymbol{t}_1 = [0{,}716,\ 0{,}933,\ 0{,}65]^T$$

2. Pri transformaciji vektorja je pomembna le rotacija, tako da so komponente magnetnega vektorja v koordinatah robota dobljene z rotacijsko transformacijo

$$\boldsymbol{R}_G^L = \boldsymbol{R}_x(0°)\boldsymbol{R}_y(10°)\boldsymbol{R}_z(20°) = \begin{bmatrix} 0{,}9254 & 0{,}3368 & -0{,}1736 \\ -0{,}3420 & 0{,}9397 & 0 \\ 0{,}1632 & 0{,}0594 & 0{,}9848 \end{bmatrix}$$

$$\boldsymbol{v}_G = (\boldsymbol{R}_G^L)^T \boldsymbol{v}_L = [26{,}8705,\ 10{,}8443,\ 37{,}5417]^T$$

5.2.3 Kinematika rotacijskih koordinatnih sistemov

V tem poglavju bomo pokazali, kako je orientacija togega telesa, ki je predstavljena s kvaternionom ali rotacijsko matriko, povezana s kotnimi hitrostmi okoli lokalnih osi togega telesa. Togo telo se vrti okrog svojih osi x, y in z s kotnimi hitrostmi ω_x, ω_y in ω_z, zato se orientacija togega telesa (npr. robota ali senzorske enote) spreminja glede na referenčni koordinatni sistem.

Rotacijska kinematika izražena s kvaternioni

Časovno odvisnost rotacije togega telesa (podana z diferencialno enačbo) lahko izpeljemo iz definicije produkta dveh kvaternionov (5.7). Če je orientacija togega telesa $\boldsymbol{q}(t)$ v času t znana, lahko njegovo orientacijo v času $t + \Delta t$ zapišemo kot

$$\boldsymbol{q}(t + \Delta t) = \boldsymbol{q}(t) \circ \Delta\boldsymbol{q}(t) \tag{5.16}$$

kjer $\Delta\boldsymbol{q}(t)$ podaja spremembo orientacije togega telesa iz $\boldsymbol{q}(t)$ v $\boldsymbol{q}(t + \Delta t)$. Z drugimi besedami, $\Delta\boldsymbol{q}(t)$ je orientacija telesa v času $t + \Delta t$ glede na njegovo orientacijo v času t. Do končne orientacije telesa $\boldsymbol{q}(t + \Delta t)$ torej pridemo tako, da najprej zavrtimo telo za rotacijo $\boldsymbol{q}(t)$ glede na nek referenčni koordinatni sistem

in nato za rotacijo $\Delta\boldsymbol{q}(t)$ glede na $\boldsymbol{q}(t)$. Tako s pomočjo enačbe (5.5) zapišemo $\Delta\boldsymbol{q}(t)$ kot

$$\Delta\boldsymbol{q}(t) = \begin{bmatrix} \cos\Delta\varphi/2 \\ e_x \sin\Delta\varphi/2 \\ e_y \sin\Delta\varphi/2 \\ e_z \sin\Delta\varphi/2 \end{bmatrix}$$

kjer je $\boldsymbol{e}(t) = [e_x,\ e_y,\ e_z]^T$ os rotacije, izražena v lokalnih koordinatah togega telesa v času t in $\Delta\varphi$ je kot zasuka v časovnem intervalu Δt. Ob predpostavki, da sta $\boldsymbol{e}(t)$ in $\Delta\varphi$ konstantna v časovnem intervalu Δt, lahko preoblikujemo produkt kvaternionov (5.16) s pomočjo (5.7) kot

$$\boldsymbol{q}(t+\Delta t) = \left(\cos\left(\frac{\Delta\varphi}{2}\right)\boldsymbol{I} + \sin\left(\frac{\Delta\varphi}{2}\right) \begin{bmatrix} 0 & -e_x & -e_y & -e_z \\ e_x & 0 & e_z & -e_y \\ e_y & -e_z & 0 & e_x \\ e_z & e_y & -e_x & 0 \end{bmatrix} \right) \boldsymbol{q}(t) \quad (5.17)$$

kjer je $\boldsymbol{I}$ enotska matrika dimenzij 4×4. Za kratke intervale Δt lahko upoštevamo aproksimacijo $\Delta\varphi \approx \sqrt{{\omega_x}^2 + {\omega_y}^2 + {\omega_z}^2}\Delta t$, kjer je $\boldsymbol{\omega}(t) = [\omega_x,\ \omega_y,\ \omega_z]^T$ vektor trenutnih kotnih hitrosti, ki ga lahko zapišemo tudi v obliki $\boldsymbol{\omega}(t) = \sqrt{{\omega_x}^2 + {\omega_y}^2 + {\omega_z}^2}\boldsymbol{e}$. Za majhne kote lahko (5.17) aproksimiramo z

$$\boldsymbol{q}(t+\Delta t) \approx \left(\boldsymbol{I} + \frac{\Delta t\boldsymbol{\Omega}}{2}\right)\boldsymbol{q}(t)$$

kjer je

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

Diferencialno enačbo, ki opisuje orientacijo togega telesa, dobimo z limitiranjem Δt proti nič

$$\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t+\Delta t) - \boldsymbol{q}(t)}{\Delta t} = \frac{1}{2}\boldsymbol{\Omega}\boldsymbol{q} \quad (5.18)$$

kjer so kotne hitrosti v $\boldsymbol{\Omega}$ podane v koordinatnem sistemu togega telesa.

Rotacijska kinematika izražena z rotacijsko matriko

Izpeljimo še diferencialno enačbo za predstavitev orientacije togega telesa, ki jo podaja rotacijska matrika. Podobno kot v (5.16) lahko zapišemo

$$\boldsymbol{R}(t+\Delta t) = \Delta\boldsymbol{R}(t)\boldsymbol{R}(t) \quad (5.19)$$

kjer je $\boldsymbol{R}(t)$ orientacija togega telesa v času t, $\boldsymbol{R}(t+\Delta t)$ orientacija togega telesa v času $t+\Delta t$ in $\Delta\boldsymbol{R}(t)$ sprememba orientacije (orientacija telesa v času $t = t+\Delta t$) glede na orientacijo v času t.

Sprememba orientacije $\Delta\boldsymbol{R}(t)$ je definirana kot

$$\Delta\boldsymbol{R}(t) = e^{\int_t^{t+\Delta t} \boldsymbol{\Omega}'\,\mathrm{d}t} \quad (5.20)$$

kjer je

$$\boldsymbol{\Omega}' = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix}$$

in $\boldsymbol{\omega}(t) = [\omega_x,\ \omega_y,\ \omega_z]^T$ je vektor trenutnih kotnih hitrosti telesa.

Ob predpostavki, da je $\boldsymbol{\Omega}'$ konstantna matrika v časovnem intervalu Δt, lahko aproksimiramo izraz $\int_t^{t+\Delta t} \boldsymbol{\Omega}'(t)\,\mathrm{d}t \approx \boldsymbol{\Omega}'\Delta t = \boldsymbol{B}$. Eksponent v (5.20) razvijemo v Taylorjevo vrsto

$$\begin{aligned} \Delta\boldsymbol{R}(t) = e^{\boldsymbol{B}} = \\ &= \left(\boldsymbol{I} + \boldsymbol{B} + \frac{\boldsymbol{B}^2}{2!} + \frac{\boldsymbol{B}^3}{3!} + \dots\right) = \\ &= \left(\boldsymbol{I} + \boldsymbol{B} + \frac{\boldsymbol{B}^2}{2!} - \frac{\sigma^2\boldsymbol{B}}{3!} - \frac{\sigma^2\boldsymbol{B}^2}{4!} + \frac{\sigma^4\boldsymbol{B}}{5!} + \dots\right) = \\ &= \left(\boldsymbol{I} + \frac{\sin\sigma}{\sigma}\boldsymbol{B} + \frac{1-\cos\sigma}{\sigma^2}\boldsymbol{B}^2\right) \end{aligned} \tag{5.21}$$

kjer je $\boldsymbol{I}$ enotska matrika dimenzij 3×3 in $\sigma = \Delta t\sqrt{{\omega_x}^2 + {\omega_y}^2 + {\omega_z}^2}$. Za majhne kote σ lahko (5.21) aproksimiramo z

$$\Delta\boldsymbol{R}(t) = \boldsymbol{I} + \boldsymbol{B} = \boldsymbol{I} + \boldsymbol{\Omega}'\Delta t$$

kar lahko uporabimo za izračun rotacijske matrike (5.19) v času $t + \Delta t$

$$\boldsymbol{R}(t + \Delta t) = (\boldsymbol{I} + \boldsymbol{\Omega}'\Delta t)\,\boldsymbol{R}(t)$$

Končno diferencialno enačbo dobimo z limitiranjem Δt proti nič

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}t} = \lim_{\Delta t\to 0} \frac{\boldsymbol{R}(t+\Delta t) - \boldsymbol{R}(t)}{\Delta t} = \boldsymbol{\Omega}'\boldsymbol{R} \tag{5.22}$$

Zavoljo popolnosti je podana tudi enakovredna parametrična predstavitev rotacije z Eulerjevimi koti (notacija *3-2-1*)

$$\begin{aligned} \dot{\varphi} &= \omega_x + \omega_y \sin\varphi \tan\theta + \omega_z \cos\varphi \tan\theta \\ \dot{\theta} &= \omega_y \cos\varphi - \omega_z \sin\varphi \\ \dot{\Psi} &= \frac{\omega_y \sin\varphi + \omega_z \cos\varphi}{\cos\theta} \end{aligned} \tag{5.23}$$

kjer lahko vidimo, da postane notacija (prva in tretja enačba (5.23)) singularna pri $\theta = \pm\pi/2$.

5.2.4 Projekcijska geometrija

Projekcija je preslikava prostora z $N > 0$ dimenzijami v prostor z $M < N$ dimenzijami. Običajno se pri projekcijski preslikavi nekaj informacije nepovratno izgubi. Če pa je na voljo več projekcij objekta iz različnih vidnih kotov, je v

nekaterih primerih možno rekonstruirati opazovani objekt v N-dimenzionalnem prostoru. Dve najbolj osnovni projekciji sta: *perspektivična projekcija* in *vzporedna projekcija* (linearna preslikava z goriščem v neskončnosti).

Glede na *model kamere z luknjico*, se 3D-točka $\boldsymbol{p}_C^T = [x_C \; y_C \; z_C]$, podana v koordinatnem sistemu kamere C, preslika v 2D-točko $\boldsymbol{p}_P^T = [x_P \; y_P]$ v koordinatnem sistemu slike P kot sledi (glejte sliko 5.4)

$$\underline{\boldsymbol{p}}_P = \frac{1}{z_C}\boldsymbol{S}\boldsymbol{p}_C \tag{5.24}$$

kjer $\underline{\boldsymbol{p}}_P$ predstavlja točko $\boldsymbol{p}_P^T = [x_P \; y_P]$ v homogenih koordinatah, tj. $\underline{\boldsymbol{p}}_P^T = [x_P \; y_P \; 1]$. Matrika $\boldsymbol{S} \in \mathbb{R}^3 \times \mathbb{R}^3$ opisuje *notranji model kamere*

$$\boldsymbol{S} = \begin{bmatrix} \alpha_x f & \gamma & c_x \\ 0 & \alpha_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.25}$$

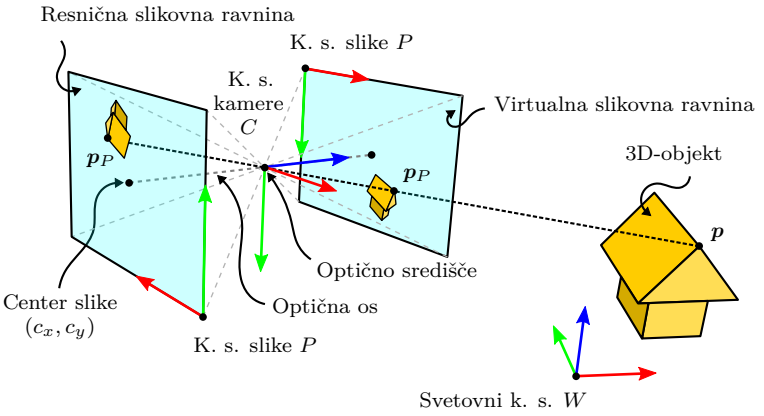
Intrinzični (notranji) parametri kamere vsebovani v $\boldsymbol{S}$ so: goriščna razdalja f; faktorja skaliranja α_x in α_y v horizontalni in vertikalni smeri; optično središče na sliki (c_x, c_y) in strig γ. Parametre modela kamere se običajno določi ali popravi s postopkom kalibracije kamere. Perspektivični model kamere (5.24) ne nelinearen, ker vsebuje člen z_C^{-1} (inverz razdalje do točke v smeri osi z v koordinatnem sistemu kamere C). Čeprav je perspektivična transformacija invariantna za točke, premice in splošne stožnice (stožnice so dvodimenzionalne krivulje, ki nastanejo pri preseku stožca z ravnino) — tj. točke se preslikajo v točke, premice v premice in stožnice v stožnice —, je slika prizora nekoliko popačena predstavitev opazovanega prizora. V splošnem velja, da se koti med premicami in razmerja med razdaljami ne ohranjajo (tj. vzporedne premice se v splošnem ne preslikajo v vzporedne premice). V nekaterih posebnih postavitvah kamere je perspektivično projekcijo možno zadovoljivo aproksimirati z ustreznim linearnim modelom [2] — kar lahko poenostavi kalibracijo kamere. Model kamere (5.24) lahko zapišemo tudi v obliki

$$\underline{\boldsymbol{p}}_P \propto \boldsymbol{S}\boldsymbol{p}_C$$

Slika (projekcija) opazovanega objekta nastane na zaslonu za optičnim središčem kamere (na goriščni razdalji f v smeri negativne osi z_C od optičnega središča) in je obrnjena za 180° ter pomanjšana. Pri vizualizaciji projekcije kamere si lahko zamislimo, da nastane ne-obrnjena slika pred optičnim središčem kamere (na pozitivni osi z_C na enaki razdalji od optičnega središča kot prava slika), kot je prikazano na sliki 5.4.

Primer 5.4

Notranji model kamere (5.25) ima naslednje parametre: $\alpha_x f = \alpha_y f = 1000$, brez striga ($\gamma = 0$) in optično središče je na sredini slike, ki je dimenzij 1024 krat 768. Preslikajte naslednjo množico 3D-točk, ki so podane v koordinatnem sistemu



Slika 5.4: Projekcija pri modelu kamere z luknjico

kamere, na zaslon kamere (v koordinatni sistem slike)

$$\boldsymbol{p}_C^T \in \{[-1\ 1\ 4],\ [1\ 1\ 5],\ [0\ \ -1\ 4],\ [-1\ 1\ \ -4],\ [4\ 1\ 5]\}$$

Rešitev

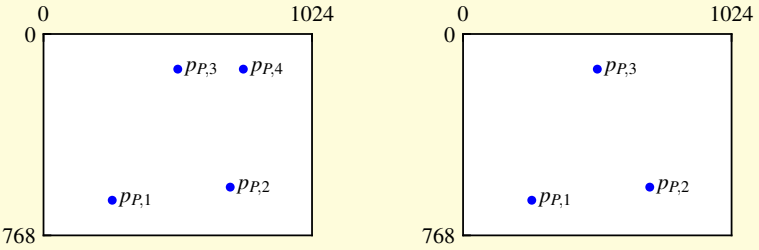
Projekcija 3D-točk na sliko z enačbo (5.24) podana v programu 5.1. Rezultati so prikazani tudi grafično na levi strani slike 5.5. Opazimo, da četrta točka ni prikazana na sliki, ker je izven omejene slikovne ravnine (zaslona). Nahaja se za kamero, in torej ni vidna. To je posledica dejstva, da matematični model (5.24) ne upošteva omejitev vidnega polja kamere. Zatorej moramo izvesti dodatna preverjanja, da zagotovimo, da se na slikovno ravnino preslikajo le točke v vidnem polju kamere: tj. projicirane točke morajo biti znotraj meja slikovne ravnine in pred kamero. Zatorej se na slikovni ravnini v resnici pojavijo le prve tri točke (glejte desno stran slike 5.5).

Program 5.1: Rešitev primera 5.4

./src/sen/example_projection.m

```
% Notranji parametri kamere in velikost zaslona
alphaF = 1000; % alpha*f, v px/m
s = [1024; 768]; % Velikost zaslona, v px
c = s/2; % Optično središče na sliki, v px
S = [alphaF, 0, c(1); 0, alphaF, c(2); 0, 0, 1]; % Model kamere

% Množica 3D točk v koordinatnem sistemu kamere
pC = [-1 1 4; 1 1 5; 0 -1 4; -1 1 -4; 4 1 5].';

% Projekcija točk na slikovno ravnino
pP = (S*pC)./repmat(pC(3,:), 3, 1)
```

```
pP =
         262         712         512         762        1312
         634         584         134         134         584
           1           1           1           1           1
```

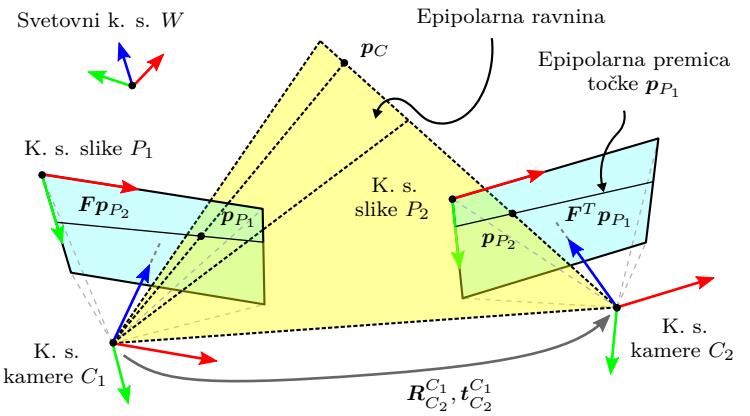


Slika 5.5: Projicirane točke na slikovno ravnino iz primera 5.4. Na levi so točke, ki so preslikane z matematičnim modelom (5.24) in na desni so točke, ki so v resnici vidne na slikovni ravnini.

Jasno je, da je mogoče s preslikavo (5.24) enolično preslikati vsako točko v 3D-prostoru na slikovno ravnino, kar pa ne velja za inverzno preslikavo. Ker perspektivična preslikava povzroči izgubo informacije o globini prizora, lahko vsako točko na slikovni ravnini preslikamo le v poltrak (žarek) v 3D-prostoru, če ni na voljo nobene dodatne informacije. Prizor lahko rekonstruiramo, če lahko nekako pridobimo podatek o globini. Obstaja veliko metod, ki omogočajo 3D-rekonstrukcijo in temeljijo na uporabi globinskih kamer, strukturirane svetlobe, dodatni svetlobnih namigov, gibanja itd. Položaj točke v 3D-prostoru lahko ocenimo tudi na podlagi več (najmanj dveh) projekcij 3D-točke iz različnih pogledov. 3D-rekonstrukcija je torej možna z uporabo stereo kamere.

Geometrija več pogledov

Geometrija več pogledov ni pomembna le zato, ker omogoča rekonstrukcijo opazovanega prizora, temveč tudi zaradi lastnosti, ki jih lahko s pridom izkoristimo pri razvoju algoritmov strojnega vida (npr. pri iskanju parov točk med slikami in pri ocenjevanju lege kamere na podlagi pogleda kamere). Predpostavimo, da rotacijska matrika $\boldsymbol{R}_{C_2}^{C_1}$ in translacijski vektor $\boldsymbol{t}_{C_2}^{C_1}$ opisujeta relativno lego med dvema kamerama (slika 5.6). Če optični središči obeh kamera ne sovpadata ($\boldsymbol{t}_{C_2}^{C_1} \neq \mathbf{0}$), lahko po kratki matematični manipulaciji pridemo do izraza (5.27)



Slika 5.6: Perspektivična geometrija pogledov dveh kamer

(predpostavljamo, da sta notranja modela kamer enaka)

$$\begin{aligned} \boldsymbol{p}_{C_1} &= \boldsymbol{R}_{C_2}^{C_1}\boldsymbol{p}_{C_2} + \boldsymbol{t}_{C_2}^{C_1} \qquad (5.26)\\ \boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_1} &\propto \boldsymbol{R}_{C_2}^{C_1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2} + \boldsymbol{t}_{C_2}^{C_1}\\ [\boldsymbol{t}_{C_2}^{C_1}]_\times\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_1} &\propto [\boldsymbol{t}_{C_2}^{C_1}]_\times\boldsymbol{R}_{C_2}^{C_1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2}\\ 0 &= \underline{\boldsymbol{p}}_{P_1}^T\boldsymbol{S}^{-T}[\boldsymbol{t}_{C_2}^{C_1}]_\times\boldsymbol{R}_{C_2}^{C_1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2}\\ 0 &= \underline{\boldsymbol{p}}_{P_1}^T\boldsymbol{F}\underline{\boldsymbol{p}}_{P_2} \qquad (5.27) \end{aligned}$$

Vektorski produkt vektorjev $\boldsymbol{a}^T = [a_1\ a_2\ a_3]$ in $\boldsymbol{b}^T = [b_1\ b_2\ b_3]$ smo zapisali kot $\boldsymbol{a}\times\boldsymbol{b} = [\boldsymbol{a}]_\times\boldsymbol{b}$, kjer je $[\boldsymbol{a}]_\times$ antisimetrična matrika

$$[\boldsymbol{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

Matrika $\boldsymbol{F}$ je znana kot *fundamentalna matrika* in opisuje *epipolarno omejitev* (5.27): točka $\underline{\boldsymbol{p}}_{P_1}$ leži na premici $\boldsymbol{F}\underline{\boldsymbol{p}}_{P_2}$ na prvi sliki in točka $\underline{\boldsymbol{p}}_{P_2}$ leži na premici $\boldsymbol{F}^T\underline{\boldsymbol{p}}_{P_1}$ na drugi sliki. Pomembna je tudi relacija $\boldsymbol{p}_{C_1}^T\boldsymbol{E}\boldsymbol{p}_{C_2} = 0$, kjer je matrika $\boldsymbol{E} = [\boldsymbol{t}_{C_2}^{C_1}]_\times\boldsymbol{R}_{C_2}^{C_1}$ na področju strojnega vida znana kot *esencialna matrika*. Povezava med esencialno in fundamentalno matriko je $\boldsymbol{E} = \boldsymbol{S}^T\boldsymbol{F}\boldsymbol{S}$.

Epipolarno omejitev lahko izkoristimo za izboljšanje iskanja parov točk med slikama istega prizora iz dveh zornih kotov, če je znana medsebojna lega med kalibriranima kamerama. Ker mora korespondenčni par točke $\underline{\boldsymbol{p}}_{P_1}$ na prvi sliki ležati na premici $\boldsymbol{F}^T\underline{\boldsymbol{p}}_{P_1}$ na drugi sliki, se iskanje para na 2D-ravnini slike skrči na iskanje vzdolž 1D-premice (epipolarne premice). Zato je možna velika pohitritev iskanja parov točk in tudi iskanje parov je lahko bolj robustno, saj lahko zavržemo pare, ki ne zadostijo epipolarni omejitvi.

Primer 5.5

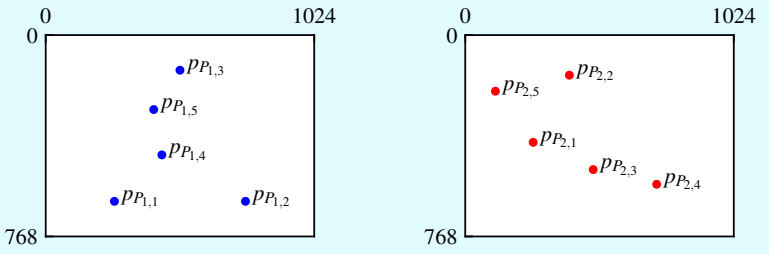
Z dvema kamerama opazujemo množico 3D-točk v prostoru. Medsebojna lega kamer je podana z rotacijsko matriko $\boldsymbol{R}_{C_2}^{C_1} = \boldsymbol{R}_x(30°)\boldsymbol{R}_y(60°)$ in translacijskim vektorjem $\boldsymbol{t}_{C_2}^{C_1} = [4\ \ -1\ 2]^T$. Notranji parametri modela (5.25) obeh kamer so enaki: $\alpha_x f = \alpha_y f = 1000$, brez striga ($\gamma = 0$) in optična os poteka skozi središče slike z dimenzijami 1024 krat 768. Množica točk, ki nastane na zaslonu prve kamere, je

$$\boldsymbol{p}_{P_1}^T \in \{[262\ 634],\ [762\ 634],\ [512\ 134],\ [443\ 457],\ [412\ 284]\}$$

Množica točk, ki nastane na zaslonu druge kamere, je

$$\boldsymbol{p}_{P_1}^T \in \{[259\ 409],\ [397\ 153],\ [488\ 513],\ [730\ 569],\ [115\ 214]\}$$

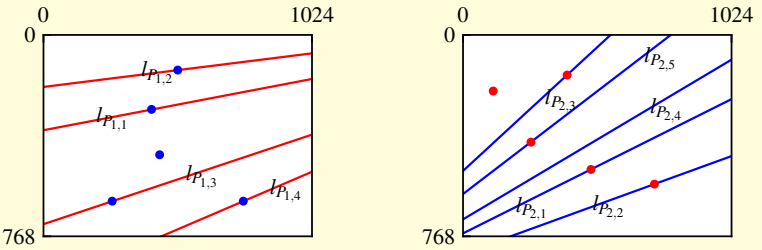
Točke, ki nastanejo na slikovnih ravninah obeh kamer, so prikazane na sliki 5.7. Določite vse možne korespondenčne pare točk, ki zadostijo epipolarni omejitvi. Za vse najdene korespondenčne pare točk rekonstruirajte položaj 3D-točk v prostoru glede na koordinatna sistema obeh kamer.



Slika 5.7: Točke na dveh slikovnih ravninah iz primera 5.5

Rešitev

Pri iskanju parov točk med dvema slikama iz kamer, katerih medsebojna lega je znana, velja, da morajo projekcije točk, ki pripadajo isti 3D-točki, zadostiti epipolarni omejitvi (5.27). Enačba (5.27) je oblike $\boldsymbol{l}^T\underline{\boldsymbol{p}} = 0$, tj. homogena točka $\underline{\boldsymbol{p}}^T = [x\ y\ 1]$ leži na premici $\boldsymbol{l}^T = [a\ b\ c]$, saj je $ax + by + c = 0$ implicitna oblika enačbe premice. S pomočjo epipolarne omejitve (5.27) lahko torej najdemo premico $\boldsymbol{l}_{P_2}$ na drugi sliki za vsako točko $\underline{\boldsymbol{p}}_{P_1}$ na prvi sliki, in obratno, premico $\boldsymbol{l}_{P_1}$ za vsako točko $\underline{\boldsymbol{p}}_{P_2}$. Epipolarne premice za prvo in drugo sliko so prikazane na sliki 5.8. Glede na sliki 5.7 in 5.8 lahko grafično najdemo štiri možne pare točk.



Slika 5.8: Točke in epipolarne premice na dveh slikah iz primera 5.5

Implementacija algoritma za iskanje parov točk je podana v programu 5.2 (funkciji `rotX` in `rotY` sta Matlabovi implementaciji enačb (5.1) in (5.2)). Pari točk so zbrani v indeksni matriki `pairs`.

Program 5.2: Iskanje parov točk iz primera 5.5

./src/sen/example_fundamental.m

```
% Notranji parametri kamere in velikost zaslona
alphaF = 1000; % alpha*f, v px/m
s = [1024; 768]; % Velikost zaslona, v px
c = s/2; % Optično središče na sliki, v px
S = [alphaF, 0, c(1); 0, alphaF, c(2); 0, 0, 1]; % Model kamere

% Lega med kamerama
R = rotX(pi/6)*rotY(pi/3); t = [4; -1; 2];

% Množica točk
pP1 = [262, 634; 762, 634; 512, 134; 443, 457; 412, 284].';
pP2 = [259, 409; 397, 153; 488, 513; 730, 569; 115, 214].';

N1 = size(pP1, 2); N2 = size(pP2, 2); % Število točk

% Fundamentalna matrika
tx = [0, -t(3), t(2); t(3), 0, -t(1); -t(2), t(1), 0];
F = S.'\tx*R/S;

epsilon = 1e-2; % Toleranca napake po razdalji
pP1 = [pP1; ones(1,N1)]; pP2 = [pP2; ones(1,N2)]; % Točke v homogenih koordinatah

% Epipolarne premice v k. s. P1, ki pripadajo točkam v k. s. P2
lP1 = F*pP2;
% Epipolarne premice v k. s. P2, ki pripadajo točkam v k. s. P1
lP2 = F.'*pP1;

% Iskanje parov točk (upoštevanje epipolarne omejitve)
pairs = [];
for i = 1:N1
    d = abs(lP2(:,i).'*pP2);
    k = find(d<epsilon);
    if ~isempty(k), pairs = [pairs, [i; k(1)]]; end
end
pairs
```

```
pairs =
     1     2     3     5
     3     4     2     1
```

Singularni primeri Enačba (5.27) postane singularna, če optični središči obeh kamer sovpadata. V primeru, če je $\boldsymbol{t}_{C_2}^{C_1}$ nič, lahko iz (5.26) izpeljemo relacijo

$$\underline{\boldsymbol{p}}_{P_1} \propto \boldsymbol{S}\boldsymbol{R}_{C_2}^{C_1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2} \tag{5.28}$$

Do podobne oblike enačbe pridemo tudi v primeru, če se vse točke v 3D-prostoru nahajajo le na eni ravnini. Brez izgube na splošnosti lahko predpostavimo, da je to ravnina $z_W = 0$

$$\underline{\boldsymbol{p}}_{P_1} \propto \boldsymbol{S}[\boldsymbol{r}_{W,1}^{C_1}\ \boldsymbol{r}_{W,2}^{C_1}\ \boldsymbol{t}_W^{C_1}][\boldsymbol{r}_{W,1}^{C_2}\ \boldsymbol{r}_{W,2}^{C_2}\ \boldsymbol{t}_W^{C_2}]^{-1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2} \tag{5.29}$$

kjer $\boldsymbol{R}_W^C = [\boldsymbol{r}_{W,1}^C\ \boldsymbol{r}_{W,2}^C\ \boldsymbol{r}_{W,3}^C]$. Preslikava ravnine v svetovnih koordinatah na slikovno ravnino je

$$\underline{\boldsymbol{p}}_P \propto \boldsymbol{S}[\boldsymbol{r}_{W,1}^C\ \boldsymbol{r}_{W,2}^C\ \boldsymbol{t}_W^C]\underline{\boldsymbol{p}}_W \tag{5.30}$$

kjer v tem primeru velja $\underline{\boldsymbol{p}}_W^T = [x_W\ y_W\ 1]$. Enačbe (5.28), (5.29) in (5.30) imajo vse podobno obliko: $\underline{\boldsymbol{p}}' \propto \boldsymbol{H}\underline{\boldsymbol{p}}$. Matrika $\boldsymbol{H}$ je na področju strojnega vida znana kot *homografija*.

3D-rekonstrukcija

V primeru stereo kamere je položaj 3D-točke možno določiti na podlagi obeh projekcij (slik) točke. Postopek zahteva določitev korespondenčnega para točk na slikah, ki ustrezata 3D-točki v prostoru, kar je eden izmed fundamentalnih problemov na področju strojnega vida. Ko je korespondenčni par točk najden in če je znana lege med obema kamerama (translacija med kamerama ne sme biti nič) ter če sta znana še notranja modela obeh kamer, lahko določimo položaj točke v 3D-prostoru. Če sta oba notranja modela kamer enaka, lahko ocenimo globino točke v koordinatnih sistemih obeh kamer (z_{C_1} in z_{C_2}), če rešimo naslednji sistem enačb (npr. z uporabo metode najmanjših kvadratov)

$$\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_1} z_{C_1} - \boldsymbol{R}_{C_2}^{C_1}\boldsymbol{S}^{-1}\underline{\boldsymbol{p}}_{P_2} z_{C_2} = \boldsymbol{t}_{C_2}^{C_1} \tag{5.31}$$

Primer 5.6

Za vse najdene korespondenčne pare točk na slikah iz primera 5.5 določite položaje točk v 3D-prostoru glede na koordinatni sistem prve in druge kamere.

Rešitev

Položaje točk v 3D-prostoru lahko določimo, če rešimo sistem enačb (5.31) za vsak korespondenčni par točk na slikah iz primera 5.5. Rešitev vstavimo v inverzno transformacijo enačbe (5.24). Implementacija rešitve v okolju Matlab je podana v programu 5.3. Položaji točk v 3D-prostoru glede na koordinatni sistem prve in druge kamere so prirejeni spremenljivkam `pC1` in `pC2`.

Program 5.3: Rekonstrukcija točk v 3D-prostoru iz primera 5.6

./src/sen/example_reconstruct.m

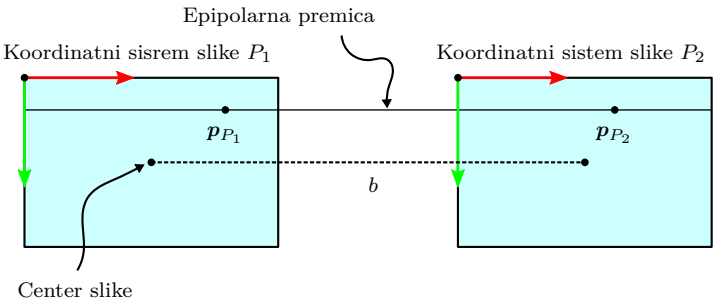
```
% Rekonstrukcija
M = size(pairs, 2);
pC1 = zeros(3,M);
for i = 1:M
    a = pairs(1,i); b = pairs(2,i);
    c1 = S\pP1(:,a);
    c2 = -R*(S\pP2(:,b));
    psi = [c1, c2]\t;
    pC1(:,i) = psi(1)*c1;
end
pC2 = R.'*(pC1-repmat(t, 1, M));
pC1, pC2
```

```
pC1 =
   -0.9989    0.9994         0   -0.2999
    0.9989    0.9994   -1.0005   -0.2999
    3.9956    3.9978    4.0018    2.9994
pC2 =
   -0.1372    0.8638   -0.4988   -1.0973
    0.7333    0.7327   -1.0013    0.1066
    5.6930    3.9635    4.3308    4.3316
```

Problem rekonstrukcije se poenostavi v primeru kanonične postavitve stereo kamere, kjer je prva kamera glede na drugo le premaknjena vzdolž osi x za razdaljo b (kot je prikazano na sliki 5.9). V tem primeru postanejo epipolarne premice vzporedne in epipolarna premice točke $\boldsymbol{p}_{P_1}$ gre tudi skozi to točko (ne le točko $\boldsymbol{p}_{P_2}$ na drugi sliki). V primeru digitalnih slik to pomeni, da se par točke na prvi sliki nahaja na drugi sliki v isti vrstici kot na prvi sliki. Predpostavimo, da so parametri modelov (5.25) obeh kamer enaki: $\alpha_x = \alpha_y = \alpha$ in $\gamma = 0$. Položaj točke v 3D-prostoru glede na koordinatni sistem prve kamere lahko dobimo iz obeh njenih projekcij (slik)

$$\boldsymbol{p}_{C_1}^T = \frac{b}{d}\begin{bmatrix} x_{P_1} - c_x & y_{P_1} - c_y & \alpha f \end{bmatrix} \tag{5.32}$$

kjer smo uvedli *dispariteto* $d = x_{P_1} - x_{P_2}$. Dispariteta vsebuje informacijo o globini, kot je razvidno iz zadnjega elementa v vektorju (5.32): $z_{C_1} = \alpha f b d^{-1}$. Za vse točke, ki se nahajajo pred kamero, je dispariteta pozitivna: $d \geq 0$.



Slika 5.9: Kanonična postavitev stereo kamere

Primer 5.7

Predpostavimo kanonično postavitev stereo kamere z razdaljo med kamerama $b = 0{,}2$ in naslednjimi notranjimi parametri obeh kamer: $\alpha_x f = \alpha_y f = 1000$, $c_x = 512$ in $c_y = 384$. Določite položaj 3D-točke, ki se projicira v točko $\boldsymbol{p}_{P_1}^T = [351\ 522]$ na prvi sliki in v točko $\boldsymbol{p}_{P_2}^T = [236\ 522]$ na drugi sliki.

Rešitev

Ker imamo opravka s kanonično postavitvijo stereo kamere, lahko rešitev dobimo direktno z uporabo (5.32) (glejte program 5.4).

Program 5.4: Rekonstrukcija 3D-točke iz primera 5.7

./src/sen/example_reconstruct0.m

```
% Notranji parametri kamere in velikost zaslona
alphaF = 1000; % alpha*f, v px/m
c = [512; 384]; % Optično središče na sliki, v px
S = [alphaF, 0, c(1); 0, alphaF, c(2); 0, 0, 1]; % Model kamere

b = 0.2; % Razdalja med kamerama, v m

% Pari točk
pP1 = [351; 522];
pP2 = [236; 522];

% 3D-rekonstrukcija
d = pP1(1)-pP2(1);
pC1 = b/d*[pP1-c; alphaF]
```

```
pC1 =
   -0.2800
    0.2400
    1.7391
```

5.3 Metode merjenja lege

V nadaljevanju bomo predstavili glavne metodologije uporabe senzorjev za ocenjevanje lege robota v okolju. Ti pristopi lahko merijo *relativno spremembo lege* glede na predhodno določeno lego ali pa *absolutno lego* glede na nek referenčen koordinatni sistem.

5.3.1 Relativno določanje lege

Relativno določanje lege (angl. *dead reckoning*) ocenjuje trenutno lego robota s pomočjo prejšnje znane lege in izmerjenih relativnih premikov iz prejšnje lege. Ti premiki ali inkrementi gibanja (razdalja in orientacija) se izračunajo iz izmerjenih hitrosti v pretečenem času in smeri gibanja. Za te pristope je skupna uporaba integracije poti za oceno trenutne lege, zato se običajno pojavijo različni pogreški (pogrešek integracijske metode, merilni pogrešek, pristranskost, šum itd.).

V mobilni robotiki sta najpogosteje uporabljena pristopa *odometrija* in *inercialni navigacijski sistem*.

Odometrija

Odometrija ocenjuje lego robota s pomočjo integracije premikov, ki jih lahko izmerimo ali pridobimo iz uporabljenih regulirnih veličin za gibanje. V mobilni robotiki običajno pridobimo relativne premike iz osnih senzorjev (npr. inkrementalni enkoder), ki so pritrjeni na osi koles robota. Z uporabo notranjega kinematičnega modela (glejte (2.1) za diferencialni pogon) so meritve zasuka koles povezane s spremembami pozicije in orientacije mobilnega robota. Spremembe pozicije in orientacije v določenem časovnem intervalu med zaporednima meritvama lahko izrazimo s hitrostmi robota. V nekaterih primerih se lahko kotne hitrosti koles izmerijo neposredno ali pa jih izrazimo iz znanih reguliranih hitrosti (predpostavimo, da so hitrostni regulatorji točni).

Vzemimo robota z diferencialnim pogonom, ki ima na kolesih nameščena inkrementalna enkoderja. Senzorja merita relativno spremembo zasuka levega $\Delta\alpha_L(t)$ in desnega kolesa $\Delta\alpha_R(t)$ glede na (prejšnjo) orientacijo v času $t^- = t - \Delta t$. Če predpostavimo idealno vrtenje koles (brez zdrsov koles ipd.), je njuna prevožena razdalja

$$\Delta d_L(t) = \Delta\alpha_L(t)R \qquad \Delta d_R(t) = \Delta\alpha_R(t)R$$

kjer je R polmer koles. Z uporabo notranjega kinematičnega modela (2.1) sta sprememba orientacije in prevožena razdalja (premik)

$$\Delta\varphi(t) = \frac{\Delta d_R(t) - \Delta d_L(t)}{L}$$
$$\Delta d(t) = \frac{\Delta d_R(t) + \Delta d_L(t)}{2}$$

kjer je L razdalja med kolesoma.

Lego robota lahko ocenimo iz njegovih izmerjenih hitrosti s pomočjo integracije zunanjega kinematičnega modela (2.2) ali iz vsote izračunane pozicije robota in sprememb orientacije. Z uporabo trapezne metode integracije dobimo ocenjeno lego robota

$$\begin{aligned} x(t) &= x(t^-) + \Delta d(t) \cos\left(\varphi(t^-) + \frac{\Delta\varphi(t)}{2}\right) \\ y(t) &= y(t^-) + \Delta d(t) \sin\left(\varphi(t^-) + \frac{\Delta\varphi(t)}{2}\right) \\ \varphi(t) &= \varphi(t^-) + \Delta\varphi(t) \end{aligned}$$

Vendar pa zaradi integralne narave odometrije pride do kumulativnega pogreška (lezenja), ki ga v glavnem delimo na sistematične in nedeterministične pogreške. *Sistematični pogreški* se pojavijo zaradi kinematičnih nepravilnosti (npr. napačen podatek za polmer koles) ter (ne)točnosti uporabljene integracijske metode in meritve (neznana pristranskost). *Nedeterministični pogreški* pa so posledica zdrsa koles, šuma meritve ipd. Zato je odometrija samostojno uporabna le za kratkoročno ocenjevanje pri znani začetni legi. Pogosteje se uporablja v kombinaciji z meritvami absolutnih senzorjev za napovedovanje in filtriranje absolutnih meritev lege. Tako dobimo boljše ocene lege.

V kolesnih mobilnih sistemih za odometrijo pogosto uporabljamo senzorje, ki jih pritrdimo na os kolesa (npr. inkrementalni optični dajalniki, potenciometri) in merijo kot zasuka ali kotno hitrost.

Inercialna navigacija

Inercialni navigacijski sistem (INS) je samostojna tehnika za oceno lege, orientacije in hitrosti vozila s pomočjo relativnega merjenja položaja. INS vključuje senzorje gibanja (pospeškometer) in merilnike zasuka (žiroskop), kjer sta pozicija in orientacija vozila ocenjeni glede na znano začetno lego.

Meritev pospeškometra in žiroskopa predstavljata tridimenzionalna vektorja pospeškov in kotnih hitrosti v prostoru. Za oceno lege in orientacije robota je potrebna dvojna integracija meritve pospeška in enojna integracija meritve kotne hitrosti. Uporaba integracije je glavni vzrok za položajni pogrešek v INS, saj se ob integraciji akumulirajo konstantni (sistematični) pogreški (lezenje senzorja, slaba kalibracija itd.). Zaradi stalnega pogreška napaka ocenjevanja pozicije narašča kvadratično s časom, napaka ocene orientacije pa narašča linearno s časom. Poleg tega je napaka ocenjevanja pozicije odvisna od napake ocenjevanja orientacije, ker pospeškometer meri celotni pospešek, torej tudi gravitacijo. Da lahko ocenimo pospešek vozila, moramo od meritve odšteti gravitacijski pospešek, kar zahteva natančno poznavanje orientacije vozila. Vsak pogrešek orientacije (nagnjena podlaga) povzroči napačni navidezni pogrešek zaradi gravitacije, kar je še posebej problematično, ker so pospeški vozila običajno veliko manjši od

gravitacije. Pri majhnih pospeških povzroča dodatne težave relativno velik šum (majhna vrednost razmerja signal-šum). Ti vplivi so jasno razvidni iz modela meritve pospeškometra, ki je sestavljen iz translacijskega pospeška vozila $\boldsymbol{a}$, gravitacije $\boldsymbol{g} = [0,\ 0,\ 0{,}981]^T$ in radialnega pospeška

$$\boldsymbol{a}_m = \boldsymbol{R}_i^{wT}(\boldsymbol{a} + \boldsymbol{R}_e^w \boldsymbol{g} + \boldsymbol{\omega} \times \boldsymbol{v}) + \boldsymbol{a}_{bias} + \boldsymbol{a}_{noise} \tag{5.33}$$

kjer je $\boldsymbol{a}_m$ izmerjeni pospešek v lokalnih koordinatah senzorja, $\boldsymbol{R}_e^w$ rotacijska matrika med koordinatnim sistemom Zemlje (ECEF) in globalnim koordinatnim sistemom, v katerem sledimo legi, $\boldsymbol{R}_i^w$ rotacijska matrika iz lokalnega koordinatnega sistema INS v globalni koordinatni sistem, $\boldsymbol{\omega}$ kotna hitrost v globalnih koordinatah, $\boldsymbol{v}$ translacijska hitrost, $\boldsymbol{a}_{bias}$ pristranskost pospeška in $\boldsymbol{a}_{noise}$ šum.

Orientacijo ocenimo s pomočjo meritev žiroskopa. Model meritve žiroskopa je

$$\boldsymbol{\omega}_m = \boldsymbol{\omega}_i + \boldsymbol{\omega}_{bias} + \boldsymbol{\omega}_{noise} \tag{5.34}$$

kjer je $\boldsymbol{\omega}_m$ vektor izmerjene kotne hitrosti, $\boldsymbol{\omega}_i$ pravi vektor kotnih hitrosti telesa v lokalnih koordinatah, $\boldsymbol{\omega}_{bias}$ je pristranskost senzorja in $\boldsymbol{\omega}_{noise}$ šum senzorja.

Oceno orientacije INS-enote dobimo s pomočjo ocenjenih enotskih kotnih hitrosti iz (5.34) kot $\boldsymbol{\omega}_i = \boldsymbol{\omega}_m - \boldsymbol{\omega}_{bias}$. Pristranski del se ponavadi oceni sproti s pomočjo nekega ocenjevalnika (npr. Kalmanov filter) ali pa predpostavimo, da je konstanten za kratke ocenjevalne intervale, kakovostni žiroskop in začetno kalibracijo (ocena hitrosti $\boldsymbol{\omega}_{bias}$). Vendar se pristranskost spreminja s časom, zato je slednji pristop primeren samo za kratkoročno uporabo. Najenostavnejšo kalibracijo izvedemo s povprečenjem N meritev žiroskopa, medtem ko držimo INS-enoto v konstantnem položaju ($\boldsymbol{\omega}_{bias} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\omega}_m$, ko $\boldsymbol{\omega}_i = \boldsymbol{0}$).

S pomočjo rotacijske kinematike za kvaternione (5.18) dobimo relativno oceno orientacije enote INS glede na začetno orientacijo kot

$$\frac{\mathrm{d}\boldsymbol{q}_w^i(t)}{\mathrm{d}t} = \frac{1}{2}\boldsymbol{\Omega}(t)\boldsymbol{q}_w^i(t)$$

$$\boldsymbol{q}_w^i(t) = \int_0^t \frac{\mathrm{d}\boldsymbol{q}_w^i(t)}{\mathrm{d}t}\,\mathrm{d}t + \boldsymbol{q}_w^i(0) \tag{5.35}$$

kjer je $\boldsymbol{q}_w^i(t)$ kvaternion, ki opisuje rotacijo iz globalnega koordinatnega sistema w v koordinanti sistem INS-enote i, $\boldsymbol{q}_w^i(0)$ začetna orientacija

$$\boldsymbol{\Omega}(t) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

in $\boldsymbol{\omega}_i(t) = [\omega_x,\ \omega_y,\ \omega_z]^T$. Rotacijsko matriko $\boldsymbol{R}_w^i = \boldsymbol{R}_i^{wT}$ pridobimo iz razmerja (5.8). V (5.35) lahko uporabimo numerično integracijo (5.17).

Da lahko ocenimo lego INS-enote, izračunamo translacijski pospešek v globalnih koordinatah iz (5.33)

$$\boldsymbol{a} = \boldsymbol{R}_i^w(\boldsymbol{a}_m - \boldsymbol{a}_{bias}) - \boldsymbol{R}_e^w \boldsymbol{g} - \boldsymbol{\omega} \times \boldsymbol{v}$$

kjer je $\boldsymbol{\omega} = \boldsymbol{R}_i^w \boldsymbol{\omega}_i$ vektor kotne hitrosti v globalnih koordinatah, izraz za pristranskost $\boldsymbol{a}_{bias}$ pa se ocenjuje sproti s pomočjo nekega ocenjevalnika (npr. Kalmanov filter). Potrebna je tudi ustrezna kalibracija pospeškometra. Oceno hitrosti $\boldsymbol{v}(t)$ in pozicije $\boldsymbol{x}(t)$ dobimo z integracijo

$$\boldsymbol{v}(t) = \int_0^t \boldsymbol{a}(t)\,\mathrm{d}t + \boldsymbol{v}(0)$$
$$\boldsymbol{x}(t) = \int_0^t \boldsymbol{v}(t)\,\mathrm{d}t + \boldsymbol{x}(0)$$

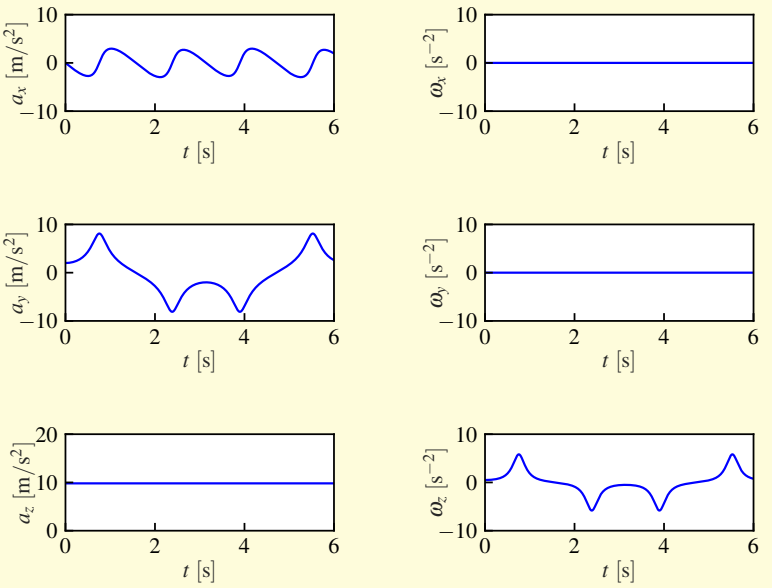
Primer 5.8

Izvedite simulacijo izmerjenega pospeška in kotnih hitrosti za robota z diferencialnim pogonom in INS-enoto, ki se vozi po krivulji $x(t) = cos(t)$, $y(t) = sin(2t)$ in $z(t) = 0$ (v globalnem koordinatnem sistemu). Čas simulacije je 6 s, računski korak pa 1 ms. INS-enota je usmerjena tako, da je njena x-os tangenta na trajektorijo, y-os je pravokotna na x-os in z-os je poravnana z z-osjo globalnega koordinatnega sistema.

Iz meritev ocenite pozicijo in orientacijo INS-enote v globalnem koordinatnem sistemu. Poleg tega upoštevajte pristranskost senzorjev in šum ter opazujte, kako vplivata na ocenjeno lego INS-enote.

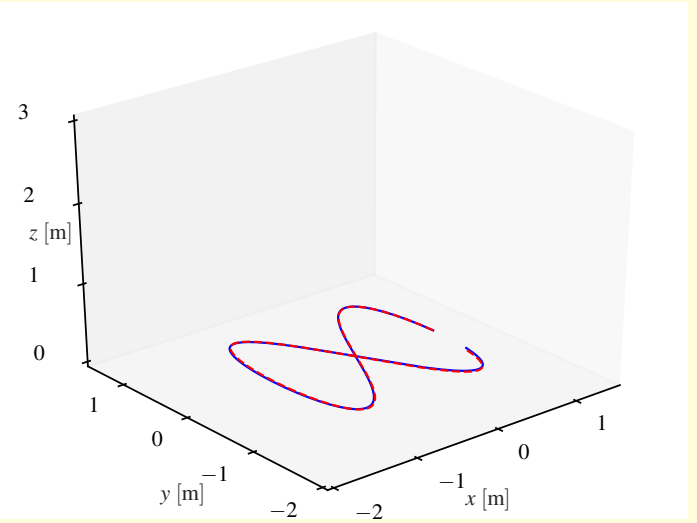
Rešitev

V simulaciji dobimo kotne hitrosti INS-enote iz razmerja (5.22) in jih uporabimo za matriko kotnih hitrosti $\boldsymbol{\Omega}' = \frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}t}\boldsymbol{\Omega}'\boldsymbol{R}^T$. Simulacije meritev pospeškometra in žiroskopa so modelirane s pomočjo (5.33) in (5.34) ter prikazane na sliki 5.10.

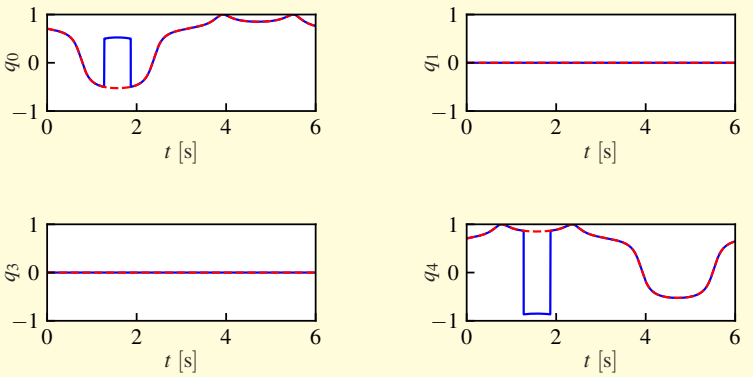


Slika 5.10: Simulacija meritev pospeška (levi stolpec) in kotne hitrosti (desni stolpec) INS-enote

Ocenjena pozicija in orientacije INS-enote v idealnem primeru brez šuma in pristranskosti sta prikazani na sliki 5.11.



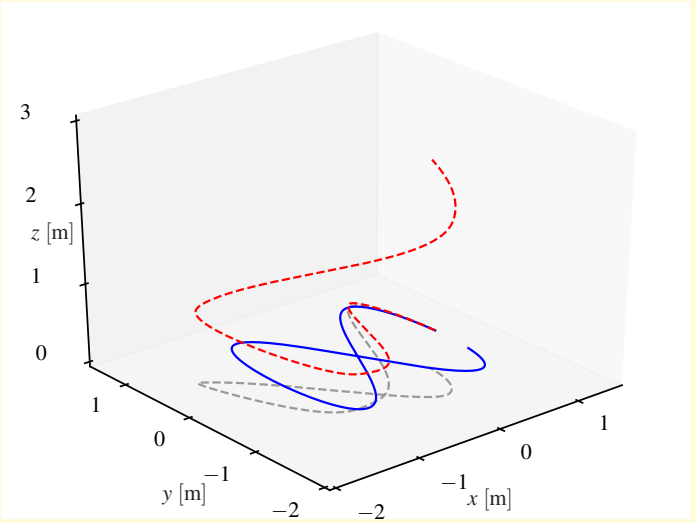
(a)



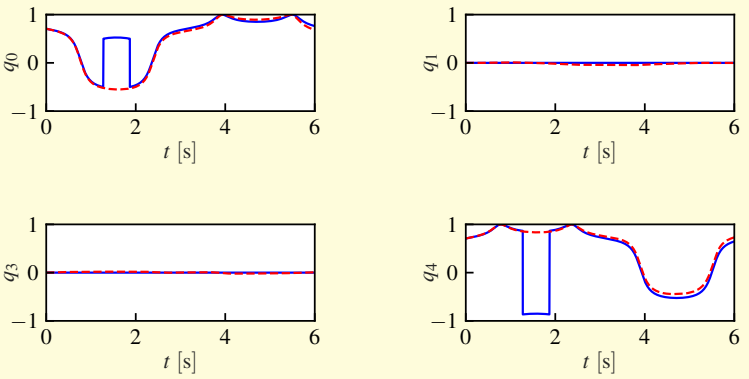
(b)

Slika 5.11: Ocenjena pozicija (a) in orientacija (b) INS-enote iz meritev pospeškometra in žiroskopa v idealnih razmerah, brez šuma in pristranskosti (prava vrednost je označena s polno krivuljo, ocena pa s črtkano krivuljo). Na koncu simulacije lahko opazimo manjši pozicijski pogrešek zaradi numerične integracije.

Ocenjena lega v primeru šumnih meritev in vključene pristranskosti pa je prikazana na sliki 5.12, kjer je mogoče opaziti hitro rast pozicijskega pogreška.



(a)



(b)

Slika 5.12: Ocenjena pozicija (a) in orientacija (b) INS-enote iz meritev pospeškometra in žiroskopa z upoštevanim šumom in pristranskostjo senzorja (prava vrednost je označena s polno krivuljo, ocena pa s črtkano krivuljo). Ker pristranskost ni kompenzirana, se pojavi velik pogrešek ocene pozicije in majhen pogrešek ocene orientacije.

V programu 5.5 je podana koda za simulacijo INS-enote in oceno njene lege (funkcije `rotX`, `rotY` in `rotZ` so Matlabove izvedbe funkcij (5.1), (5.2) in (5.3)).

Program 5.5

./src/sen/example_inertial_sensors_navigation.m

```
biasA = [1; 1; 1]*0.02; biasW = [1; 1; 1]*0.04; % Sistematična napaka senzorja
SigmaA = 0.1; SigmaW = 0.05; % Šum senzorja

nSteps = 6000; dT = 0.001; t = 0; % Število vzorcev in velikost koraka

for k = 1:nSteps
    % Simulacija gibanja senzorja: prava lega in pravi pospeški
    x = [cos(t); sin(2*t); 0];    % Lega
    v = [-sin(t); 2*cos(2*t); 0]; % Hitrost
    a = [-cos(t);-4*sin(2*t); 0]; % Pospešek

    fi = [0; 0; atan2(v(2), v(1))]; % Eulerjevi koti glede na svetovni k. s.
    dfi = [0; 0; (v(1)*a(2) - v(2)*a(1))/(v(1)^2 + v(2)^2)]; % Odvod
    Rx = rotX(fi(1)); Ry = rotY(fi(2)); Rz = rotZ(fi(3));
    dRx = [0, 0, 0; 0,-sin(fi(1)), cos(fi(1)); 0,-cos(fi(1)),-sin(fi(1))];
    dRy = [-sin(fi(2)), 0,-cos(fi(2)); 0, 0, 0; cos(fi(2)), 0,-sin(fi(2))];
    dRz = [-sin(fi(3)), cos(fi(3)), 0;-cos(fi(3)),-sin(fi(3)), 0; 0, 0, 0];
    R = Rx*Ry*Rz;
    dR = dRx*Ry*Rz*dfi(1) + Rx*dRy*Rz*dfi(2) + Rx*Ry*dRz*dfi(3);
    q = dcm2quat(R).'; % Kvaternion med svetovnim k. s. in senzorjem

    % Meritve žiroskopa
    Omega = dR*R.'; % Zapis odvoda kotne hitorsti v matrični obliki
    wb = -[Omega(3,2); Omega(1,3); -Omega(1,2)]; % Kotne hitrosti

    % Meritve pospeškometra
    agDyn = a; % Dinamični pospešek v svetovnem k. s.
    agGrav = [0; 0; 9.81]; % Gravitacija
    Rearth = eye(3); % Globalni k. s. sovpada s k. s. Zemlje
    wg = R.'*wb; % Prave kotne hitrosti v svetovnem k. s.
    wgSkew = [0 -wg(3) wg(2); wg(3) 0 -wg(1); -wg(2) wg(1) 0];
    vg = v; % Hitrosti v svetovnem k. s.

    % Izmerjene kotne hitrosti in pospeški
    wbMea = wb + biasW + randn(3,1)*SigmaW;
    abMea = R*(agDyn + Rearth*agGrav + wgSkew*vg) +biasA+randn(3,1)*SigmaA;

    % Inercialna navigacija
    if k==1 % Inicializacija
        qEst = q; xEst = x; vEst = v; % Inicializacija začetnih vrednosti
    else % Posodobitev
        % Žiroskop
        wx = wbMea(1); wy = wbMea(2); wz = wbMea(3);
        OMEGA = [ 0 -wx -wy -wz; ...
                  wx   0  wz -wy; ...
                  wy -wz   0  wx; ...
                  wz  wy -wx   0];
        dQest = 0.5*OMEGA*qEst;
        qEst = qEst + dQest*dT; % Integracija kvaternionov
        qEst = qEst/norm(qEst); % Normiranje kvaternionov
        % Pospešek
        agGrav = [ 0; 0; 9.81]; % Gravitacija
        R_ = quat2dcm(qEst.'); % Lega senzorja glede na svetovni k. s.
        Rearth = eye(3); % Svetovni k. s. sovpada s k. s. Zemlje
        wg_ = R_.'*[wx; wy; wz]; % Kotne hitrosti v svetovnem k. s.
        wgSkew_ = [0 -wg_(3) wg_(2); wg_(3) 0 -wg_(1); -wg_(2) wg_(1) 0];
        Aest = R_.'*abMea - Rearth*agGrav - wgSkew_*vEst; % Ocena pospeška
        vEst = vEst + Aest*dT; % Ocena hitrosti
        xEst = xEst + vEst*dT; % Ocena lege
    end
```

```
61      t = t + dT;
62  end
```

5.3.2 Merjenje smeri gibanja

Sistemi za merjenje smeri podajajo informacijo o orientaciji vozila v prostoru, oz. v katero smer je vozilo usmerjeno. Za oceno smeri običajno uporabimo več senzorjev, kot so magnetometer, žiroskop in pospeškometer. Njihove informacije so vgrajene v senzorske sisteme za oceno smeri (npr. žiroskop, kompas ali inklinometer).

Magnetometer in pospeškometer zagotavljata absolutne meritve tridimenzionalnih smernih vektorjev zemeljskega magnetnega polja (jakost in smer) ter smernega vektorja gravitacije Zemlje (če senzorska enota ne pospešuje). Za zmanjšanje šuma meritve in izboljšanje točnosti, so v ocenjevalne filtre vključene tudi relativne meritve žiroskopa.

Za oceno orientacije senzorske enote glede na nek referenčni koordinatni sistem (npr. fiksni zemeljski koordinatni sistem) sta potrebna vsaj dva smerna vektorja. Senzorsko enoto lahko sestavljata magnetometer in pospeškometer. Merilni model magnetometra zapišemo kot

$$\boldsymbol{b}_m = \boldsymbol{R}_w^i \boldsymbol{R}_e^w \boldsymbol{b}_{true} + \boldsymbol{b}_{bias} + \boldsymbol{b}_{noise}$$

kjer je $\boldsymbol{b}_m$ izmerjeno magnetno polje v koordinatnem sistemu senzorja, $\boldsymbol{b}_{true}$ magnetno polje Zemlje v zemeljskem koordinatnem sistemu za neko mesto na Zemlji, rotacijski matriki pa sta definirani enako kot v (5.33). Magnetno polje $\boldsymbol{b}_{true}$ lahko aproksimiramo kot konstanto za nekaj majhnih površin na Zemlji (npr. $100\,\mathrm{km}^2$). Model pospeškometra je podan v (5.33). V primeru enakomernega gibanja kaže smerni vektor meritve v smeri gravitacije in z-osi fiksnega zemeljskega koordinatnega sistema.

Orientacija senzorske enote glede na fiksni zemeljski koordinatni sistem je opisana z rotacijsko matriko $\boldsymbol{R}_e^i = \boldsymbol{R}_i^{wT} \boldsymbol{R}_e^w$, ki jo dobimo z zapisom matrike vektorjev osi zemeljskega koordinatnega sistema, izraženih v lokalnem koordinatnem sistemu. V primeru mirovanja pospeškometer izmeri smerni vektor, ki kaže od središča Zemlje proti poziciji INS. Ta smer torej določa z-os zemeljskega koordinatnega sistema in je izražena v lokalnem (senzorskem) koordinatnem sistemu kot

$$\boldsymbol{z}_d = \frac{\boldsymbol{a}_m}{\|\boldsymbol{a}_m\|}$$

Smer Zemljine x-osi, izražena v lokalnih koordinatah, je določena s komponento vektorja magnetometra, ki je pravokotna na $\boldsymbol{z}_d$

$$\boldsymbol{x}_d = \frac{\boldsymbol{b}_m \times \boldsymbol{z}_d}{\|\boldsymbol{b}_m \times \boldsymbol{z}_d\|}$$

kjer $\times$ označuje vektorski produkt. Smer proti severu določa y-os Zemlje, izraženo v lokalnem koordinatnem sistemu kot

$$\boldsymbol{y}_d = \boldsymbol{z}_d \times \boldsymbol{x}_d$$

Dobljena rotacijska matrika je

$$\boldsymbol{R}_e^i = [\boldsymbol{x}_d,\ \boldsymbol{y}_d,\ \boldsymbol{z}_d]$$

kjer je rotacijska matrika med globalnim in zemeljskimi koordinatnim sistemom enaka $\boldsymbol{R}_w^i = \boldsymbol{R}_i^{wT} = \left(\boldsymbol{R}_e^i \boldsymbol{R}_e^{wT}\right)^T$. To orientacijo lahko opišemo tudi s kvaternioni s pomočjo enačb (5.9) – (5.10) ali z Eulerjevimi koti s pomočjo relacije (5.4).

Točna ocena orientacije vozila je pomembna tudi pri izvedbi odometrije ali inercialne navigacije za zmanjšanje pogreška orientacije in posledično tudi pogreška ocene pozicije. Zato v absolutnih meritvah smeri gibanja pogosto uporabimo relativne meritve v koraku korekcije ocenjevalnikov (npr. Kalmanovega filtra).

5.3.3 Aktivne značke in globalne meritve pozicije

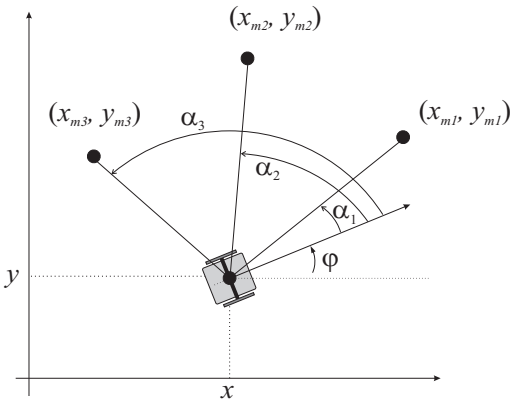
Lokalizacija v okolici je možna tudi z opazovanjem **značk**, ki se nahajajo na znanih pozicijah v okolici. Značke so lahko *naravne*, če so že del okolice (npr. luči na stropu, brezžični oddajniki itd.), ali pa *umetne*, če so nameščene v okolico za namen lokalizacije (npr. radijski oddajniki, ultrazvočni ali infrardeči oddajniki, v zemljo zakopane žice za robotske kosilnice, GPS-sateliti itd.).

Glavna prednost uporabe aktivnih značk je preprosta, robustna in hitra lokalizacija. Vendar so stroški za njihovo namestitev, delovanje in vzdrževanje relativno visoki.

Za oceno pozicije ali lege sistema se običajno uporablja **triangulacija** ali **trilateracija**. Trilateracija s pomočjo izmerjenih razdalj do več oddajnikov (značk) oceni pozicijo sprejemnika, ki je nameščen na vozilo. Zelo znan tovrstni pristop je globalni pozicijski sistem, kjer so aktivne značke sateliti na znanih lokacijah v vesolju. Triangulacija pa uporablja izmerjene kote do treh ali več značk (npr. svetlobni vir) na znanih lokacijah.

Osnovna ideja triangulacije je ponazorjena na sliki 5.13, kjer robot meri relativne kote α_i glede na aktivne značke. Predpostavimo tri značke, kot je prikazano na sliki 5.13. Trenutna lega robota $\boldsymbol{q} = [x,\ y,\ \varphi]^T$ in izmerjeni koti α_j $(j = 1, 2, 3)$ so povezani z naslednjimi relacijami

$$\begin{aligned} \tan(\alpha_1 + \varphi) &= \frac{y_{m1} - y}{x_{m1} - x} \\ \tan(\alpha_2 + \varphi) &= \frac{y_{m2} - y}{x_{m2} - x} \\ \tan(\alpha_3 + \varphi) &= \frac{y_{m3} - y}{x_{m3} - x} \end{aligned} \tag{5.36}$$



Slika 5.13: Lokalizacija robota s pomočjo triangulacije, kjer so izmerjeni relativni koti α_i do značk

Rešitev triangulacije dobimo z rešitvijo enačb (5.36) za $\boldsymbol{q} = [x,\ y,\ \varphi]^T$.

Osnovna ideja trilateracije je prikazana na sliki 5.17, kjer so trenutna pozicija robota, izmerjene razdalje do značk in njihove pozicije povezane z naslednjim sistemom enačb

$$\begin{aligned} d_1^2 &= (x_{m1} - x)^2 + (y_{m1} - y)^2 \\ d_2^2 &= (x_{m2} - x)^2 + (y_{m2} - y)^2 \\ d_3^2 &= (x_{m3} - x)^2 + (y_{m3} - y)^2 \end{aligned} \tag{5.37}$$

V nadaljevanju bomo obravnavali nekaj primerov trilateracije in triangulacije.

Primer 5.9

Robot je opremljen s senzorjem, ki meri smeri do aktivnih značk. V okolici so tri aktivne značke na znanih lokacijah $\boldsymbol{m}_1 = [x_{m1},\ y_{m1}]^T = [0,\ 0]^T$, $\boldsymbol{m}_2 = [x_{m2},\ y_{m2}]^T = [5,\ 3]^T$ in $\boldsymbol{m}_3 = [x_{m3},\ y_{m3}]^T = [1,\ 5]^T$. Pri trenutni legi robota $\boldsymbol{q} = [x,\ y,\ \varphi]^T$ so izmerjene smeri podane kot relativni koti $\alpha_1 = -2{,}7691$, $\alpha_2 = -0{,}3585$ in $\alpha_3 = 1{,}4277$.

Kakšna je trenutna lega robota $\boldsymbol{q}$?

Možnih je več rešitev tega triangulacijskega problema, pri čemer bomo v nadaljevanju predstavili dve možnosti. V prvem delu je uporabljena optimizacija z rojem delcev (PSO), ki smo jo predstavili v poglavju 3.3.9. V drugem delu pa je uporabljen priljubljen geometrijski algoritem, ki temelji na presečišču krožnih lokov.

Rešitev A

Trenutna lega robota $q = [x,\ y,\ \varphi]^T$ in izmerjeni koti α_j $(j = 1, 2, 3)$ so povezani z enačbami (5.36). Naloga algoritma PSO je najti neznanke (x, y in φ), da so relacije (5.36) veljavne. Vsaka pozicija delca predstavlja eno od možnih rešitev ($\boldsymbol{q}_i$) in med optimizacijami se množica delcev posodobi v smislu bolj optimalnih rešitev. Merilo optimalnosti rešitve i-tega delca je podana na naslednji način

$$J_i = f(\boldsymbol{q}_i) = \sum_{j=1}^{3} (\alpha_j - \hat{\alpha}_j)^2$$

kjer je $\hat{\alpha}_j$ simulacija meritve i-tega delca, ki jo dobimo iz (5.36)

$$\hat{\alpha}_j = \arctan \frac{y_{mj} - y}{x_{mj} - x} - \varphi$$

Upoštevajte, da mora funkcija arctan vrniti pravilen kot v območju $(-\pi, \pi]$ (v programskem okolju Matlab se za ta namen lahko uporabi funkcijo `atan2`).

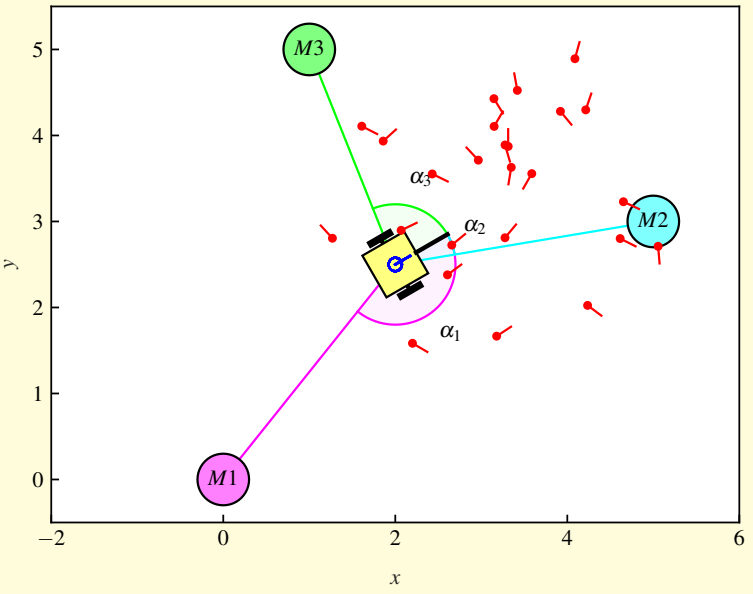
Pravilna rešitev je $\boldsymbol{q} = [2,\ 2{,}5,\ \pi/6]^T$, katere koda je podana v programu 5.6, končna situacija pa je prikazana na sliki 5.14.

Program 5.6

./src/sen/example_triangulation_pso.m

```
m = [0, 0; 5, 3; 1, 5].'; % Značke
r0 = [2; 2.5; pi/6]; % Prava lega robota, ki ni znana.

% Izmerjeni koti
alpha = wrapToPi(atan2(m(2,:)-r0(2), m(1,:)-r0(1))-r0(3));

% Uporaba metode rojenja delcev (PSO)
iterations = 50; % Število iteracij
omega = 0.5; % Faktor vztrajnosti
c1 = 0.5; % Samozavedna konstanta
c2 = 0.5; % Socialna konstanta
N = 25; % Velikost roja delcev

% Začetni položaji delcev
swarm = zeros([3,N,4]);
swarm(1,:,1) = 3 + randn(1,N); % Začetne vrednosti x
swarm(2,:,1) = 3 + randn(1,N); % Začetne vrednosti y
swarm(3,:,1) = 0 + randn(1,N); % Začetne vrednosti fi
swarm(:,:,2) = 0; % Začetne hitrosti delcev
swarm(1,:,4) = 1000; % Najboljša vrednost kriterijske funkcije

for iter = 1:iterations % Iterativno iskanje optimalne rešitve s PSO
    % Vrednotenje parametrov delcev
    for i = 1:N
        % Izračun novega predvidenega kota na podlagi i-tega delca
        pEst = swarm(:,i,1); % Ocenjeni parametri delca (x, y, fi)

        % Primerjava predvidenih kotov z izmerjenimi koti
        alphaEst = wrapToPi(atan2(m(2,:)-pEst(2), m(1,:)-pEst(1))-pEst(3));

        % Izračun kriterijske funkcije
        cost = (alphaEst-alpha)*(alphaEst-alpha).';
```

```
33          if cost<swarm(1,i,4) % Če je novi parameter boljši, posodobi:
34              swarm(:,i,3) = swarm(:,i,1); % vrednosti parametrov (x, y, in fi)
35              swarm(1,i,4) = cost; % in najboljšo vrednost kriterijske funkcije.
36          end
37      end
38      [~, gBest] = min(swarm(1,:,4)); % Parametri globalno najboljšega delca
39
40      % Posodobitev parametrov s hitrostnimi vektorji
41      swarm(:,:,2) = omega*swarm(:,:,2) + ...
42              c1*rand(3,N).*(swarm(:,:,3)-swarm(:,:,1)) + ...
43              c2*rand(3,N).*(repmat(swarm(:,gBest,3), 1, N) - swarm(:,:,1));
44      swarm(:,:,1) = swarm(:,:,1) + swarm(:,:,2);
45  end
46
47  r = swarm(:,gBest,1) % Rešitev, najboljša ocena lege
```

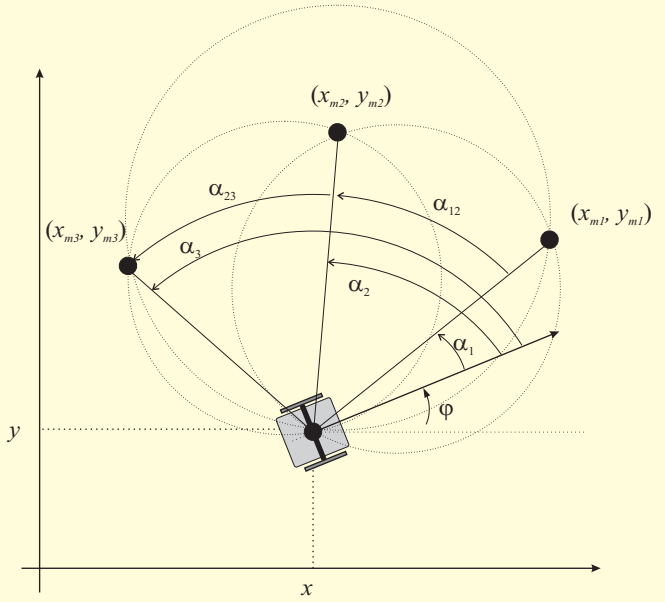


Slika 5.14: Rešitev problema triangulacije iz primera 5.9 z optimizacijo roja delcev (PSO). Začetne lege delcev so označene s *pika-črtica*, končne lege pa s *krog-črta*.

Rešitev B

Obstaja mnogo analitičnih rešitev triangulacije, od katerih se najpogosteje uporablja presek krožnih lokov. Opisali bomo osnovno idejo principa in uporabili končno analitično rešitev za izračun lege robota. Celotno izpeljavo algoritma lahko najdete v [3].

Algoritem temelji na treh krogih, kjer je vsak krog definiran s tremi točkami: par značk $\boldsymbol{m}_i$, $\boldsymbol{m}_j$, $(i, j = 1, 2, 3,\ i \neq j)$ in pozicija robota x, y (glejte sliko 5.15).



Slika 5.15: Lokalizacija robota s triangulacijo na podlagi preseka krožnih lokov

Par značk je povezan s položajem robota z dvema daljicama, med katerima je kot $\alpha_{ij} = \alpha_j - \alpha_i$. Središča in polmeri teh treh krogov so

$$\boldsymbol{c}_{ij} = \begin{bmatrix} x_{cij} \\ y_{cij} \end{bmatrix} = \frac{1}{2}\left(\boldsymbol{m}_i + \boldsymbol{m}_j + \begin{bmatrix} (y_{mi} - y_{mj}\cot\alpha_{ij}) \\ (x_{mj} - x_{mi}\cot\alpha_{ij}) \end{bmatrix}\right)$$
$$r_{ij} = \frac{\|\boldsymbol{m}_i - \boldsymbol{m}_j\|}{2\sin\alpha_{ij}}$$

Ker velja $\alpha_{13} = \alpha_{12} + \alpha_{23}$, sta samo dva od teh kotov neodvisna in njuna pripadajoča kroga (za α_{12} in α_{23}) sta

$$\begin{aligned} (x - x_{12})^2 + (y - y_{12})^2 &= r_{12}^2 \\ (x - x_{23})^2 + (y - y_{23})^2 &= r_{23}^2 \end{aligned} \tag{5.38}$$

Presek obeh krogov (5.38) je rešitev za pozicijo robota. Za več podrobnosti glede pridobitve analitične rešitve, glejte [3]; tukaj je navedena le končna rešitev. Novi začasni koordinatni sistem je definiran tako, da je $\boldsymbol{m}_2$ v njegovem izhodišču in $\boldsymbol{m}_3$ leži na njegovi x-osi, kar omogoča lažjo pridobitev rešitve. Rotacijska matrika od referenčnega do začasnega koordinatnega sistema je

$$\boldsymbol{R} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}$$

kjer je $\beta = \tan\frac{y_{m3}-y_{m2}}{x_{m3}-x_{m2}}$. Značke, izražene v začasnih koordinatah ($\bar{\boldsymbol{m}}_i = [\bar{x}_{mi},\ \bar{y}_{mi}]^T$), dobimo s transformacijo $\bar{\boldsymbol{m}}_i = \boldsymbol{R}^{-1}(\boldsymbol{m}_i - \boldsymbol{m}_2)$. Presečišče krogov v časovnih koordinatah $(\bar{x},\, \bar{y})$ je

$$\begin{bmatrix}\bar{x}\\ \bar{y}\end{bmatrix} = \bar{x}_{m3}\frac{1-\eta\cot\alpha_{12}}{1-\eta^2}\begin{bmatrix}1\\ -\eta\end{bmatrix}$$

kjer je $\eta = \frac{\bar{x}_{m3}-\bar{x}_{m1}-\bar{y}_{m1}\cot\alpha_{12}}{\bar{x}_{m3}\cot\alpha_{23}-\bar{y}_{m1}+\bar{x}_{m1}\cot\alpha_{12}}$. Rešitev v referenčnih koordinatah je

$$\begin{bmatrix}x\\ y\end{bmatrix} = \boldsymbol{m}_2 + \boldsymbol{R}\begin{bmatrix}\bar{x}\\ \bar{y}\end{bmatrix}$$

in orientacija robota

$$\varphi = \arctan\frac{y_{m1}-y}{x_{m1}-x} - \alpha_1$$

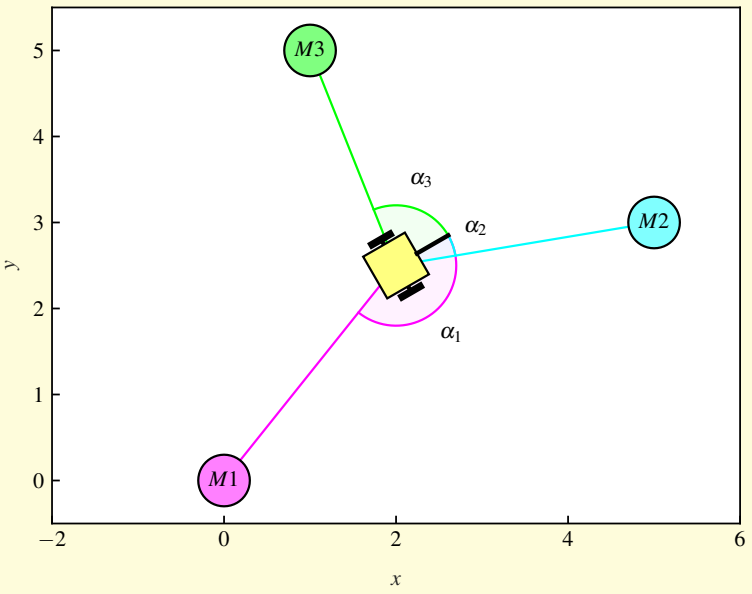
Celoten algoritem za pridobitev rešitve je podan v Matlab kodi v programu 5.7. Rešitev je grafično prikazana na sliki 5.16.

Program 5.7

./src/sen/example_triangulation.m

```
m = [0, 0; 5, 3; 1, 5].'; % Položaji treh značk
r0 = [2; 2.5; pi/6]; % Prava lega robota, ki ni znana.

% Izmerjeni koti
alpha = wrapToPi(atan2(m(2,:)-r0(2), m(1,:)-r0(1))-r0(3));

% Triangulacija: izračun lege na podlagi izmerjenih kotov
f = atan2(m(2,3)-m(2,2), m(1,3)-m(1,2));
S = [cos(f) -sin(f); sin(f) cos(f)]; % Rotacija za koordinatni sistem v m2
m_ = S.'*(m - repmat(m(:,2),1,3)); % Preslikani položaji značk

cta = cot(alpha(2)-alpha(1));
ctb = cot(alpha(3)-alpha(2));
ni = (m_(1,3)-m_(1,1)-m_(2,1)*cta)/(m_(1,3)*ctb-m_(2,1)+m_(1,1)*cta);
p_ = m_(1,3)*(1-ni*ctb)/(1+ni^2)*[1; -ni];

% Rešitev
p = m(:,2) + S*p_ % Položaj
fi = wrapToPi(atan2(m(2,1)-p(2), m(1,1)-r0(1))-alpha(1)) % Orientacija
```

```
p =
    2.0000
    2.5000
fi =
    0.5236
```



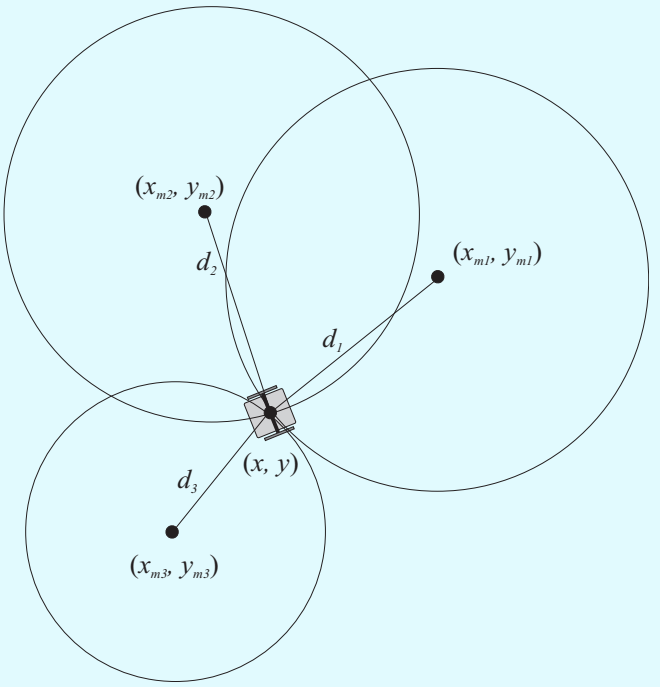
Slika 5.16: Rešitev problema triangulacije iz primera 5.9 z direktnim pristopom

Primer 5.10

Robot je opremljen s senzorjem, ki meri razdaljo do aktivnih značk na podlagi merjenja časa potovanja signala od značke (oddajnik) do robota s sprejemnikom.

Imamo tri aktivne značke na znanih lokacijah $\boldsymbol{m}_1 = [x_{m1},\ y_{m1}]^T = [0,\ 0]^T$, $\boldsymbol{m}_2 = [x_{m2},\ y_{m2}]^T = [5,\ 3]^T$ in $\boldsymbol{m}_3 = [x_{m3},\ y_{m3}]^T = [1,\ 5]^T$.

Pri trenutni poziciji robota $\boldsymbol{r} = [x,\ y]^T$ so izmerjene razdalje $d_1 = 3{,}2016\,\mathrm{m}$, $d_2 = 3{,}0414\,\mathrm{m}$ in $d_3 = 2{,}6926\,\mathrm{m}$, kot je prikazano na sliki 5.17.



Slika 5.17: Lokalizacija robota s trilateracijo, kjer so merjene razdalje do značk d_i

Kakšna je trenutna pozicija robota $\boldsymbol{r}$?

Rešitev

Naloga trilateracijskega algoritma je najti neznano pozicijo x, y, tako da so relacije (5.37) veljavne. To lahko storimo s PSO kot v primeru 5.9, kjer je potrebno izvedbo dopolniti z relacijami (5.37) pri izračunu kriterijske funkcije.

Rešitev (5.37) lahko najdemo tudi analitično. Na voljo imamo več različnih algoritmov, tukaj pa je podana enostavna rešitev. Od prve in druge enačbe v (5.37) odštejemo tretjo ter dobimo

$$d_1^2 - d_3^2 = (x_{m1} - x)^2 - (x_{m3} - x)^2 + (y_{m1} - y)^2 - (y_{m3} - y)^2$$
$$d_2^2 - d_3^2 = (x_{m2} - x)^2 - (x_{m3} - x)^2 + (y_{m2} - y)^2 - (y_{m3} - y)^2$$

Enačbi preuredimo v

$$2(x_{m3} - x_{m1})x + 2(y_{m3} - y_{m1})y = d_1^2 - d_3^2 - x_{m1}^2 + x_{m3}^2 - y_{m1}^2 + y_{m3}^2$$
$$2(x_{m3} - x_{m2})x + 2(y_{m3} - y_{m2})y = d_2^2 - d_3^2 - x_{m2}^2 + x_{m3}^2 - y_{m2}^2 + y_{m3}^2$$

ter tako dobimo linearni enačbi v odvisnosti od x in y, ki ju lahko zapišemo v

obliki $\boldsymbol{A}\boldsymbol{r} = \boldsymbol{b}$, kjer sta

$$\boldsymbol{A} = \begin{bmatrix} 2(x_{m3} - x_{m1}) & 2(y_{m3} - y_{m1}) \\ 2(x_{m3} - x_{m2}) & 2(y_{m3} - y_{m2}) \end{bmatrix}$$

in

$$\boldsymbol{b} = \begin{bmatrix} d_1^2 - d_3^2 - x_{m1}^2 + x_{m3}^2 - y_{m1}^2 + y_{m3}^2 \\ d_2^2 - d_3^2 - x_{m2}^2 + x_{m3}^2 - y_{m2}^2 + y_{m3}^2 \end{bmatrix}$$

od koder izračunamo neznano pozicijo

$$\boldsymbol{r} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

S tem dobimo pravilno rešitev $\boldsymbol{r} = [2,\ 2{,}5]^T$, katere Matlab koda je podana v programu 5.8.

Program 5.8

./src/sen/example_trilateration.m

```
m = [0, 0; 5, 3; 1, 5].'; % Položaji značk
r0 = [2; 2.5; pi/6]; % Prava lega robota, ki ni znana.

% Izmerjene razdalje do značk
d = sqrt((m(1,:)-r0(1)).^2+(m(2,:)-r0(2)).^2);

% Trilateracija: iskanje lege robota glede na izmerjene razdalje
N = size(m,2); A = zeros(N-1,2); b = zeros(N-1,1);
for i = 1:N-1
    A(i,:) = 2*[m(1,N)-m(1,i), m(2,N)-m(2,i)];
    b(i) = d(i)^2-d(N)^2-m(1,i)^2 + m(1,N)^2-m(2,i)^2 + m(2,N)^2;
end

r = A\b % Izračunan položaj

r =
    2.0000
    2.5000
```

Primer 5.11

V primeru 5.10 so bile izmerjene razdalje točne, kar je nerealna predpostavka. Običajno so v meritvah prisotni šumi in druge motnje, zato je potreben predoločen sistem z več kot tremi značkami, da se minimizira pogrešek ocene pozicije. Za ponazoritev uporabimo $n = 4$ značke na lokacijah $\boldsymbol{m}_1 = [x_{m1},\ y_{m1}]^T = [0,\ 0]^T$, $\boldsymbol{m}_2 = [x_{m2},\ y_{m2}]^T = [5,\ 3]^T$, $\boldsymbol{m}_3 = [x_{m3},\ y_{m3}]^T = [1,\ 5]^T$ in $\boldsymbol{m}_4 = [x_{m4},\ y_{m4}]^T = [2,\ 4]^T$. Izmerjene razdalje s šumom so $d_1 = 3{,}2297\,\mathrm{m}$, $d_2 = 3{,}0697\,\mathrm{m}$, $d_3 = 2{,}7060\,\mathrm{m}$ in $d_4 = 1{,}4759\,\mathrm{m}$. Ocenite trenutno pozicijo robota $\boldsymbol{r}$.

Rešitev

Predoločen sistem z n aktivnimi značkami, ki minimizira povprečno kvadratno napako $\|\boldsymbol{A}\boldsymbol{r} - \boldsymbol{b}\|$, dobimo na naslednji način. Matrika $\boldsymbol{A}$ je

$$\boldsymbol{A} = \begin{bmatrix} 2(x_{mn} - x_{m1}) & 2(y_{mn} - y_{m1}) \\ 2(x_{mn} - x_{m2}) & 2(y_{mn} - y_{m2}) \\ \vdots & \vdots \\ 2(x_{mn} - x_{mn-1}) & 2(y_{mn} - y_{mn-1}) \end{bmatrix}$$

in vektor $\boldsymbol{b}$ je

$$\boldsymbol{b} = \begin{bmatrix} d_1^2 - d_n^2 - x_{m1}^2 + x_{mn}^2 - y_{m1}^2 + y_{mn}^2 \\ d_2^2 - d_n^2 - x_{m2}^2 + x_{mn}^2 - y_{m2}^2 + y_{mn}^2 \\ \vdots \\ d_{n-1}^2 - d_n^2 - x_{mn-1}^2 + x_{mn}^2 - y_{mn-1}^2 + y_{mn}^2 \end{bmatrix}$$

Rešitev dobimo z uporabo psevdoinverza

$$\boldsymbol{r} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$$

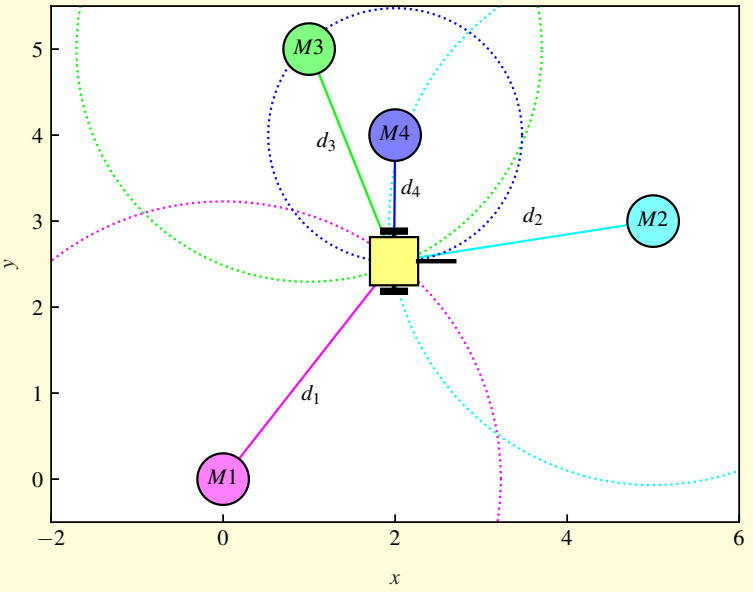
Za dano razdaljo je rešitev $\boldsymbol{r} = [1{,}9873,\ 2{,}5337]^T$. Matlab koda implementacije je podana v programu 5.9, rešitev pa je prikazana na sliki 5.18.

Program 5.9

./src/sen/example_trilateration_noise.m

```
m = [0, 0; 5, 3; 1, 5; 2, 4].'; % Položaji značk

% Izmerjene razdalje do značk
d = [3.2297, 3.0697, 2.7060, 1.4759];

% Trilateracija: iskanje lege robota glede na izmerjene razdalje
N = size(m,2); A = zeros(N-1,2); b = zeros(N-1,1);
for i = 1:N-1
    A(i,:) = 2*[m(1,N)-m(1,i), m(2,N)-m(2,i)];
    b(i) = d(i)^2-d(N)^2-m(1,i)^2 + m(1,N)^2-m(2,i)^2 + m(2,N)^2;
end

r = A\b % Izračunan položaj
```

```
r =
    1.9873
    2.5337
```



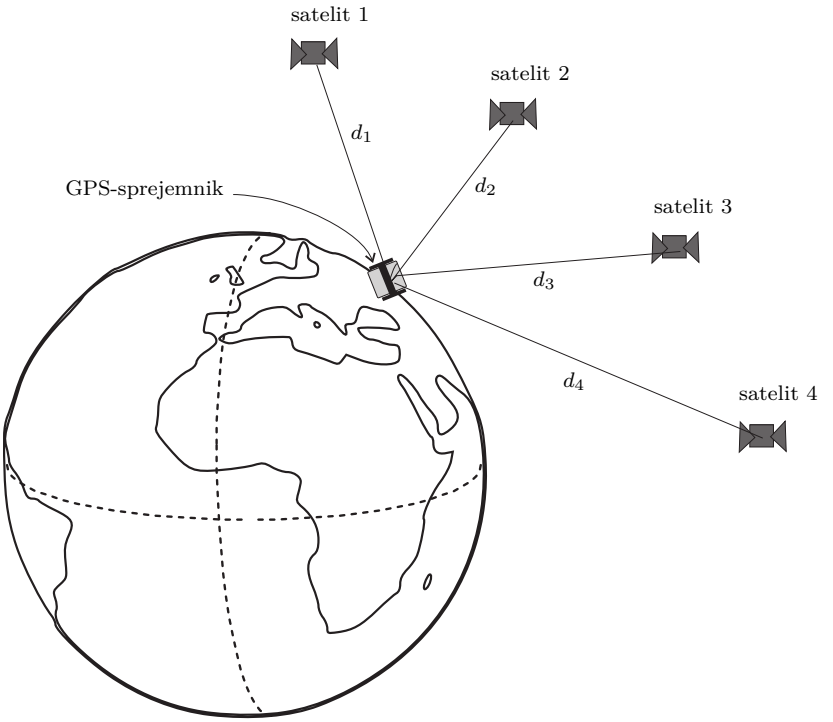
Slika 5.18: Rešitev problema trilateracije iz primera 5.11

Globalni pozicijski sistem

Najpogosteje uporabljen princip trilateracije za lokalizacijo je globalni pozicijski sistem (GPS, angl. *Global positioning system*). Sateliti predstavljajo aktivne značke, ki pošiljajo kodiran signal GPS sprejemni postaji, katere pozicijo je potrebno oceniti s trilateracijo. Sateliti imajo zelo točno atomsko uro ter znane pozicije, določene s Keplerjevimi elementi in drugimi dvovrstičnimi parametri. Obstaja več GPS-sistemov: *Navstar* iz ZDA, *Glonass* iz Rusije in *Galileo* iz Evrope. GPS-sistem Navstar sestoji iz najmanj 24 satelitov, ki dvakrat dnevno obkrožijo Zemljo na višini 20 200 km.

GPS se zdi zelo priročen senzorski sistem za lokalizacijo, vendar ima nekatere omejitve, ki jih je potrebno upoštevati pri uporabi v mobilnih sistemih. Ovire, kot so drevesa, hribi in zgradbe, blokirajo GPS-signal in onemogočijo sprejem. Zaradi večkratnih odbojev pa lahko pride do interference signalov in posledično napačne ocene razdalje. Vseeno gre za zelo zmogljiv sistem, ki dosega točnost okoli 5 m oz. celo 1 cm za diferencialne sisteme z dodatnim sprejemnikom v referenčni postaji.

Lokalizacijo z uporabo GPS lahko razložimo na enostaven način. Sprejemnik meri čas potovanja signala iz določenega satelita. Čas potovanja je razlika med časom sprejema t_r in časom oddaje t_t. Signal potuje s svetlobno hitrostjo c,



Slika 5.19: GPS lokalizacija zahteva sprejem od najmanj štirih satelitov za oceno pozicije GPS-sprejemnika in časovne zakasnitve

zato se lahko izračuna razdalja med sprejemnikom in satelitom. Vendar pa ura sprejemnika ni tako natančna kot atomska ura na satelitih, zato se pojavi neznana časovna pristranskost ali pogrešek razdalje, ki je enak za vse razdalje do satelitov. Torej mora GPS sprejemnik oceniti 4 parametre: svojo tridimenzionalno pozicijo (x, y, z) in časovno pristranskost t_b.

Okoli vsakega satelita narišemo sfero (tj. površina krogle), katere polmer določa izmerjena razdalja. Presek dveh sfer je krožnica, presek treh sfer pa sta dve točki, v katerih se lahko nahaja sprejemnik. Zato potrebujemo vsaj še eno sfero, da zanesljivo ocenimo pozicijo sprejemnika. Če se sprejemnik nahaja na površju Zemlje, jo lahko obravnavamo kot četrto sfero, s katero izločimo pravilno točko, pridobljeno iz preseka treh satelitskih sfer. V idealnem primeru bi bili trije satelitski sprejemniki dovolj. Ampak kot smo že omenili, je ura sprejemnika netočna, kar povzroča neznano časovno pristranskost t_b. Posledično presek štirih sfer (tri od satelitov in ena od Zemlje) ni točka ampak območje. Za oceno časa t_b in manjši pogrešek lokalizacije je potreben sprejem četrtega satelita, kar pomeni, da so za GPS-lokalizacijo potrebni vsaj štirje sateliti, kot je prikazano na sliki 5.19. GPS lokalizacija mora rešiti naslednji sistem enačb

$$d_1 = c(t_{r1} - t_{t1} - t_d) = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2}$$
$$d_2 = c(t_{r2} - t_{t2} - t_d) = \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2}$$
$$d_3 = c(t_{r3} - t_{t3} - t_d) = \sqrt{(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2}$$
$$d_4 = c(t_{r4} - t_{t4} - t_d) = \sqrt{(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2}$$

kjer so neznanke pozicija sprejemnika x, y, z in časovna zakasnitev sprejemnika t_d. Za i-ti satelit so znane vrednosti pozicija $(x_i,\ y_i,\ z_i)$, čas sprejema t_{ri}, čas prenosa t_{ti} in hitrost svetlobe c.

5.3.4 Navigacija z uporabo značilk okolja

Značilke so podmnožica vzorcev, ki jih je mogoče robustno razbrati iz neobdelanih meritev senzorja ali drugih podatkov. Značilke so lahko premice, daljice, krogi, pike, robovi, vogali in drugi vzorci. Zaznavanje značilk v okolju se lahko uporablja za namene lokalizacije (ocene lege) mobilnega robota in gradnjo zemljevida. V primeru dvodimenzionalnega laserskega pregledovalnika razdalj lahko iz dobljenih podatkov (razdalja in kot) izločimo značilke v obliki daljic. Linijske značilke v lokalnem koordinatnem sistemu mobilnega robota lahko nato primerjamo z globalnim zemljevidom okolice, ki je prav tako predstavljen z nizom linij, da bi določili lego mobilnega robota na zemljevidu. Dandanes se v mobilni robotiki uporabljajo različni senzorji, med katerimi je najpopularnejša kamera. V zadnjih letih so bili razviti številni algoritmi strojnega vida za zaznavanje slikovnih značilk, ki se lahko uporabijo tudi za merjenje lege robota v okolici. Pristopi, ki temeljijo na značilkah, običajno vsebujejo naslednje korake: *detekcija značilk*, *opis značilk* in *ujemanje značilk*. V fazi detekcije značilk se obdelujejo neobdelani podatki za določitev lokacij značilk. Za opis zaznane značilke se običajno uporablja območje okoli njene lokacije, nato se lahko uporabijo deskriptorji (opisi značilk) za iskanje podobnih značilk (faza ujemanja značilk).

Značilke se nahajajo na znanih lokacijah, zato lahko njihovo opazovanje izboljša znanje o lokaciji mobilnega robota (manjša negotovost lokacije). Seznam značilk z njihovimi lokacijami se imenuje **zemljevid**, ki je lahko predhodno shranjen v pomnilniku ali pa se gradi sproti med lokalizacijo — pristop, ki to omogoča, se imenuje SLAM (angl. *simultaneous localization and mapping*). Prvi pristop je metodološko enostavnejši, a hkrati nepraktičen, še posebej za večja okolja, saj zahteva uporabo nekega referenčnega sistema lokalizacije ali pa je potrebno ročno zapisati zaznane značilke. Glavna ideja drugega pristopa pa je možnost lokalizacije iz opazovanih značilk, ki so že na zemljevidu, in shranjevanje novo opaženih značilk na podlagi zaznane lokacije. Za zanesljivo zaznavanje značilk in kar se da točno lokalizacijo robota je priporočljiva metoda relativnega določanja položaja – odometrija. V primeru težko razpoznavnih značilk (npr. debla dreves v sadovnjaku ali zaznavanje daljic v stavbah) so za identifikacijo opazovanih značilk potrebne približne informacije o lokaciji robota. Približna lokacija robota v trenutnem času je pridobljena iz lokacije v prejšnjem času in napovedi odometrije za relativno gibanje od prejšnje do trenutne lokacije.

Značilke so lahko *naravne*, če so že del okolja, ali *umetne*, če so izdelane posebej za namen lokalizacije. Naravne značilke v strukturiranih okoljih (običajno v zaprtih prostorih) so stene, talne plošče, luči, vogali ipd., v nestrukturiranih okoljih (običajno na prostem) pa so to drevesna debla, prometni znaki itd. Umetne značilke so narejene izključno za namen preproste in robustne lokalizacije (barvne oznake, črtne kode, talne linije itd.).

Običajno pridobivanje značilk zahteva nekaj obdelave podatkov senzorjev, da bi dobili bolj kompaktno, informativno in abstraktno predstavitev trenutnega pogleda senzorja (linijska predstavitev proti množici točk). V nekaterih primerih lahko uporabimo tudi neobdelane meritve senzorja (npr. slika kamere) za proces lokalizacije s korelacijo pogleda senzorja in shranjenega zemljevida.

Pogosto se uporabljajo vizualne značilke, ki jih je mogoče zaznati z nekaterimi slikovnimi senzorji. Ena najpreprostejših in najbolj uporabljenih značilk je premica, ki jo lahko v okolju zazna kamera ali laserski merilnik razdalj.

Premica kot značilka

V lokalizaciji je premica pogosta izbira za značilko, saj gre za preprosto geometrijsko obliko. Uporabimo jo lahko za opisovanje notranjega ali zunanjega okolja (stene, ploski predmeti, cestne proge itd.). S primerjavo trenutno opazovanih parametrov značilk in parametrov predhodno znanega zemljevida okolja lahko ocenimo lego robota.

V ta namen se pogosto uporablja laserski merilnik razdalj, ki meri oblak točk odboja v okolju. Iz tega izmerjenega oblaka točk se s pomočjo različnih namenskih algoritmov ocenijo (običajno dvodimenzionalni) parametri premic. Postopek prilagajanja premici navadno zahteva dva koraka: prvi je *identifikacija rojev*, ki jih je mogoče predstaviti s premico, drugi pa je *ocena parametrov prilagajanja premici* za vsak roj, recimo z metodo najmanjših kvadratov. Običajno se ta dva koraka izvajata iterativno.

Algoritem razcepi-in-združi Zelo priljubljen algoritem za obdelavo podatkov laserskega pregledovalnika razdalj je *razcepi-in-združi* (angl. *split-and-merge*) [4, 5], ki je preprost za izvedbo, ima nizko računsko zahtevnost in dobro zmogljivost. Algoritem zahteva paketne podatke, ki so iterativno razdeljeni na roje, kjer je vsak roj opisan z linearnim prototipom (premica za dvodimenzionalne podatke). Algoritem se lahko uporabi le za urejene podatke, kjer zaporedni vzorci podatkov pripadajo isti premici (podatki iz laserskega pregledovalnika razdalj so običajno urejeni).

Sprva vsi vzorci podatkov pripadajo enemu roju, katerega parametri linearnega prototipa (premica) so prepoznani. Roj se nato razdeli pri vzorcu, ki ima največjo razdaljo od prototipa in je ta razdalja večja od praga d_{split}. Izbira vrednosti d_{split} je odvisna od šuma podatkov in mora biti večja od pričakovanega merilnega

pogreška zaradi šuma (npr. tri standardne deviacije). Ko se rojenje zaključi, se združijo kolinearni roji. Ta korak je neobvezen in običajno ni potreben pri urejenih podatkovnih vrstah.

Za vsak roj j ($j = 1, \ldots, m$) zapišemo linearni prototip v normalni obliki

$$[\boldsymbol{z}^T(k),\ 1]\boldsymbol{\theta}_j = 0 \tag{5.39}$$

kjer je $\boldsymbol{z}(k) = [x(k),\ y(k)]^T$ k-ti vzorec ($k = 1, \ldots, n$; n je število vseh vzorcev), ki leži na premici, določeni s parametri $\boldsymbol{\theta}$. Za roj j, ki vsebuje vzorce $k_j = 1, \ldots, n_j$, lahko s pomočjo singularnega razcepa ocenimo vektor parametrov $\boldsymbol{\theta}_j$. Regresijska matrika

$$\boldsymbol{\psi} = \begin{bmatrix} \boldsymbol{z}^T(1) & 1 \\ \vdots & \vdots \\ \boldsymbol{z}^T(n_j) & 1 \end{bmatrix}$$

določa množico homogenih enačb v matrični obliki $\boldsymbol{\psi}\boldsymbol{\theta}_j = \mathbf{0}$, kjer je potrebno oceniti parametre prototipa $\boldsymbol{\theta}_j$ v smislu minimizacije najmanjših kvadratov. Rešitev predstavlja lastni vektor ($\boldsymbol{p}_r = [p_x,\ p_y,\ p_p]^T$) regresijske matrike $\boldsymbol{\psi}^T\boldsymbol{\psi}$, ki pripada najmanjši lastni vrednosti (izračunana z uporabo singularnega razcepa). Parametre prototipa v normalni obliki dobimo z normalizacijo $\boldsymbol{p}_r$

$$\boldsymbol{\theta}_j = \pm \frac{\boldsymbol{p}_r}{\sqrt{p_x^2 + p_y^2}}$$

kjer je izbrani predznak nasproten od tretjega parametra $\boldsymbol{p}_r$ (p_p). Za podatkovni vzorec $\boldsymbol{z}(k)$, ki ni v prototipu j, izračunamo ortogonalno razdaljo

$$d_j(k) = \left|\ [\boldsymbol{z}^T(k),\ 1]\boldsymbol{\theta}_j\ \right|$$

V primeru dvodimenzionalnih podatkov se lahko linearni prototip alternativno oceni s povezovanjem prvega in zadnjega podatkovnega vzorca v roju. To ni optimalno v smislu najmanjših kvadratov, vendar zmanjšuje računsko kompleksnost in zagotavlja, da se vzorec, ki definira razcep, ne pojavi v prvem ali zadnjem podatkovnem vzorcu.

Prikaz prilagoditve podatkov laserskega pregledovalnika razdalj na premice je podan v primeru 5.12.

Primer 5.12

Za podatke laserskega pregledovalnika razdalj, ki so sestavljeni iz 180 točk odboja (glejte sliko 5.20 in program 5.10), ocenite roje premic, ki najbolje opisujejo meritve. Rojenje poteka z uporabo praga razdalje $d_{split} = 0{,}06\,\mathrm{m}$.

Program 5.10: Podatki laserskega pregledovalnika razdalj
./src/sen/script_laserscandata.m

```
data = [-0 -2149  38 -2158  76 -2166  110 -2092  137 -1962  162 -1851 ...
    185 -1761  222 -1809  255 -1817  289 -1822  324 -1835  358 -1840 ...
    393 -1849  428 -1853  464 -1862  502 -1873  539 -1879  577 -1887 ...
    617 -1898  656 -1905  697 -1915  738 -1921  780 -1929  824 -1942 ...
    860 -1931  873 -1872  885 -1814  898 -1763  911 -1714  957 -1726 ...
    1001 -1734  1035 -1722  2407 -3852  2431 -3743  2452 -3635  2476 ...
    -3536  2497 -3437  2519 -3342  2539 -3250  2558 -3158  2576 -3070 ...
    2596 -2986  2614 -2903  2630 -2821  2649 -2744  2664 -2664  2683 ...
    -2591  2698 -2516  2715 -2445  2730 -2373  2742 -2301  2759 -2234 ...
    2774 -2167  2790 -2102  2802 -2036  2817 -1973  2831 -1910  2845 ...
    -1847  2857 -1785  2869 -1724  2885 -1666  2898 -1606  2908 -1546 ...
    2924 -1490  2936 -1432  1146 -535  1149 -512  1155 -490  1160 -469 ...
    1164 -447  1169 -425  1174 -404  1178 -383  1182 -361  1188 -341 ...
    1190 -319  1195 -298  1199 -277  1205 -256  1210 -235  1214 -214 ...
    1220 -193  1222 -172  1227 -151  1231 -129  1238 -108  1241 -87 ...
    1248 -65  1253 -44  1254 -22  1259 0  1265 22  1268 44  1277 67 ...
    1279 89  1287 113  1291 136  1295 159  1300 183  1306 207  1312 ...
    231  1315 256  1320 280  1328 307  1178 294  1133 304  1092 313 ...
    1050 321  1014 329  977 336  944 344  915 351  885 357  856 363 ...
    829 369  805 375  778 380  756 385  735 391  713 395  694 401  674 ...
    405  652 408  637 413  617 416  600 420  582 423  566 427  552 432 ...
    537 435  520 436  507 441  493 444  478 446  463 447  452 452  439 ...
    455  426 456  414 459  401 461  391 466  380 469  1920 2457  1944 ...
    2580  1976 2720  2011 2872  2044 3030  2081 3204  2115 3385  2042 ...
    3399  1971 3415  1903 3433  1832 3445  1764 3462  1694 3474  1629 ...
    3493  1561 3506  1495 3522  1428 3533  1362 3549  1297 3564  1231 ...
    3575  1167 3593  1102 3603  187 654  175 655  164 656  153 662 ...
    141 662  130 667  118 668  106 671  94 670  83 677  71 678  60 681 ...
    48 683  36 686  24 691  12 690].';
x = data(2:2:end)/1000;
y = data(1:2:end-1)/1000;
```

Rešitev

Predstavljena je enostavna izvedba algoritma, ki izračuna parametre premic vsakega roja v smislu najmanjših kvadratov. Če je potrebno roj razdeliti in se pojavi delitveni vzorec (ki definira razcep) kot prva ali zadnja točka v roju, potem roja ni mogoče razcepiti. V tem primeru ponovno izračunamo parametre premic, da se prilegajo le prvemu in zadnjemu vzorcu, ter izvedemo razcep.

Izvedba ocene premic je podana v programu 5.11, podatki laserskega pregledovalnika razdalj pa so v programu 5.10. Ocenjene daljice so prikazane na sliki 5.20.

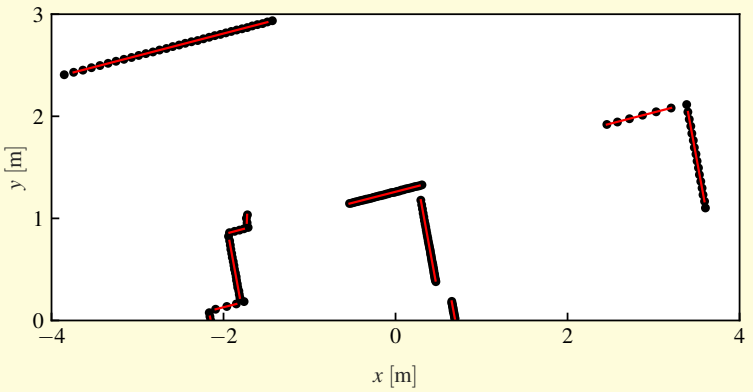
Program 5.11

./src/sen/example_lines_sandm.m

```
X = [x, y]; % Meritve laserskega merilnika razdalj
[N, M] = size(X);

% Init
C = 50; % Maksimalno število rojev
clusters = 1; % Zadnji aktivni roj
dMin = 0.06; % Prag razdalje za deljenje roja

```

```
sizeOfCluster = zeros(C,1); % Število točk v rojih
clusterBounds = zeros(C,2); % Indeksi mejnih točk v rojih
clusterParams = zeros(C,M+1); % Parametri rojev
splitCluster = zeros(C,1); % Zastavica za deljenje

% Na začetku so vse točke v enem roju
sizeOfCluster(clusters,1) = N;
clusterBounds(clusters,:)= [1, N]; % Točke so urejene
splitCluster(clusters,1) = 1; % Začetni roj lahko delimo

exit = false;
while ~exit
    exit = true;
    tmpLastCluster = clusters;
    for i = 1:tmpLastCluster
        if splitCluster(i)
            p0 = clusterBounds(i,1); % Začetna točka v roju
            p1 = clusterBounds(i,2); % Končna točka v roju

            % Ocena parametrov roja v smislu najmanjših kvadratov
            Psi = [X(p0:p1,:), ones(p1-p0+1,1)];
            [~, ~, V] = svd(Psi);
            thetaEst = V(:,3);
            % Preslikava premice ax+by+c=0 v normalno obliko
            s = -sign(thetaEst(3)); if s==0, s = 1; end
            mi = 1/sqrt(thetaEst(1)^2+thetaEst(2)^2)*s;
            Theta = thetaEst*mi;

            % Ocenitev enostavnih parametrov premice (an podlagi začetne in
            % končne točke). Ti parametri se uporabijo, ko je točka deljenja
            % na meji roja
            if abs(X(p1,1)-X(p0,1))<100*eps % Vertikalna premica
                a = 1; b = 0; c = -X(1,1);
            else
                a = (X(p1,2)-X(p0,2))/(X(p1,1)-X(p0,1));
                b = -1;
                c = -a*X(p0,1) + X(p0,2);
            end
            % Preslikava premice ax+by+c=0 v normalno obliko
            thetaEst = [a; b; c];
            s = -sign(thetaEst(3)); if s==0, s = 1; end
            mi = 1/sqrt(thetaEst(1)^2+thetaEst(2)^2)*s;
            Theta0 = thetaEst*mi;

            % Shranjevanje optimalnih parametrov
            clusterParams(i,:) = Theta.';
            ind = p0:p1;
            XX = X(ind,:);

            % Izračun razdalje na podlagi prve in zadnje
            % točke v roju (enostavna premica)
            dik = [XX, ones(size(XX,1),1)]*Theta0;
            [dd0, iii] = max(abs(dik)); ii0 = ind(iii);

            % Izračun razdalje od premice v smislu najmanjših kvadratov
            dik = [XX, ones(size(XX,1),1)]*Theta;
            [dd, iii] = max(abs(dik)); ii = ind(iii);

            % Deljenje roja
            doSplit = 0;
            if dd>dMin && (ii-p0)>=2 && (p1-ii)>=1 % Optimalne premice
                if clusters<C
                    iiFin = ii; % Lokacija deljenja
```

```
72                      doSplit = 1;
73                      clusterParams(i,:) = Theta.';
74                  end
75              elseif dd0>dMin && (ii0-p0)>=2 && (p1-ii0)>=1 % Enostavne premice
76                  if clusters<C
77                      iiFin = ii0; % Lokacija deljenja
78                      doSplit = 1;
79                      clusterParams(i,:) = Theta0.';
80                  end
81              else
82                  splitCluster(i) = 0;
83              end
84
85
86              if doSplit==1 && clusters<C
87                  % Deljenje roja v roja A in B
88                  clusters = clusters + 1; % Nov roj
89                  % Prva in zadnja točka v roju A
90                  clusterBounds(i,1);
91                  clusterBounds(i,2) = iiFin-1;
92                  splitCluster(i) = 1;
93                  % Prva in zadnja točka v roju B
94                  clusterBounds(clusters,1) = iiFin+1;
95                  clusterBounds(clusters,2) = p1;
96                  splitCluster(clusters) = 1;
97
98                  exit = false;
99              end
100         end
101     end
102 end
```



Slika 5.20: Podatki LRF in identični roji premic z uporabo algoritma *razcepi-in-združi*

Samorazvijajoče se rojenje premic Podobno kot pri algoritmu *razcepi-in-združi* lahko ocenimo premice tudi v primeru podatkovnih vrst. Rojenje se izvaja sproti in se iterativno posodobi, ko prispejo novi podatki. Primer preprostega in

računalniško učinkovitega algoritma je *samorazvijajoče se rojenje premic* [6]. V nadaljevanju bomo na kratko opisali njegove glavne korake.

Zapišemo j-ti prototip, ki modelira podatke $\boldsymbol{z}(k_j)$ $(k_j = 1, \ldots, n_j)$ v j-tem roju

$$(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j)^T \cdot \boldsymbol{p}_j = 0$$

kjer je $\boldsymbol{\mu}_j(k_j)$ srednja vrednost podatkov v j-tem roju, ki se posodobi v vsaki iteraciji (ko je na voljo nov vzorec) kot

$$\boldsymbol{\mu}_j(k_j) = \frac{k_j - 1}{k_j}\boldsymbol{\mu}_j(k_j - 1) + \frac{1}{k_j}\boldsymbol{z}(k_j)$$

in $\boldsymbol{p}_j$ je normalni vektor j-tega prototipa, ki ga izračunamo iz kovariančne matrike j-tega roja (za dvodimenzionalne podatke)

$$\boldsymbol{\Sigma}_j(k_j) = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 \end{bmatrix}$$

kot lastni vektor pri pripadajoči najmanjši lastni vrednosti $\boldsymbol{\Sigma}_j(k_j)$

$$\boldsymbol{p}_j = \begin{cases} \begin{bmatrix} \frac{\theta}{\sqrt{1+\theta^2}} & \frac{-1}{\sqrt{1+\theta^2}} \end{bmatrix}^T & ; \quad |\lambda_1| \leq |\lambda_2| \\ \begin{bmatrix} \frac{1}{\sqrt{1+\theta^2}} & \frac{\theta}{\sqrt{1+\theta^2}} \end{bmatrix}^T & ; \quad |\lambda_1| > |\lambda_2| \end{cases}$$

kjer so θ in lastne vrednosti λ_1 in λ_2 določene z

$$\begin{aligned} \theta &= \frac{-\sigma_{11}^2 + \sigma_{22}^2 + \sqrt{\sigma_{11}^4 + \sigma_{22}^4 - 2\sigma_{11}^2\sigma_{22}^2 + 4\sigma_{12}^4}}{2\sigma_{12}^2} \\ \lambda_1 &= \sigma_{22}^2 - \theta\sigma_{12}^2 \\ \lambda_2 &= \sigma_{22}^2 - \theta\sigma_{12}^2 + \frac{1+\theta^2}{\theta}\sigma_{12}^2 \end{aligned}$$

Kovariančna matrika se posodablja iterativno

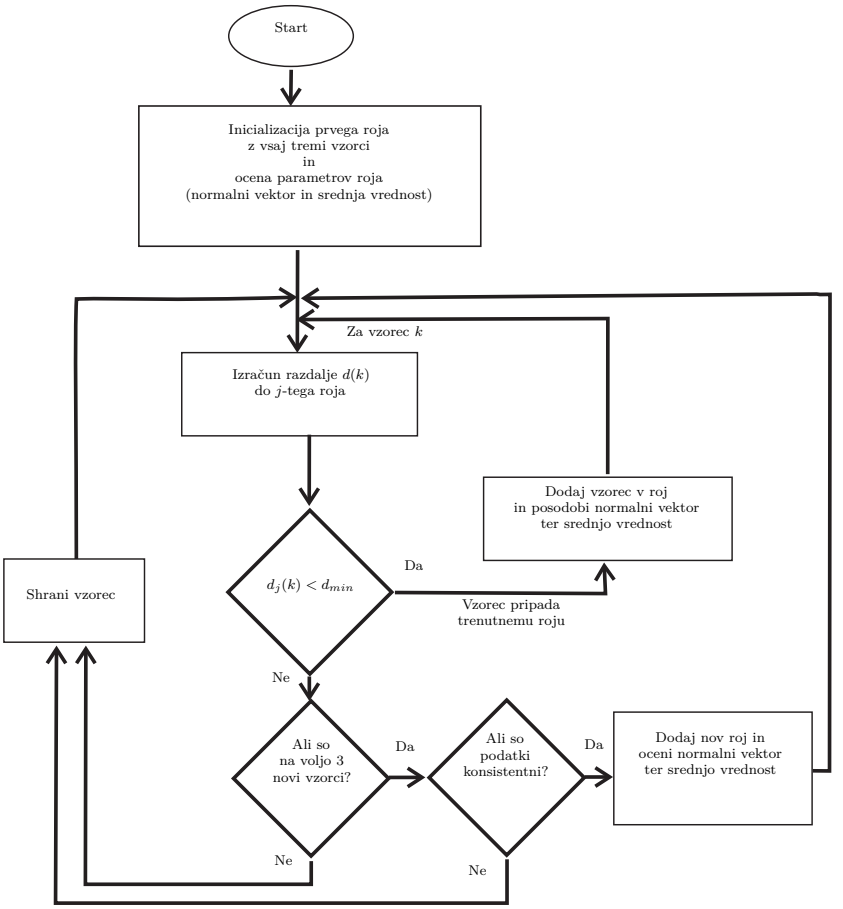
$$\boldsymbol{\Sigma}_j(k_j) = \frac{k_j - 2}{k_j - 1}\boldsymbol{\Sigma}_j(k_j - 1) + \frac{1}{k_j}\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j(k_j - 1)\right)\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j(k_j - 1)\right)^T$$

Trenutni vzorec $\boldsymbol{z}(k)$ je potrebno razvrstiti v enega od obstoječih prototipov j $(j \in \{1, \ldots, m\})$. To se izvede z izračunom ortogonalne razdalje $d_j(k)$ od vsakega j-tega prototipa

$$d_j(k) = |(\boldsymbol{z}(k) - \boldsymbol{\mu}_j)^T \boldsymbol{p}_j|$$

Če je $d_j(k) = 0$, podatkovni vzorec leži na j-tem linearnem prototipu. Vzorec spada v j-ti roj, če je razdalja $d_j(k)$ za j-ti roj najmanjša ter hkrati manjša od vnaprej določenega praga d_{min} $(d_j(k) < d_{min})$. V [6] je predlagano robustno rojenje, kjer je d_{min} sproti ocenjen iz podatkov j-tega roja.

Osnovna ideja algoritma rojenja je prikazana na sliki 5.21. Lahko ga uporabimo sproti za urejene podatkovne vrste ali pa paketne podatke vzorcev (kot *razcepi-in-združi*). Rezultati razvrstitve primera 5.12 so računsko manj zahtevni za podobno kakovost rojenja kot pri algoritmu *razcepi-in-združi*.



Slika 5.21: Princip samorazvijajočega se rojenja za pretočne podatke, kjer roje določajo prototipi premic

Primer 5.13

Za podatke laserskega pregledovalnika razdalj, ki so sestavljeni iz 180 točk odboja (glejte sliko 5.20 in začetek rešitve za koordinate točk), ocenite roje premic, ki najbolje opišejo meritve. Rojenje poteka s pomočjo algoritma razvijajočega se rojenja premic.

Rešitev

Trenutni vzorec pripada j-temu roju, če je njegova razdalja $d_j(k)$ do premice roja manjša od praga d_{min} ($d_j(k) < d_{min}$). Prag d_{min} je lahko konstanta ali pa ga, kot v tem primeru, ocenjujemo sproti. Prag razdalje d_{min} se izračuna iz ocenjene variance razdalje roja $\sigma_j(k_j)$ (varianca razdalje vzorcev od premice). Rekurzivna ocena variance je $\sigma_j(k_j) = \sigma_j(k_j - 1)\frac{k_j-2}{k_j-1} + \frac{d_j^2(k)}{k_j}$ in prag je $d_{min} = \kappa_{max}\sqrt{\sigma_j}$, kjer je $\kappa_{max} = 7$ nastavitveni parameter.

Izvedba ocene premic je podana v programu 5.12 (podatki laserskega pregledovalnika razdalj so opredeljeni v programu 5.10). Ocenjene daljice so prikazane na sliki 5.22.

Program 5.12

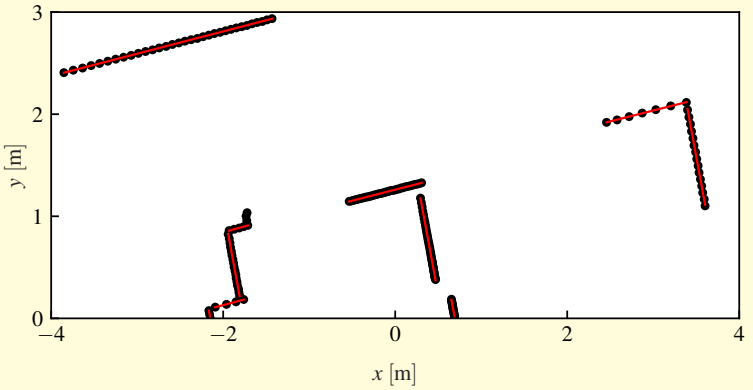
./src/sen/example_straight_lines.m

```
X = [x, y]; % Meritve laserskega merilnika razdalj
kappaMax = 7; % Prag za faktor ortogonalne razdalje
cosPhiTh = cos(10/180*pi); % Prag za začetno kolinearnost roja

[n,m] = size(X); dimension = m;
% Parametri
Nr_cloud_max = 20; Current_clust = 1;

Nr_points_in_cloud = zeros(Nr_cloud_max,1); % Vektor, ki določa število
% točk v oblaku, kjer vsaka vrstica pripada drugemu oblaku
M_of_clouds = zeros(Nr_cloud_max, dimension); % Matrika - vrstice pripadajo
% različnim oblakom in stolpci vsebujejo elemente vhodnega vektorja
V_of_clouds = zeros(Nr_cloud_max, dimension^2);
VarD_of_cloud = zeros(Nr_cloud_max,1); % Varianca razdalje točke
% od roja
M_dist = zeros(Nr_cloud_max,1); % Razdalje med točkami v roju
Eig_of_clouds = zeros(Nr_cloud_max, dimension);
EigLat_of_cloud = zeros(Nr_cloud_max, dimension);
Points_in_buffer = zeros(6,dimension); % Shranimo do 6 vzorcev
StartEndX_points_in_cloud = zeros(Nr_cloud_max,2);

% Inicializacija
% Prvi roj ima tri točke (če so le-te kolinearne)
Nr_points_in_cloud(Current_clust,1) = 3;
Nr_points_in_buffer = 0;

N = 3; % Začetno število točk
XX = X(1:N,:);
M = sum(XX)/N;

dXX = XX - repmat(M,N,1);
```

```
Vmat = dXX'*dXX/(N-1); % Kovariančna matrika vzorcev

theta = ((Vmat(1,1)^2 - 2*Vmat(1,1)*Vmat(2,2) + 4*Vmat(1,2)^2 ...
      + Vmat(2,2)^2)^(1/2) - Vmat(1,1) + Vmat(2,2))/(2*Vmat(1,2));
lam1 = Vmat(2,2) - theta*Vmat(1,2);
lam2 = Vmat(2,2) - theta*Vmat(1,2) + (1 + theta^2)/theta*Vmat(1,2);
if norm(lam2) > norm(lam1) % Izberi glavni lastni vektor
    p = [1/sqrt((1+theta^2)), theta/sqrt((1+theta^2))];
else
    p = [theta/sqrt((1+theta^2)), -1/sqrt((1+theta^2))];
end
p = p / sqrt(p*p'); % Prvi lastni vektor (normaliziran)
pL = [p(1,2), -p(1,1)]; % Drugi lastni vektor (normaliziran)

% Razdalja od premice
sumD = 0;
for i = 1:N
    sumD = sumD + (((X(i,:) - M)*pL')/(pL*pL'))^2;
end
varD = sumD/(N-1);

% Povprečna razdalja med točkami
d1 = sqrt( (X(1,:)-X(2,:))*(X(1,:)-X(2,:))' );
d2 = sqrt( (X(3,:)-X(2,:))*(X(3,:)-X(2,:))' );
d_med_toc = (d1+d2)/2;

% Shranjevanje
M_of_clouds(Current_clust,:) = M;
V_of_clouds(Current_clust,:) = [Vmat(1,:), Vmat(2,:)];
Eig_of_clouds(Current_clust,:) = p;
EigLat_of_cloud(Current_clust,:) = pL;
VarD_of_cloud(Current_clust,1) = varD;
M_dist(Current_clust,1) = d_med_toc;
StartEndX_points_in_cloud(Current_clust,1) = 1;
StartEndX_points_in_cloud(Current_clust,2) = 3;

for k = 4:n % Sprehod čez vse vzorce
    % Izračun razdalje do trenutnega vzorca
    pL = EigLat_of_cloud(Current_clust,:);
    M = M_of_clouds(Current_clust,:);
    Vmat = [V_of_clouds(Current_clust,1:2); V_of_clouds(Current_clust,3:4)];
    varD = VarD_of_cloud(Current_clust,1);

    d = abs((X(k,:) - M)*pL');

    if Nr_points_in_cloud(Current_clust) > 1
        d_med_toc = sqrt((X(k,:) - X(k-1,:))*(X(k,:) - X(k-1,:))');
    end

    % Nova točka pripada roju, če je njena razdalja do roja dovolj
    % majhna in so vzorci dovolj skupaj
    if d < kappaMax*sqrt(varD) && d_med_toc < 2*M_dist(Current_clust,1)
        StartEndX_points_in_cloud(Current_clust,2) = k;
        Nr_points_in_buffer = 0;
        j = Nr_points_in_cloud(Current_clust)+1; % Povečanje vzorcev v roju
        Mold = M_of_clouds(Current_clust,:);
        dXM = X(k,:) - Mold;
        M = Mold + dXM/j; % Rekurzivni izračun srednje vrednosti

        % Kovariančna matrika podatkov
        Vmat = Vmat*(j-2)/(j-1) + 1/j*dXM'*dXM;

        theta = ((Vmat(1,1)^2 - 2*Vmat(1,1)*Vmat(2,2) + 4*Vmat(1,2)^2 ...
```

```
95              + Vmat(2,2)^2)^(1/2) - Vmat(1,1) + Vmat(2,2))/(2*Vmat(1,2));
96          lam1 = Vmat(2,2) - theta*Vmat(1,2);
97          lam2 = Vmat(2,2) - theta*Vmat(1,2) + (1 + theta^2)/theta*Vmat(1,2);
98          if norm(lam2) > norm(lam1) % Izberi glavni lastni vektor
99              p = [1/sqrt((1+theta^2)), theta/sqrt((1+theta^2))];
100         else
101             p = [theta/sqrt((1+theta^2)), -1/sqrt((1+theta^2))];
102         end
103         p = p/sqrt(p*p'); % Prvi lastni vektor (normaliziran)
104         pL = [p(1,2), -p(1,1)]; % Drugi lastni vektor (normaliziran)
105
106         % Rekurzivni izračun variance razdalje
107         d = abs( (X(k,:) - M)*pL');
108         varDold = VarD_of_cloud(Current_clust,1);
109         varD = varDold*(j-2)/(j-1)+d^2/j;
110
111         % Shranjevanje
112         M_of_clouds( Current_clust,:) = M;
113         V_of_clouds( Current_clust,:) = [Vmat(1,:), Vmat(2,:)];
114         Eig_of_clouds (Current_clust,:) = p;
115         EigLat_of_cloud( Current_clust,:) = pL;
116         Nr_points_in_cloud(Current_clust) = j;
117         VarD_of_cloud(Current_clust,1) = varD;
118     else   % Nova točka ne pripada roju - ustvarimo nov roj
119         Nr_points_in_buffer = Nr_points_in_buffer + 1;
120         Points_in_buffer(Nr_points_in_buffer, :) = X(k,:);
121
122         % Novi roj mora imeti 3 konsistentne vzorce
123         if Nr_points_in_buffer >= 3
124             XX = Points_in_buffer(1:Nr_points_in_buffer,:);
125             M = sum(XX)/Nr_points_in_buffer;
126             dXX = XX - repmat(M,Nr_points_in_buffer,1);
127             Vmat = dXX'*dXX/(Nr_points_in_buffer-1);
128
129             theta = ((Vmat(1,1)^2 - 2*Vmat(1,1)*Vmat(2,2) + 4*Vmat(1,2)^2 ...
130                   + Vmat(2,2)^2)^(1/2) - Vmat(1,1) + Vmat(2,2))/(2*Vmat(1,2));
131             lam1 = Vmat(2,2) - theta*Vmat(1,2);
132             lam2 = Vmat(2,2) - theta*Vmat(1,2) + (1 + theta^2)/theta*Vmat(1,2);
133             if norm(lam2) > norm(lam1) % Izberi glavni lastni vektor
134                 p = [1/sqrt((1+theta^2)), theta/sqrt((1+theta^2))];
135             else
136                 p = [theta/sqrt((1+theta^2)), -1/sqrt((1+theta^2))];
137             end
138             p = p / sqrt(p*p'); % Prvi lastni vektor (normaliziran)
139             pL = [p(1,2) , -p(1,1)]; % Drugi lastni vektor (normaliziran)
140
141             % Testiranje konsistentnosti vzorcev
142             sumD = 0;
143             for i = 1:Nr_points_in_buffer
144                d = abs((XX(i,:) - M)*pL');
145                sumD = sumD + d^2;
146             end
147             varD = sumD/(Nr_points_in_buffer-1);
148             % Popravek, če je začetna varianca premajhna
149             if varD < mean(VarD_of_cloud(:,1))
150                 varD = mean(VarD_of_cloud(:,1));
151             end
152
153             d1 = sqrt((XX(1,:)-XX(2,:))*(XX(1,:)-XX(2,:))');
154             d2 = sqrt((XX(3,:)-XX(2,:))*(XX(3,:)-XX(2,:))');
155
156             vv1 = XX(1,:) - XX(2,:);
157             vv2 = XX(2,:) - XX(3,:);
```

```
158             cosPhi = vv1*vv2'/(norm(vv1)*norm(vv2));
159
160             if d<kappaMax*sqrt(varD) && abs(d2-d1)<min(d1,d2) && cosPhi>cosPhiTh
161                 % Vzorci so konsistentni in primerno razmaknjeni
162                 Current_clust = Current_clust + 1;
163                 M_of_clouds(Current_clust,:) = M;
164                 V_of_clouds(Current_clust,:) = [Vmat(1,:), Vmat(2,:)];
165                 Eig_of_clouds(Current_clust,:) = p;
166                 EigLat_of_cloud(Current_clust,:) = pL;
167                 Nr_points_in_cloud(Current_clust) = 3;
168                 VarD_of_cloud(Current_clust,1) = varD;
169                 StartEndX_points_in_cloud(Current_clust,1) = k - 2;
170                 StartEndX_points_in_cloud(Current_clust,2) = k;
171                 d_med_toc = (d1 + d2)/2;
172                 M_dist(Current_clust,1) = d_med_toc;
173             end
174             Nr_points_in_buffer = 0; % Brisanje medpomnilnika
175         else % Čakanje na vsaj tri vzorce v medpomnilniku
176         end
177     end
178 end
```



Slika 5.22: Podatki laserskega pregledovalnika razdalj in prepoznani roji premic z uporabo algoritma samorazvijajočega se rojenja premic

Houghova transformacija Houghova transformacija [7] je zelo uporaben pristop za oceno geometrijskih "primitivov", ki se večinoma uporabljajo pri obdelavi slik. Vhodni podatki se zapišejo v parametrični prostor (npr. parametri premice), kjer maksimumi podajo število in parametre premic.

Algoritem zahteva kvantizacijo parametričnega prostora. Fina kvantizacija poveča točnost, vendar je računsko in spominsko zahtevna. Da bi se izognili kvantizaciji in povečali točnost Houghove transformacije, je bilo izvedenih več raziskav, npr. naključnostna Houghova transformacija ali adaptivne izvedbe, obravnavane v [8, 9].

Houghova transformacija lahko zanesljivo oceni roje ob prisotnosti osamelcev. V osnovni različici sta parametra premice α in d definirana z linearnim prototipom (5.39), kjer je $\boldsymbol{\theta}_j = [\cos\alpha,\ \sin\alpha,\ -d]$. Normalna parametra premice, ki se običajno nahajata v območju $-\pi < \alpha \leq \pi$ in $d_{min} < d \leq d_{max}$, sta predstavljena v akumulatorju s kvantizacijo do N diskretnih vrednosti za α in M diskretnih vrednosti za d.

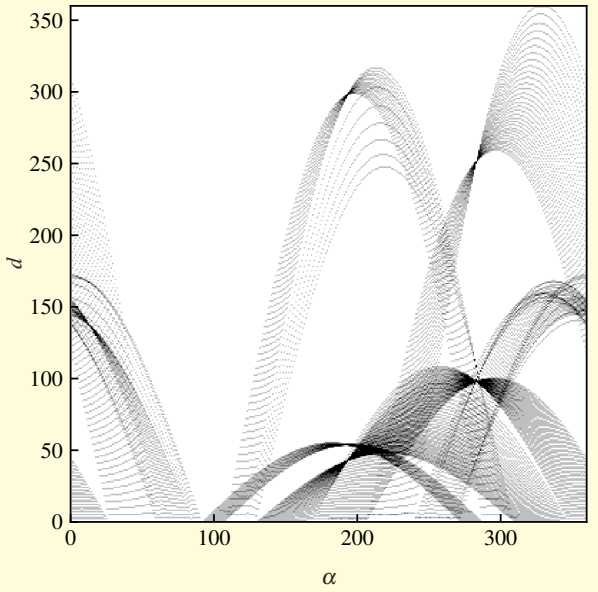
Za vsak vzorec podatkov $x(k)$, $y(k)$ in vse možne vrednosti $\alpha(n) = -\pi + \frac{\pi n}{N}$, $n \in \{1, \ldots, N\}$, se izračunajo rešitve parametra $d(n)$. Vsak par $\alpha(n)$, $d(n)$ predstavlja možno premico, ki vsebuje vzorec $x(k)$, $y(k)$. Za vsak izračunan parameter se ustrezna lokacija v akumulatorju poveča za 1. Ko so vsi vzorci podatkov obdelani, so celice akumulatorja z najvišjimi vrednostmi iskani roji premic. Za pravilno izbiro kvantizacije parametričnega prostora in vrednosti praga je potrebno nekaj predhodnega znanja, da dobimo ustrezne maksimume v akumulatorju.

Primer 5.14

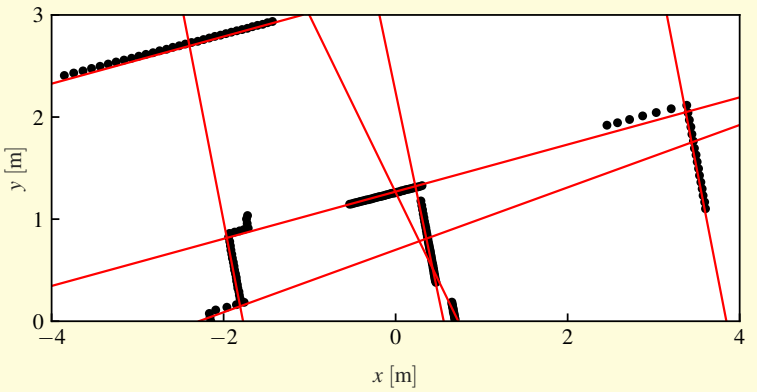
Za podatke laserskega pregledovalnika razdalj, sestavljenih iz 180 točk odboja (glejte sliko 5.24 in začetek rešitve za koordinate točk), ocenite roje premic, ki najbolje opišejo meritve. Rojenje se izvede s pomočjo Houghove transformacije, kjer sta normalna parametra premice α in d kvantizirana na 720 diskretnih vrednosti.

Rešitev

Možna rešitev v programskem okolju Matlab je predstavljena v programu 5.13 (podatki laserskega pregledovalnika razdalj so podani v programu 5.10), kjer je za iskanje maksimumov v akumulatorju uporabljena funkcija `houghpeaks`. Pridobljeni akumulator je prikazan na sliki 5.23, razpoznani roji pa na sliki 5.24.



Slika 5.23: Akumulator Houghove transformacije, kjer sta $\alpha \in (-\pi, \pi]$ in $d \in [0,\ 4{,}5]$ kvantizirana na 720 diskretnih vrednosti



Slika 5.24: Podatki laserskega pregledovalnika razdalj in razpoznani roji premic z uporabo Houghove transformacije

Program 5.13

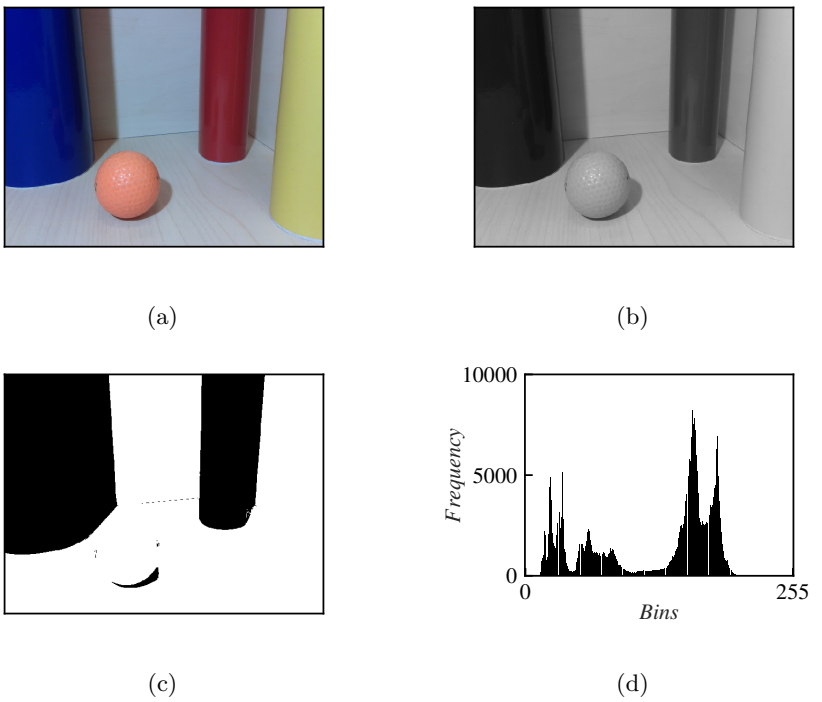
./src/sen/example_lines_hough.m

```
1 [x, y]; % Meritve laserskega merilnika razdalj
2 N = length(x);
3
```

```
dAlpha = pi/180; % Kvantizacijski kot
nAlpha = round(2*pi/dAlpha);
nDist = nAlpha; % Kvantizacijska razdalja
lutDist = zeros(N,nAlpha); % Preslikovalna tabela za razdaljo
for i = 1:N % Za vsako točko izračunamo premice v območju kota alpha
    for j = 1:nAlpha
        alpha = (j-1)*dAlpha-pi;
        % Razdalja od koordinatnega izhodišča
        d = x(i)*cos(alpha)+y(i)*sin(alpha);
        if d<0
            if alpha>pi, alpha = alpha - pi;
            else         alpha = alpha + pi; end
            jj = round((alpha+pi)/dAlpha);
            lutDist(i,jj) = -d;
        else
            lutDist(i,j) = d;
        end
    end
end

% Določitev območja za parameter razdalje
minLutDist = min(lutDist(:));
maxLutDist = max(lutDist(:));
dDist = (maxLutDist-minLutDist)/nDist; % Kvantizacijska razdalja

% Akumulator parametričnega prostora
A = zeros(nDist,nAlpha);
for i = 1:N % Sprehod čez vse točke
    for j = 1:nAlpha
        k = round((lutDist(i,j)-minLutDist)/dDist)+1;
        if k>nDist, k = nDist; end
        A(k,j) = A(k,j)+1;
    end
end
H = A(2:nAlpha,:); % Akumulator

% Določitev maksimumov v skululatorju (najbolj verjetne premice)
nLines = 7; % Število najverjetnejših premic
peaks  = houghpeaks(H, nLines, 'threshold', 3, 'NHoodSize', [31, 31]);
% Parametri premice: razdalja od izhodišča in naklon premice
distAlpha = [peaks(:,1).'*dDist+minLutDist; peaks(:,2).'*dAlpha-pi];
```

Slikovne značilke

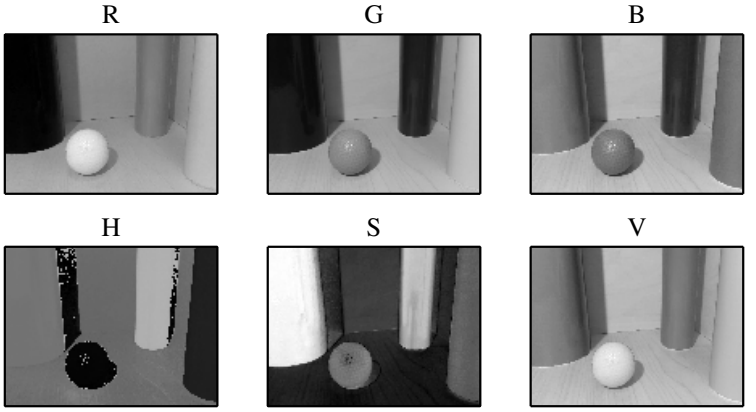
Številne dobre lastnosti kamere, računske zmogljivosti sodobnih računalnikov in napredek pri razvoju algoritmov omogočajo uporabo kamere za reševanje problemov v robotiki. Kamera se lahko uporablja za zaznavanje, prepoznavanje in sledenje opazovanih objektov v vidnem polju kamere, saj so slike projekcije tridimenzionalnih objektov v okolju (glejte poglavje 5.2.4). Digitalna slika je dvodimenzionalen diskreten signal, predstavljen z matriko kvantiziranih števil, ki predstavljajo prisotnost ali odsotnost svetlobe, jakost svetlobe (osvetljenost), barvo ali kakšno drugo veličino. Glavne vrste slik so: barvna (slika 5.25a), sivinska (slika 5.25b) in binarna (slika 5.25c). Pri strojnem vidu sivinske slike običajno zadostujejo za iskanje vzorcev, ki niso odvisni od barve. Binarne slike



Slika 5.25: Primer (a) barvne, (b) sivinske in (c) binarne slike ter (d) histogram sivinske slike

so rezultat segmentacije slike in se uporabljajo za maskiranje vsebine. Eden od najpreprostejših načinov segmentacije slike je *upragovljanje*, pri čemer so slikovni elementi s sivinsko vrednostjo nad pragom označeni z logično 1, vsi ostali pa so nastavljeni na logično 0. Upragovljanje sivinske slike 5.25b s pragom 70 privede do binarne slike 5.25c. Za določitev najustreznejše vrednosti praga lahko pogostost sivinskih vrednosti na sliki predstavimo v histogramu. Slika 5.25d prikazuje histogram z 256 intervali, ki ustrezajo 256 nivojem sivinske slike 5.25b.

V zadnjih letih so bili razviti številni algoritmi strojnega vida, ki omogočajo sledenje objektom na podlagi slike. V ta namen je vsebina slike običajno predstavljena z značilkami slike. Značilke so lahko območja slike s podobnimi lastnostmi (npr. podobna barva), oznake z določenim vzorcem ali nekatere druge značilnosti slike (npr. robovi, vogali, črte). V določenih situacijah lahko v okolje vstavimo *umetne značke*, ki omogočajo hitro in zanesljivo sledenje značilkam (npr. barvne značke ali matrične črtne kode za sledenje mobilnim robotom pri nogometu). Kadar to ni mogoče, je potrebno značilke izločiti iz slike (neprilagojene) scene. V nekaterih aplikacijah je mogoče uporabiti preprosto *barvno segmentacijo* (npr. odkrivanje rdečih jabolk v sadovnjaku). V zadnjih letih je bilo razvitih več pristopov za izločanje *naravnih lokalnih značilk slike*, ki so invariantne za nekatere transformacije in popačenja slike.



Slika 5.26: RGB in HSV komponente barvne slike 5.25a

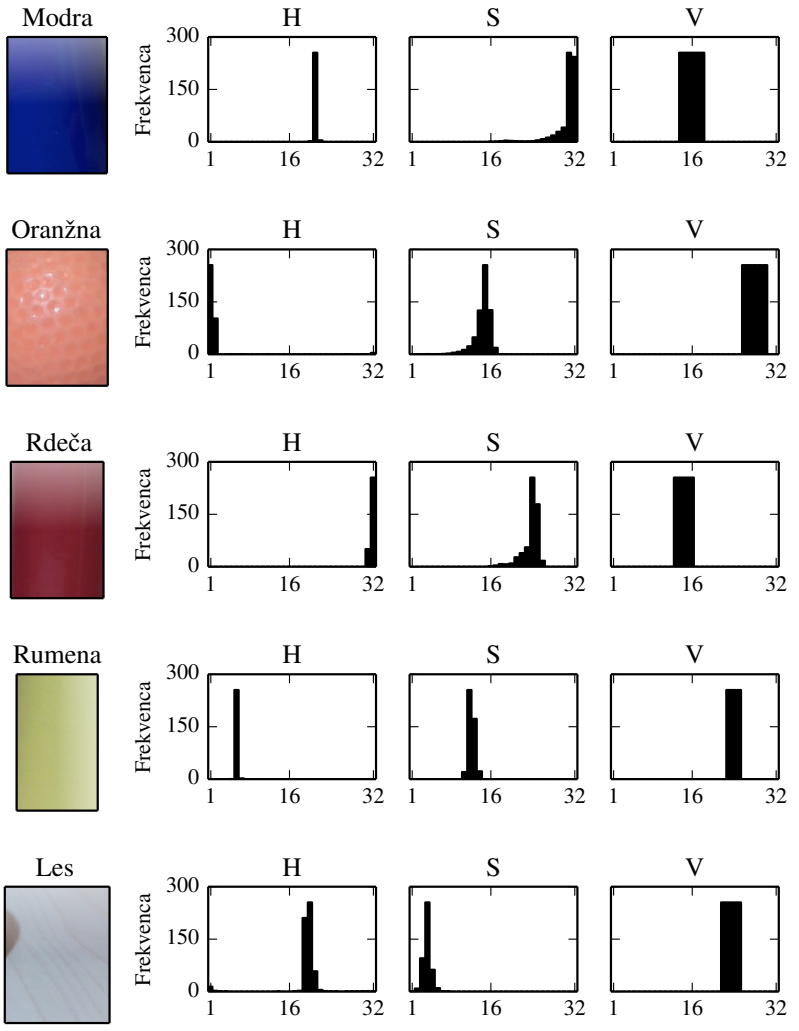
Barvne značilke Zaznavanje območij slike s podobno barvo ni trivialna naloga zaradi nehomogene osvetlitve, senc in odsevov. Sledenje z uporabo barvnih značilk je običajno uporabljeno samo v okoljih, kjer je mogoče vzpostaviti nadzorovano osvetlitev ali pa je možno barvo predmeta, ki mu sledimo, dovolj dobro razločiti od ostalih predmetov v okolju.

Barva v digitalni sliki je ponavadi predstavljena s tremi barvnimi komponentami, tj. rdečo, zeleno in modro, kar je znano kot *barvni model RGB*. Določeno barvo dobimo s kombinacijo teh treh barvnih komponent, kjer posamezno komponento filtriramo skozi rdeč, zelen ali moder barvni filter. Druga dva barvna prostora, ki se pogosto uporabljata pri strojnem vidu, sta *HSL* (angl. *hue-saturation-lightness*) in *HSV* (angl. *hue-saturation-value*). Barvni prostor HSV omogoča bolj naraven opis in boljšo segmentacijo barv kot pa prostor RGB. Vrednosti v območju $[0,\ 255]$ iz barvnega prostora RGB lahko pretvorimo v barvni prostor HSV z

$$
\begin{aligned}
H &= \begin{cases} 0 & ; \quad M - m = 0 \\ 60\frac{G-B}{M-m} \mod 360 & ; \quad M = R \\ 60\frac{B-R}{M-m} & ; \quad M = G \\ 60\frac{R-G}{M-m} & ; \quad M = B \end{cases} \\
S &= \begin{cases} 0 & ; \quad M = 0 \\ \frac{M-m}{M} & ; \quad \text{sicer} \end{cases} \\
V &= \frac{M}{255}
\end{aligned}
\tag{5.40}
$$

kjer je $M = \max\{R, G, B\}$ in $m = \min\{R, G, B\}$. V (5.40) se nasičenost S in vrednost V gibljeta v območju $[0,\ 1]$, barvni odtenek H pa v območju $[0,\ 360)$. Matematična operacija $x \mod y$ predstavlja ostanek pri deljenju števila x s številom y. Na sliki 5.26 so prikazane RGB in HSV komponente barvne slike 5.25a.

Barvni histogrami se lahko uporabijo za segmentacijo območij določene barve.



Slika 5.27: Histogrami različnih barvnih predlog v barvnem prostoru HSV

HSV histogrami (z 32 intervali) različnih predlog s podobno barvo so prikazani na sliki 5.27 (barvna področja so vzeta s slike 5.25a). Histograme vsake predloge je mogoče projicirati nazaj na izvirno sliko, saj je za vsak slikovni element na sliki nastavljena frekvenca intervala, ki pripada vrednosti slikovnega elementa v ustreznem barvnem kanalu na sliki. Nastale sivinske slike lahko združimo kot linearno kombinacijo barvnih kanalov. Rezultati povratne projekcije HSV-histograma modre, oranžne, rdeče, rumene barve ter lesa so prikazani v prvem stolpcu na sliki 5.28. V dobljeni sliki sivinski nivoji predstavljajo merilo podobnosti slikovnih elementov barvni predlogi. Preden je slika upragovljena (glejte tretji stolpec na 5.28), je mogoče dodatno filtrirati sliko. Sliko lahko zgladimo z dvodimenzionalnim Gaussovim filtrom (glejte drugi stolpec na 5.28), da odstranimo nekaj šumnih vrhov. Upragovljeno binarno sliko lahko filtriramo z npr. morfološkim filtrom, da odstranimo ali zapolnimo nekatera povezana območja. Nastalo masko slike (glejte četrti stolpec na sliki 5.28) lahko uporabimo za filtriranje zaznanih območij na prvotni sliki (glejte zadnji stolpec na sliki 5.28). Več o algoritmih za obdelavo slik je med drugim na voljo v [10]. V praktičnih izvedbah segmentacije slike na podlagi barv se nekateri procesi filtriranja zavoljo hitrejšega delovanja preskočijo, kar pa zmanjša točnost. V primeru, da so barvne vrednosti vseh slikovnih komponent znotraj določene spodnje in zgornje meje, se vsak slikovni element na vhodni sliki šteje za del predmeta. Na sliki 5.27 lahko opazimo, da vrednosti barvnega odtenka in nasičenosti omogočata preprosto barvno segmentacijo. Vendar je potrebno poudariti, da je barvna segmentacija občutljiva na spreminjajoče se svetlobne pogoje v prostoru. Čeprav nekateri pristopi lahko kompenzirajo nehomogene pogoje osvetlitve [11], se barvna segmentacija slike najpogosteje uporablja le v okoliščinah, kjer je mogoče doseči ustrezne pogoje osvetlitve.

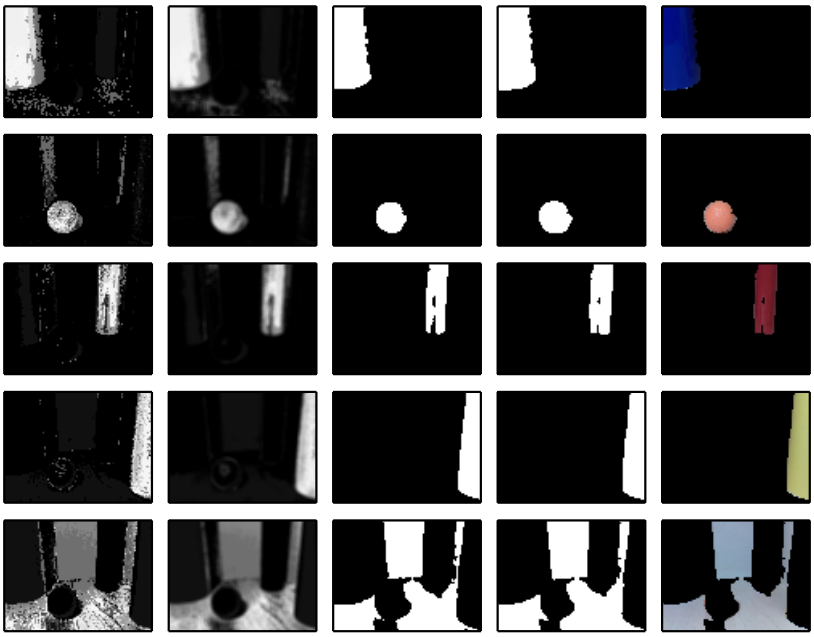
Rezultat segmentacije slike je binarna slika, v kateri so slikovni elementi, ki ne pripadajo objektu (ozadje), postavljeni na 0. Obstaja lahko več ali samo eno povezano območje. Če obstaja več povezanih območij, je potrebno uporabiti ustrezen algoritem, da najdemo pozicije in oblike teh objektov [11]. Za bolj robustno iskanje povezanih območij lahko uvedemo nekatere omejitve, ki zavrnejo območja glede na določen pogoj (npr. velikost območja). Če se opazovan objekt bistveno razlikuje od okolice in ga je mogoče zanesljivo zaznati kot eno območje na segmentirani sliki, lahko pozicijo in obliko tega območja opišemo s *slikovnimi momenti*.

Definicija *osnovnega slikovnega momenta* binarne digitalne slike $I(x, y) \in \{0, 1\}$ je

$$m_{p,q} = \sum_x \sum_y x^p y^q I(x, y)$$

kjer sta p in q pozitivni celi števili ($p + q$ je red momenta). Moment $m_{0,0}$ predstavlja maso objekta, ki je v primeru binarne slike enaka njegovi površini. Da v binarni sliki najdemo središče objekta (x_0, y_0), lahko uporabimo momente ničtega in prvega reda

$$x_0 = \frac{m_{10}}{m_{00}} \qquad y_0 = \frac{m_{01}}{m_{00}}$$



Slika 5.28: Detekcija barvnih značilk. Vsaka vrstica predstavlja drugačno barvo. Koraki od levega do desnega stolpca: povratna projekcija histograma barvne predloge, glajenje, upragovljanje, dodatno filtriranje in maskiranje slike

Za opis orientacije in oblike objekta lahko uporabimo *središčne momente*

$$\mu_{p,q} = \sum_x \sum_y (x - x_0)^p (y - y_0)^q I(x, y)$$

ki so invariantni za transformacije v sliki. Središčne momente lahko uporabimo za prilagoditev elipsoida na zaznani objekt na sliki. Veliko polos elipse a in malo polos b lahko dobimo iz lastnih vrednosti λ_a in λ_b ($\lambda_a \geq \lambda_b$) matrike $\boldsymbol{Q}$, ki je sestavljena iz središčnih momentov drugega reda

$$\boldsymbol{Q} = \begin{bmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{bmatrix}$$

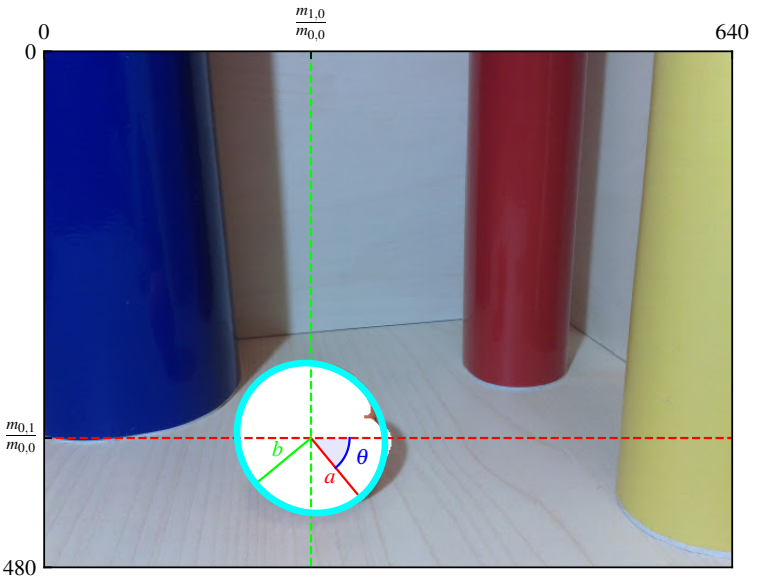
Osi elipse sta

$$a = \frac{2\sqrt{\lambda_a}}{m_{00}} \qquad b = \frac{2\sqrt{\lambda_b}}{m_{00}}$$

Orientacijo velike polosi elipse podaja kot

$$\theta = \frac{1}{2} \arctan \frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}$$

Na sliki 5.29 so uporabljeni osnovni in središčni momenti, da se elipsoid prilagodi na binarno oznako (rezultat segmentacije oranžne barve na sliki 5.28). Način za pridobitev parametrov elipsoida je prikazan v programu 5.14, kjer so središčni momenti izpeljani iz osnovnih. Vidimo, da lahko momente slike uporabimo za opis preprostih značilk in določitev njihovih lokacij na sliki. V [12] je opredeljena množica momentov, ki so invariantni za skaliranje, translacijo in rotacijo.

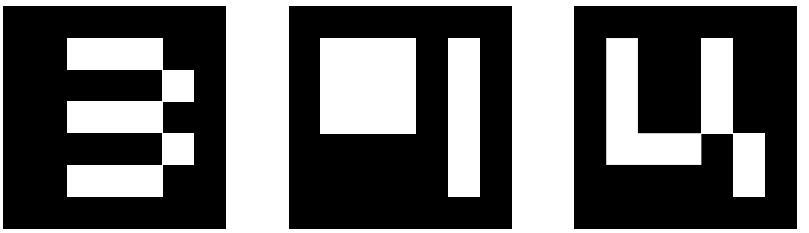


Slika 5.29: Prilagoditev elipsoida na masko, ki je pridobljena kot rezultat segmentacije oranžne barve

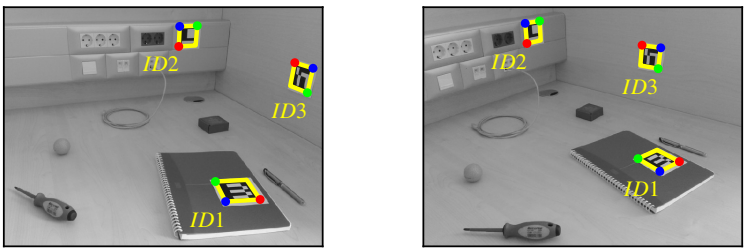
Program 5.14

./src/sen/example_moments.m

```
im = imread('colour_orange_mask.bmp')>128; % Binarna slika

[x, y] = meshgrid(1:size(im,2), 1:size(im,1));

% Osnovni momenti
m00 = sum(sum( (x.^0).*(y.^0).*double(im) ));
m10 = sum(sum( (x.^1).*(y.^0).*double(im) ));
m01 = sum(sum( (x.^0).*(y.^1).*double(im) ));
m11 = sum(sum( (x.^1).*(y.^1).*double(im) ));
m20 = sum(sum( (x.^2).*(y.^0).*double(im) ));
m02 = sum(sum( (x.^0).*(y.^2).*double(im) ));

% Površina, x in y
area = m00
x0 = m10/m00
y0 = m01/m00

% Centralni momenti
u00 = m00;
u11 = m11-x0*m01; % u11 = m11-y*m10;
u20 = m20-x0*m10;
u02 = m02-y0*m01;

% Elipsa
v = eig([u20, u11; u11, u02]); % Lastne vrednosti
a = 2*sqrt(v(2)/u00) % Velika polos
b = 2*sqrt(v(1)/u00) % Mala polos
theta = atan2(2*u11, u20-u02)/2 % Usmerjenost

area =
        14958
x0 =
```



Slika 5.30: Umetne značke



Slika 5.31: Zaznane umetne značke v dveh pogledih kamere

```
  248.1951
y0 =
  359.7101
a =
   71.7684
b =
   66.6585
theta =
    0.8795
```

Značke z umetnimi vzorci Uvedba *umetnih značk* predstavlja minimalno prilagoditev okolja, ki lahko znatno poenostavi nekatere naloge strojnega vida, npr. sledenje objektom, ocena lege kamere itd. Trije primeri umetnih značk so prikazani na sliki 5.30. Vzorci značk so običajno zasnovani tako, da jih je mogoče zanesljivo in natančno zaznati. Poleg tega je v vzorcu lahko zapisana oznaka (ID) značke, zato je mogoče sledenje in prepoznava več značk v zaporedju slik. Eden od priljubljenih algoritmov za zaznavanje umetnih značk je *ArUco* [13, 14]. Slika 5.31 prikazuje zaznane umetne značke iz slike 5.30 v dveh pogledih kamere.

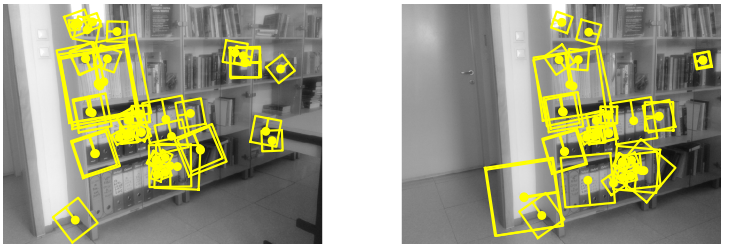
Algoritmi za zaznavanje umetnih značk običajno obsegajo korak lokalizacije značke, v katerem se določi pozicija, orientacija in velikost značke. Ta postopek mora robustno razločiti samo prave značke od preostalih objektov v okolju. Ko so lokacije značk na sliki znane, je mogoče določiti projekcijo med zaznano značko in njenim lokalnim koordinatnim sistemom. Na podlagi vzorca značke v transformiranem lokalnem koordinatnem sistemu se določi njena oznaka (ID).

Nekateri vzorci umetnih značk so zasnovani tako, da omogočajo popravljanje vzorcev, zaradi česar je njihovo prepoznavanje bolj zanesljivo in hkrati tudi robustno na šum in okluzije.

Naravne lokalne slikovne značilke Slike vsebujejo značilke, ki so same po sebi prisotne v okolju. Skozi leta so bili razviti številni algoritmi strojnega vida, ki omogočajo samodejno zaznavanje lokalnih značilk na slikah. Nekateri pomembni algoritmi za detekcijo lokalnih značilk so SIFT (angl. *scale invariant feature transform*) [15], SURF (angl. *speeded-up robust features*) [16], MSER (angl. *maximally stable extremal regions*) [17], FAST (angl. *features from accelerated segment test*) [18], AGAST (angl. *adaptive and generic accelerated segment test*) [19] idr. Večina teh algoritmov je vključena v odprtokodno knjižnico za računalniški vid *OpenCV* [20, 21]. V robotskih aplikacijah je pomembna lastnost opisovanja slik z značilkami ter učinkovitost algoritma, da omogoča delovanje v realnem času. Možni so različni pristopi za detekcijo lokalnih značilk (lokalizacija značilk), opis lokalnih značilk (predstavitev lastnosti značilk) in ujemanje značilk.

Cilj **detekcije značilk** je odkrivanje in lokalizacija točk zanimanja (lokalna območja), običajno za zmanjšanje dimenzije slike. Običajno značilke detektiramo na sivinskih slikah. Obstaja več vrst slikovnih vzorcev: robovi (npr. Cannyev filter, Sobelov operator in Robertsov operator), vogali (npr. Harrisov detektor oglišč, Hessianova matrika, FAST), mehurčki (npr. Laplaceov operator z Gaussovim filtrom, razlika Gaussovih filtrov, MSER). Za zaznavanje značilk različnih velikosti, je slika običajno predstavljena v ločljivostnem prostoru [15]. Detekcija značilk mora biti invariantna na nekatere vrste transformacij in popačenj slike, da je možno ponovljivo odkrivanje značilk v več pogledih istega prizora. Običajno je zaželeno, da so značilke invariantne za translacijo, rotacijo, skaliranje, glajenje, osvetlitev in šum. Značilke bi morale biti robustne tudi za nekatere delne okluzije. Zaželena lastnost značilk je torej lokalnost. V mobilni robotiki je običajno zahtevana točna zaznava zadosti velikega števila značilk, saj je to predpogoj za točno in robustno izvajanje algoritmov, ki so odvisni od detektiranih značilk (npr. ocena lege mobilnega robota na podlagi značilk). Primer zaznanih značilk v dveh pogledih kamere istega prizora je prikazan na sliki 5.32. Vsaka značilka je določena s pozicijo, orientacijo in velikostjo.

Namen **opisa značilke** je opisati vsako značilko glede na lastnosti slikovnega vzorca okoli nje na način, ki omogoča prepoznavo istih značilk na več slikah (če se te značilke ponovno pojavijo na drugih slikah). Lokalni vzorec okoli značilke (npr. območje, označeno s kvadratom na sliki 5.32) se uporablja za določitev njenega ustreznega deskriptorja, ki je običajno predstavljen kot vektor značilke. Lokalni deskriptorji značilk morajo biti razpoznavni na način, da je možno točno razpoznati značilke ne glede na spremembe v okolju (npr. spremembe osvetlitve, okluzije). Čeprav je mogoče lokalne slikovne značilke primerjati neposredno z npr. konvolucijo območij lokalnih značilk, ta pristop nima ustrezne izrazitosti pa tudi računsko je zelo zahteven. Vektorji deskriptorjev značilk ponavadi predstavljajo minimalno predstavitev značilk, ki omogoča ustrezno raven izrazitosti določene

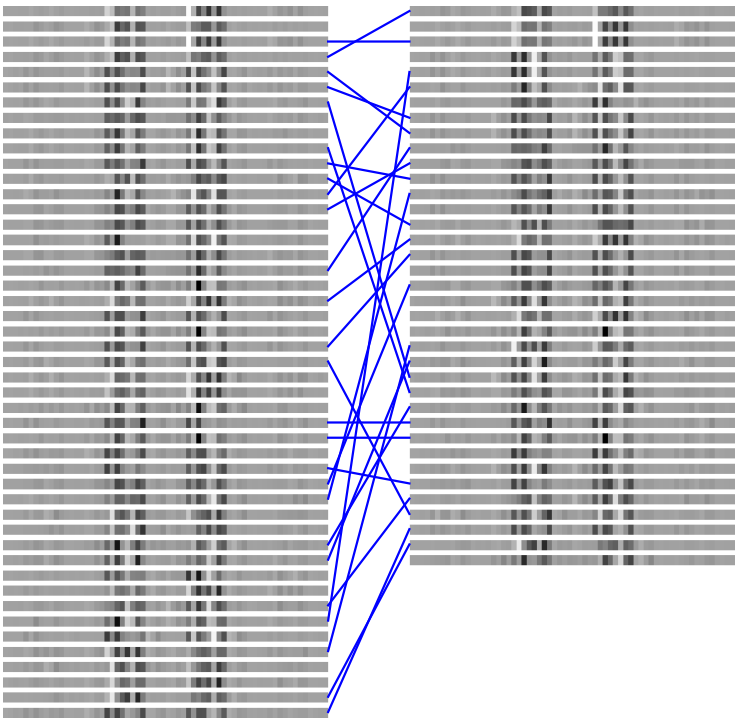


Slika 5.32: Zaznane značilke na dveh slikah istega prizora

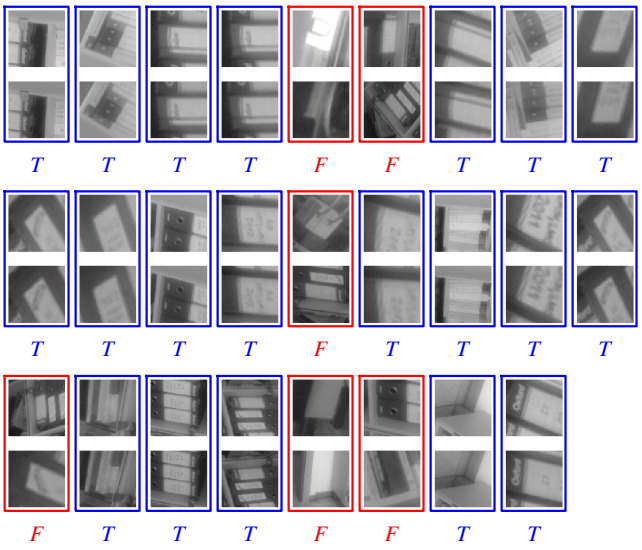
značilke za zanesljivo ujemanje z drugimi. Kateri detektor in način opisa značilke se uporabljata, je odvisno od posamezne aplikacije. Eksperimentalno primerjavo različnih detektorjev značilk lahko najdete v npr. [22]. Primer dveh sklopov vektorjev opisov značilk, izločene iz leve in desne slike na 5.32, je grafično predstavljen na sliki 5.33.

Ujemanje značilk je eden osnovnih problemov v strojnem vidu. Številni algoritmi strojnega vida, kot je ocenjevanje globine prizora, tridimenzionalna rekonstrukcija scene na podlagi slik, ocena lege kamere in drugi, so odvisni od ustreznih ujemanj značilk med več slikami. Značilke se lahko ujemajo s primerjavo razdalj med deskriptorji značilk (vektorji). Odvisno od vrste deskriptorja se lahko uporabijo različne mere razdalje. Za dejanske vrednosti dekriptorjev se običajno uporablja Evklidska razdalja ali razdalja Manhattan, za binarne deskriptorje značilk pa Hammingova razdalja. Ujemanje opisov značilk ne zagotavlja vedno ustreznih ujemanj zaradi nenatančne lokalizacije značilk, njihove popačitve ter ponavljajočih se vzorcev. Zato je potrebno uporabiti ustrezno tehniko ujemanja značilk, ki lahko odpravi zmotne pare.

Glede na dve množici deskriptorjev značilk $\mathcal{A} = \{a_i\}_{i=1,2,\ldots,N_A}$ in $\mathcal{B} = \{b_j\}_{j=1,2,\ldots,N_B}$ mora postopek ujemanja najti ustrezne (ujemajoče se) pare značilk iz obeh množic. Za vsako značilko v množici $\mathcal{A}$ je lahko najbližja značilka v množici $\mathcal{B}$ (glede na izbrano mero razdalje za primerjavo med značilkami) možni kandidat ujemanja. To je v splošnem surjektivna preslikava, saj ima lahko neka značilka iz množice $\mathcal{B}$ več kot samo eno ujemajočo se značilko v množici $\mathcal{A}$. V normalnih pogojih mora imeti funkcija iz ene množice samo eno ujemanje v drugi množici. Zato običajno izvedemo dvosmerno iskanje, kjer za vsako značilko iz množice $\mathcal{A}$ poiščemo najbližjo značilko iz množice $\mathcal{B}$ in obratno, ter upoštevamo samo pare, ki se ujemajo v obeh smereh. Če je razdalja med najboljšim in drugim najboljšim ujemanjem premajhna (pod izbranim pragom), moramo par značilk zavrniti, saj obstaja velika verjetnost napake pri ujemanju. Prav tako zavrnemo par značilk, če je razdalja med izbranimi deskriptorji značilk nad določenim pragom, ker to pomeni preveliko neenakost značilk. Na sliki 5.33 so prikazana ujemanja med dvema množicama deskriptorjev značilk. Čeprav so bile uporabljene zgoraj omenjene tehnike filtriranja, niso odstranjena vsa napačna ujemanja, kot je razvidno iz slik



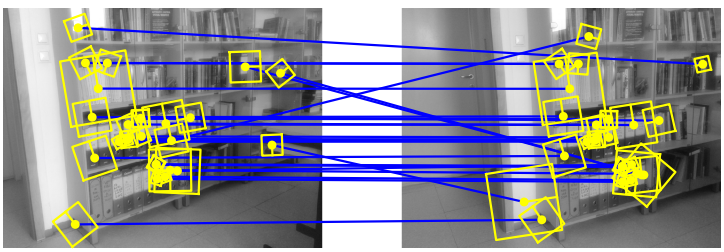
Slika 5.33: Dvosmerna ujemanja med deskriptorji značilk leve in desne slike na sliki 5.32. Vsak vodoravni trak predstavlja vrednosti opisnega vektorja s 64 elementi, ki je predstavljen kot vzorec z različnimi stopnjami intenzitete sive barve.



Slika 5.34: Najdeni pari značilk, ki temeljijo na ujemanju deskriptorjev (glejte sliko 5.33). Večina parov značilk je pravih (T), nekateri pa so tudi zmotni (F).

5.34 in 5.35. Tudi če je samo nekaj osamelcev, lahko ti pomembno vplivajo na rezultate ocenjevalnih algoritmov, ki domnevajo, da se značilke pravilno ujemajo.

Nekatere ujemajoče se značilke ne bodo ustrezale sistemskim omejitvam; npr. če par značilk ne izpolnjuje epipolarne omejitve ali če se rekonstruirana tridimenzionalna točka od para značilk pojavi za kamero, moramo par zavrniti. Več omejitev lahko najdete v [23]. V kolesni mobilni robotiki se lahko uvedejo dodatne omejitve, osnovane na predvidenih stanjih robota iz znanih dejanj in kinematičnega modela. Za filtriranje ujemajočih se parov značilk, ki niso fizično možni, lahko v postopek ujemanja vključimo model nekaterih geometrijskih omejitev. V ta namen je potrebno uporabiti robusten postopek prilagajanja modela. Običajno se uporablja metoda RANSAC (angl. *random sample consensus*) [24], ki omogoča



Slika 5.35: Pari ujemajočih se točk

prileganje modela podatkom, čeprav je veliko osamelcev.

Algoritem RANSAC lahko pojasnimo na problemu prilagajanja premice. V začetnem koraku se iz podatkov naključno izbere najmanjše število N_{min} točk, ki so potrebne za prilagoditev modela. V primeru premice sta potrebni le dve točki ($N_{min} = 2$). Model (premica) se nato namesti na izbrani točki. Okoli nastavljenega modela (premice) je izbrano neko območje zaupanja, znotraj katerega se prešteje število vseh podatkovnih točk. Nato se med podatki izbere drugih N_{min} točk in postopek se ponovi. Po nekaj iteracijah je za najboljši model izbran model z največjim številom točk znotraj območja zaupanja. V zadnjem koraku se model ponovno prilagodi na vse točke v ustreznem območju zaupanja po metodi najmanjših kvadratov.

5.3.5 Ujemanje modelov okolja — zemljevidi

Zemljevid je predstavitev okolja na osnovi nekaterih parametrov značilk, ki so lahko točke odboja laserskega pregledovalnika razdalj, množica daljic, ki predstavljajo robove ovir, množica slikovnih značilk, ki predstavljajo predmete v okolju in podobno.

Problem lokalizacije lahko predstavimo kot optimalno ujemanje dveh zemljevidov: lokalnega in globalnega zemljevida. Lokalni zemljevid, ki ga dobimo iz trenutnih meritev senzorjev, predstavlja del okolja, ki ga je mogoče neposredno opazovati (meriti) iz trenutne lege robota (npr. trenutne točke laserskega pregledovalnika razdalj). Globalni zemljevid je shranjen v notranjem pomnilniku mobilnega sistema in predstavlja znano ali že obiskano območje v okolju. S primerjavo obeh zemljevidov lahko določimo ali izboljšamo trenutno oceno lege mobilnega robota.

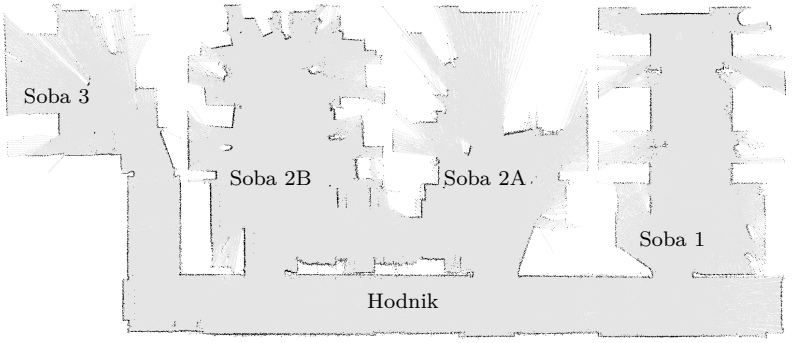
Ko se mobilni sistem giblje v okolju, se lahko globalni zemljevid razširja in sproti posodablja z lokalnim zemljevidom tako, da se trenutne informacije senzorjev, ki predstavljajo prej neodkrita mesta, dodajo globalnemu zemljevidu. Pristop, ki omogoča tovrstno sprotno gradnjo zemljevida, je SLAM (angl. *simultaneous localization and mapping*). Njegovi osnovni koraki so opisani v algoritmu 2.

Algorithm 2 Osnovni koraki algoritma SLAM

Določanje ujemanja lokalnega in globalnega zemljevida na podlagi njunih ujemajočih se parov značilk.
Ocena trenutne lege mobilnega robota na podlagi ujemanja lokalnega in globalnega zemljevida.
Lokalizacija na osnovi kombinacije odometrije in primerjave zemljevidov.
Posodobitev globalnega zemljevida s prej neopaženim delom okolja (nove značilke so dodane kot nova stanja na zemljevidu).

Primer zemljevida, pridobljen s kombinacijo meritev laserskega pregledovalnika razdalj in podatkov odometrije z algoritmom SLAM, je prikazan na sliki 5.36.

SLAM je eden od osnovnih pristopov v kolesni mobilni robotiki. Nekateri pogosti



Slika 5.36: Zemljevid notranjega okolja, zgrajen iz meritev laserskega pregledovalnika razdalj in podatkov odometrije

izzivi pri lokalizaciji mobilnih robotov so:

- **Inicializacije lege** mobilnega robota ob zagonu ni mogoče določiti z veliko verjetnostjo. Če začetna lega ni znana, moramo rešiti problem globalne lokalizacije.
- **Problem ugrabljenega robota** se pojavi, ko se med delovanjem mobilnega robota njegova (resnična) lega v okolju hipno spremeni (npr. mobilni robot je prestavljen na novo lokacijo ali pa ponovno vklopljen na drugi lokaciji). Robustni algoritmi lokalizacije so sposobni obnoviti in oceniti pravo lego robota.
- **Zagotavljanje ustrezne ocene lege med gibanjem** mobilnega robota. V ta namen se uporabljajo podatki odometrije in meritve absolutnih senzorjev.

Najpogosteje uporabljeni algoritmi za reševanje problema SLAM so razširjeni Kalmanov filter, Bayesov filter in filter delcev (glejte poglavje 6).

5.4 Senzorji

V nadaljevanju je podan kratek pregled najpogosteje uporabljenih senzorjev v kolesni mobilni robotiki, njihovih značilnostih in razvrstitev.

5.4.1 Karakteristike senzorjev

Delovanje senzorjev, njihova kakovost in lastnosti so določene z različnimi karakteristikami. Najpogostejše so opisane v nadaljevanju.

Območje določa zgornjo mejo (y_{max}) in spodnjo mejo (y_{min}) uporabljene veličine, ki jo je mogoče izmeriti. Zgornja in spodnja meja ponavadi nista simetrični. Senzorje moramo uporabljati v navedenem območju, saj lahko v nasprotnem primeru pride do poškodbe senzorja.

Dinamično območje je skupno območje od najnižje do največje vrednosti območja. Lahko je podano kot razlika $R_{dyn} = y_{max} - y_{min}$ ali pogosteje kot razmerje (v decibelih) $R_dyn = A \log \frac{y_{max}}{y_{min}}$, kjer je $A = 10$ za meritve, povezane z močjo, in $A = 20$ za ostale.

Ločljivost je najmanjša sprememba merjene veličine, ki jo senzor še zazna. Če ima senzor analogno-digitalni pretvornik, je ločljivost senzorja običajno enaka ločljivosti pretvornika (npr. za 10-bitni A/D in 5 V senzorsko območje je resolucija $\frac{5\,\mathrm{V}}{2^{10}}$).

Občutljivost je sprememba izhodne vrednosti senzorja na enoto veličine, ki jo merimo (npr. senzor razdalje, ki ima na izhodu napetost). Občutljivost je lahko konstantna v celotnem območju senzorja (linearnost) ali pa se spreminja (nelinearnost).

Linearnost je lastnost senzorja, kjer je njegov izhod linearno odvisen (proporcionalen) od merjene veličine v celotnem območju. Linearni senzor ima konstantno občutljivost v celotnem območju.

Histereza se nanaša na lastnost, da je izhodna trajektorija senzorja (ali njena odvisnost od vhoda) drugačna v primeru, ko se vhod senzorja povečuje ali zmanjšuje.

Pasovna širina se nanaša na frekvenco, s katero lahko senzor zagotavlja meritve (v Hz). Je najvišja frekvenca, pri kateri izmerimo samo 70,7 % prave vrednosti.

Točnost je določena s pričakovanim merilnim pogreškom, ki je razlika med izmerjeno m in pravo vrednostjo v. Točnost izračunamo iz relativnega merilnega pogreška kot točnost $= 1 - \frac{|m-v|}{v}$.

Natančnost je stopnja ponovljivosti meritve senzorja pri enaki pravi vrednosti merjene veličine. Pri večkratni meritvi iste prave vrednosti, izhod realnega senzorja poda obseg vrednosti. Natančnost je povezana z varianco meritve.

Sistematični pogrešek ali deterministični pogrešek povzročajo nekateri dejavniki, ki jih je mogoče napovedati ali modelirati (pristranskost, temperaturno lezenje, kalibracija senzorja, popačenje zaradi leče kamere itd.).

Naključni pogrešek ali nedeterministični pogrešek je nepredvidljiv, kar lahko opišemo le z gostoto verjetnosti (npr. normalna porazdelitev). Ta pogrešek se običajno imenuje šum in je označen kot razmerje signal-šum (SNR, angl. *signal-to-noise ratio*).

5.4.2 Klasifikacija senzorjev

Kolesni mobilni robot lahko meri svoje notranje stanje s pomočjo **proprioceptivnih** senzorjev ali zunanje stanje okolja z uporabo **eksteroceptivnih** senzorjev. Primer proprioceptivne meritve je pozicija in orientacija robota, zasuk koles ali krmilnega mehanizma, kotna hitrost koles, stanje akumulatorjev, temperatura ipd. Primer eksteroceptivne meritve pa je razdalja do ovire, slikovni zajem s kamero, mikrofon, kompas, globalni pozicijski sistem (GPS) in drugi.

Senzorji za zaznavanje okolja se uporabljajo za načrtovanje poti, zaznavanje ovir, gradnjo zemljevida itd. Ti senzorji so **aktivni**, če v okolje oddajajo energijo (elektromagnetno valovanje) in meritve prejmejo odziv okolja (laserski merilniki razdalj, ultrazvočni merilniki, kamera z integrirano osvetlitvijo itd.). Senzorji so **pasivni**, če prejmejo energijo, ki je že del okolja. Pasivni senzorji so torej vsi, ki niso aktivni (kamera brez osvetlitve, kompas, žiroskop, pospeškometer itd.).

V tabeli 5.1 so navedeni najpogosteje uporabljeni senzorji v mobilni robotiki glede na njihovo uporabo. Dodatno je podan kratek opis njihove uporabe, namen (proprioceptivni – PC ali eksteroceptivni – EC) in emitirana energija (aktivna – A ali pasivna – P).

Tabela 5.1: Klasifikacija senzorjev v mobilni robotiki glede na njihovo uporabo, namen (proprioceptivni (PC) / eksteroceptivni (EC)) in oddano energijo (aktivna (A) / pasivna (P))

Klasifikacija	Uporabnost	Senzorji	PC/EC	A/P
Taktilni in haptični senzorji	detekcija trkov, varnostni izklop, bližina, rotacija koles ali motorja	kontaktna stikala	EC	P
		taktilni odbijači	EC	P
		optične bariere	EC	A
		merilniki bližine	EC	P/A
		kontaktne letve	EC	P
Osni senzorji	rotacija koles ali motorja, orientacija sklepov, lokalizacija z odometrijo	inkrementalni enkoderji	PC	A
		absolutni enkoderji	PC	A
		potenciometer	PC	P
		tahogenerator	PC	P
Merilniki smeri gibanja	orientacija v referenčnem k. s., lokalizacija, inercialna navigacija	žiroskop	PC	P
		magnetometer	EC	P
		kompas	EC	P
		inklinometer	EC	P
Merilniki hitrosti	inercialna navigacija	pospeškometer	EC	P
		dopplerjev radar	EC	A
		kamera	EC	P
Oddajniki	sledenje objektu, lokalizacija	IR-oddajnik	EC	A
		WiFi-oddajnik	EC	A
		RF-oddajnik	EC	A
		ultrazvočni oddajnik	EC	A
		GPS	EC	A/P
Merilniki na osnovi časa	merjenje razdalje do ovire, časa preleta, lokalizacija	ultrazvočni senzor	EC	A
		laserski merilnik razdalj	EC	A
		kamera	EC	P/A
Merilniki na osnovi strojnega vida	identifikacija, detekcija objektov, sledenje objektom, lokalizacija, segmentacija	kamera	EC	P/A
		globinska kamera	EC	A
		stereo kamera	EC	P/A
		RFID	EC	A
		radar	EC	A
		optična triangulacija	EC	A

Literatura

[1] J. R. Wertz. *Spacecraft Attitude Determination and Control.* D. Reidel Publishing Company, Dordrecht, 1978, 858 str.

[2] F. Chaumette in S. Hutchinson. Visual servo control, part I: basic approaches. *IEEE Robotics and Automation Magazine*, zv. 13, št. 4, str. 82–90, 2006.

[3] J. M. Font-Llagunes in J. A. Batlle. Consistent triangulation for mobile robot localization using discontinuous angular measurements. *Robotics and Autonomous Systems*, zv. 57, št. 9, str. 931–942, 2009.

[4] V. Nguyen, S. Gächter in sod. A comparison of line extraction algorithms using 2D range data for indoor mobile robotics. *Autonomous Robots*, zv. 23, št. 2, str. 97–111, 2007.

[5] T. Pavlidis in S. L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, zv. 23, št. 8, str. 860–870, 1974.

[6] G. Klančar in I. Škrjanc. Evolving principal component clustering with a low run-time complexity for LRF data mapping. *Applied soft computing*, zv. 35, str. 349–358, 2015.

[7] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, zv. 13, št. 2, str. 111–122, 1981.

[8] L. Xu in E. Oja. Randomized hough transform (RHT): Basic mechanisms, algorithms, and computational complexities. *CVGIP: Image Understanding*, zv. 57, št. 2, str. 131–154, 1993.

[9] J. Basak in A. Das. Hough transform network: Learning conoidal structures in a connectionist framework. *IEEE Transactions on Neural Networks*, zv. 13, št. 2, str. 381–392, 2002.

[10] D. Forsyth in J. Ponce. *Computer vision: a modern approach.* Prentice Hall, illustrate izd., 2003, 693 str.

[11] G. Klančar, M. Kristan in sod. Robust and efficient vision system for group of cooperating mobile robots with application to soccer robots. *ISA transactions*, zv. 43, št. 3, str. 329–342, 2004.

[12] P. I. Corke. *Visual control of robots: high-performance visual servoing.* Wiley, New York, 1996, 353 str.

[13] S. Garrido-Jurado, R. Munoz-Salinas in sod. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, zv. 47, št. 6, str. 2280–2292, 2014.

[14] S. Garrido-Jurado, R. Munoz-Salinas in sod. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, zv. 51, str. 481–491, 2016.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, zv. 60, št. 2, str. 91–110, 2004.

[16] H. Bay, A. Ess in sod. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, zv. 110, št. 3, str. 346–359, 2008.

[17] J. Matas, O. Chum in sod. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, zv. 22, št. 10, str. 761–767, 2004.

[18] E. Rosten in T. Drummond. Fusing points and lines for high performance tracking. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, zv. 2, str. 1508–1515, 2005.

[19] E. Mair, G. D. Hager in sod. Adaptive and Generic Corner Detection Based on the

Accelerated Segment Test. V *Computer Vision — ECCV 2010*, str. 183–196. Springer Berlin Heidelberg, 2010.

[20] The OpenCV reference manual: release 2.4.3, 2012.

[21] G. Bradski in A. Kaehler. *Learning OpenCV*. O?Reilly, 1. izd., 2008, 555 str.

[22] K. Mikolajczyk in C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, zv. 27, št. 10, str. 1615–1630, 2005.

[23] B. Cyganek in J. P. Siebert. *An introduction to 3D computer vision techniques and algorithms*. John Wiley & Sons, Chichester, UK, 2009, 483 str.

[24] M. A. Fischler in R. C. Bolles. Random sample consensus: a paradigm for model fitting with applicatlons to image analysis and automated cartography. *Communications of the ACM*, zv. 24, št. 6, str. 381–395, 1981.

6

Nedeterminističnost v mobilnih sistemih

6.1 Uvod

V realnem svetu so deterministični dogodki zelo redki (skoraj ne obstajajo). Poglejmo si nekaj primerov.

Predstavljajmo si primer merjenja hitrosti vozila. Nikoli ne moremo z absolutno natančnostjo določiti prave hitrosti vozila, saj imajo merilniki določeno negotovost (občutljivost, končna ločljivost, omejeno merilno območje, fizikalne omejitve ipd.). Dodatno so meritve senzorjev podvržene šumu in motnjam iz okolice, učinkovitost senzorja se običajno sčasoma spremeni ter senzorji lahko odpovejo ali se pokvarijo. Vsi ti dejavniki omejujejo koristno informacijo (o veličini, ki jo merimo).

Do podobnih zaključkov lahko pridemo tudi pri aktuatorjih. V primeru motornega pogona zaradi šuma, trenja, obrabe, neznane obremenitve ali mehanske okvare ni mogoče določiti prave kotne hitrosti pri danem vzbujanju.

Poleg tega so nekateri algoritmi sami po sebi negotovi. Običajno dajo algoritmi rezultat z omejeno natančnostjo, ki je zadovoljiva za dani problem, saj je potrebno rezultat pridobiti v določenem časovnem roku. Sistemi v mobilni robotiki pogosto delujejo v realnem času, kjer je hitrost računanja pomembnejša od absolutne natančnosti. Zaradi omejene procesne zmogljivosti številni algoritmi niso izvedljivi v realnem času z želenimi odzivnimi časi brez zmanjšanja natančnosti.

Tudi delovno okolje mobilnih agentov (kolesnih mobilnih robotov) je negotovo. Negotovost je običajno nižja v strukturiranih okoljih (npr. stavbe pravilnih oblik)

in višja v časovno spremenljivih dinamičnih okoljih (npr. domovi, cestni promet, bližina ljudi, gozdovi itd.).

Pri razvoju avtonomnega kolesnega mobilnega sistema moramo upoštevati probleme negotovosti in jih uspešno rešiti.

6.2 Osnove verjetnosti

Naj bo X slučajna spremenljivka in x vrednost, ki jo lahko X zavzame. Če je prostor vzorcev slučajne spremenljivke X množica s končnim številom vrednosti (npr. metanje kovanca ima le dva možna izida), je X **diskretna slučajna spremenljivka**. V primeru, da je njen prostor vzorcev množica realnih števil (npr. teža kovanca), je X **zvezna slučajna spremenljivka**. V tem poglavju je podan kratek pregled osnovnih konceptov verjetnostnega računa. Poglobljeno obravnavo tematike najdete v številnih učbenikih o statistiki (npr. [1]).

6.2.1 Diskretna slučajna spremenljivka

Diskretna slučajna spremenljivka X ima končni ali števni prostor vzorcev, ki vsebuje vse možne vrednosti slučajne spremenljivke X. Verjetnost, da diskretna slučajna spremenljivka X zavzame vrednost x, je

$$P(X = x)$$

ali zapisano krajše $P(x) = P(X = x)$. Zaloga vrednosti verjetnostne funkcije $P(x)$ je znotraj omejenega intervala med 0 in 1 za vsako vrednost v prostoru vzorcev X, zapišemo $P(x) \in [0,\ 1]\ \forall\ x \in X$. Vsota verjetnosti vseh možnih vrednosti slučajne spremenljivke X je enaka

$$\sum_{x \in X} P(x) = 1 \tag{6.1}$$

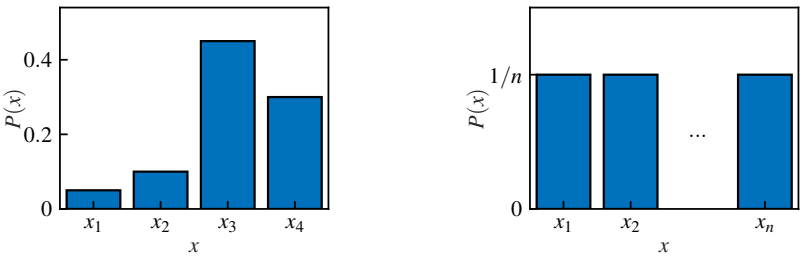
Verjetnost dveh (ali več) dogodkov, ki se pojavljajo skupaj (npr. slučajna spremenljivka X zavzame vrednost x in slučajna spremenljivka Y zavzame vrednost y), je opisana z **verjetnostjo produkta**

$$P(x, y) = P(X = x, Y = y) \tag{6.2}$$

Če sta slučajni spremenljivki X in Y neodvisni, je verjetnost produkta (6.2) preprosto produkt posameznih verjetnosti

$$P(x, y) = P(x)P(y)$$

Pogojna verjetnost $P(x|y)$ podaja verjetnost, da slučajna spremenljivka X zavzame vrednost x, če je predhodno znano, da ima slučajna spremenljivka Y



(a) (b)

Slika 6.1: (a) Primer diskretne porazdelitve in (b) enakomerna diskretna porazdelitev (vsota višin vseh stolpcev je enaka 1)

vrednost y. Če je $P(y) > 0$, lahko določimo pogojno verjetnost z

$$P(x|y) = \frac{P(x,y)}{P(y)} \tag{6.3}$$

V primeru neodvisnih slučajnih spremenljivk X in Y je izračun (6.3) trivialen

$$P(x|y) = \frac{P(x,y)}{P(y)} = \frac{P(x)P(y)}{P(y)} = P(x)$$

Eden od temeljnih rezultatov verjetnostnega računa je **teorem popolne verjetnosti**

$$P(x) = \sum_{y \in Y} P(x,y) \tag{6.4}$$

V primeru, da je na voljo pogojna verjetnost, lahko teorem (6.4) podamo v drugi obliki

$$P(x) = \sum_{y \in Y} P(x|y)P(y) = \boldsymbol{p}^T(x|Y)\boldsymbol{p}(Y) \tag{6.5}$$

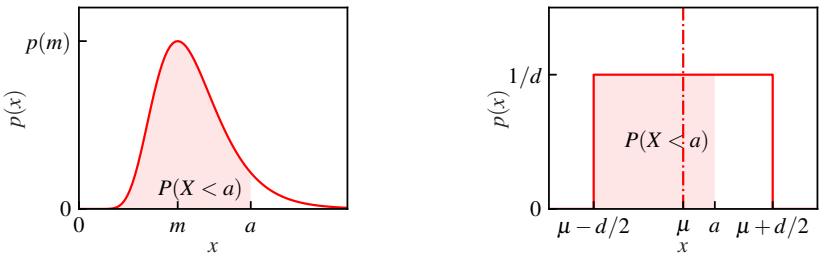
V (6.5) smo uvedli **diskretno porazdelitev**, ki je definirana kot stolpični vektor $\boldsymbol{p}(Y)$

$$\boldsymbol{p}(Y) = \begin{bmatrix} P(y_1) & P(y_2) & \dots & P(y_N) \end{bmatrix}^T$$

kjer je $N \in \mathbb{N}$ število vseh možnih stanj slučajne spremenljivke Y. Podobno je stolpični vektor $\boldsymbol{p}(x|Y)$ definiran kot

$$\boldsymbol{p}(x|Y) = \begin{bmatrix} P(x|y_1) & P(x|y_2) & \dots & P(x|y_N) \end{bmatrix}^T$$

Diskretne porazdelitve se lahko prikažejo s histogrami (slika 6.1). V skladu z (6.1) je vsota višin vseh stolpcev v histogramu enaka 1. Če so vsa stanja slučajne spremenljivke enako verjetna, lahko slučajno spremenljivko opišemo z enakomerno porazdelitvijo (slika 6.1b).



(a) (b)

Slika 6.2: (a) Primer gostote verjetnosti in (b) enakomerna gostota verjetnosti (integral površine pod krivuljo je enak 1)

6.2.2 Zvezna slučajna spremenljivka

Območje zvezne slučajne spremenljivke je (bodisi končen bodisi neskončen) interval realnih števil. V zveznem primeru velja $P(X = x) = 0$, ker ima zvezna slučajna spremenljivka X neskončni prostor vzorcev. Zato je uvedena **gostota verjetnosti** $p(x)$ (angl. *probability density function*), ki ima omejeno območje med 0 in 1, torej je $p(x) \in [0,\ 1]$. Verjetnost, da zvezna slučajna spremenljivke X zavzame vrednost manjšo od a je

$$P(X < a) = \int_{-\infty}^{a} p(x)\,\mathrm{d}x$$

Primera gostote verjetnosti sta prikazana na sliki 6.2. Podobno kot velja relacija (6.1) za diskretno slučajno spremenljivko, je integral gostote verjetnosti celotnega prostora vzorcev zvezne slučajne spremenljivke X enak

$$P(-\infty < X < +\infty) = \int_{-\infty}^{+\infty} p(x)\,\mathrm{d}x = 1$$

Podobna razmerja, ki veljajo za diskretno slučajno spremenljivko (predstavljena v poglavju 6.2.1), se lahko razširijo tudi na gostoto verjetnosti. Nekatere pomembne relacije za diskretne in zvezne slučajne spremenljivke so vzporedno prikazane v tabeli 6.1.

Porazdelitve slučajne spremenljivke so pogosto opisane z različnimi statističnimi parametri. Srednja vrednost zvezne slučajne spremenljivke X je določena kot matematično upanje

$$\mu_X = \mathrm{E}\{X\} = \int_{-\infty}^{+\infty} xp(x)\,\mathrm{d}x$$

Tabela 6.1: Izbrane enačbe verjetnostnega računa za diskretno in zvezno slučajno spremenljivko

Opis	Diskretna slučajna spr.	Zvezna slučajna spr.
Polna verjetnost	$\sum_{x\in X} P(x) = 1$	$\int_{-\infty}^{+\infty} p(x)\,\mathrm{d}x = 1$
Teorem polne verjetnosti	$P(x) = \sum_{y\in Y} P(x,y)$	$p(x) = \int_{-\infty}^{+\infty} p(x,y)\,\mathrm{d}y$
	$P(x) = \sum_{y\in Y} P(x\|y)P(y)$	$p(x) = \int_{-\infty}^{+\infty} p(x\|y)p(y)\,\mathrm{d}y$
Srednja vrednost	$\mu_X = \sum_{x\in X} xP(x)$	$\mu_X = \int_{-\infty}^{+\infty} xp(x)\,\mathrm{d}x$
Varianca	$\sigma_X^2 = \sum_{x\in X} (x-\mu_X)^2 P(x)$	$\sigma_X^2 = \int_{-\infty}^{+\infty} (x-\mu_X)^2 p(x)\,\mathrm{d}x$

Eden od osnovnih parametrov, ki opisujejo obliko porazdelitve, je varianca

$$\sigma_X^2 = \mathrm{var}\{X\} = \mathrm{E}\big\{(X - \mathrm{E}\{X\})^2\big\} = \int_{-\infty}^{+\infty} (x-\mu_X)^2 p(x)\,\mathrm{d}x$$

Ti lastnosti je mogoče določiti tudi za diskretne slučajne spremenljivke in so podane v tabeli 6.1.

Srednja vrednost μ in varianca σ^2 sta edina parametra, ki sta potrebna za enoličen zapis **normalne porazdelitve** (slika 6.3). Normalna porazdelitev je ena izmed najpomembnejših gostot verjetnosti in je predstavljena z Gaussovo funkcijo

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

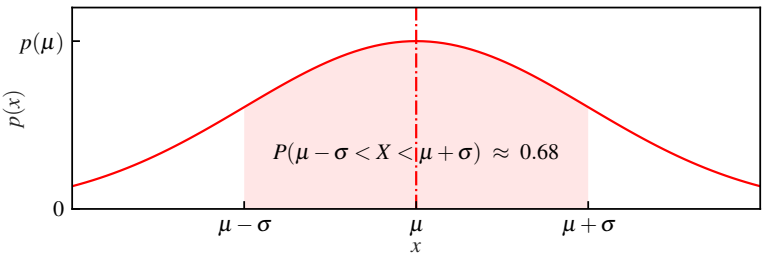
V večdimenzionalnem primeru, ko je slučajna spremenljivka vektor $\boldsymbol{x}$, ima normalna porazdelitev naslednjo obliko

$$p(\boldsymbol{x}) = \det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

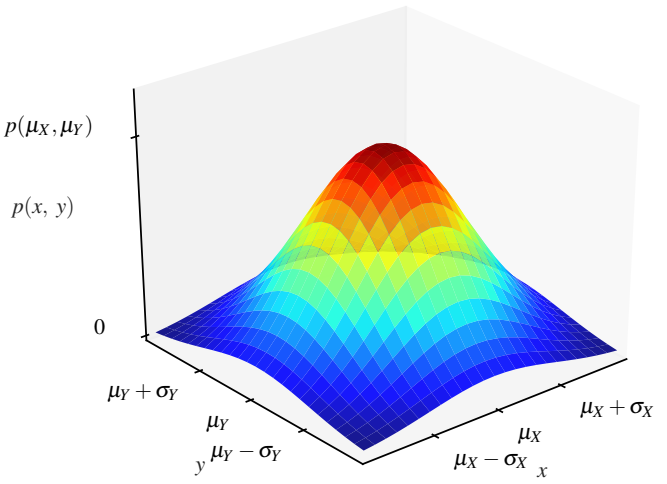
kjer je $\boldsymbol{\Sigma}$ kovariančna matrika. Primer dvodimenzionalne Gaussove funkcije je prikazan na sliki 6.4. Kovariančna matrika je simetrična — element v vrstici i in stolpcu j je kovarianca $\mathrm{cov}\{X_i, X_j\}$ med slučajnima spremenljivkama X_i in X_j.

Kovarianca $\mathrm{cov}\{X,Y\}$ je merilo linearnega razmerja med slučajnima spremenljivkama X in Y

$$\begin{aligned}\sigma_{XY}^2 = \mathrm{cov}\{X,Y\} &= \mathrm{E}\{(X-\mu_X)(Y-\mu_Y)\} = \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} (X-\mu_X)(Y-\mu_Y)p(x,y)\,\mathrm{d}x\,\mathrm{d}y\end{aligned} \tag{6.6}$$



Slika 6.3: Normalna porazdelitev (gostota verjetnosti)



Slika 6.4: Dvodimenzionalna Gaussova gostota verjetnosti

kjer $p(x, y)$ predstavlja gostoto verjetnosti produkta X in Y. Relacijo (6.6) lahko poenostavimo na

$$\sigma_{XY}^2 = \mathrm{E}\{XY\} - \mu_X \mu_Y \tag{6.7}$$

Če sta slučajni spremenljivki X in Y **neodvisni**, velja $\mathrm{E}\{XY\} = \mathrm{E}\{X\}\mathrm{E}\{Y\}$ in kovarianca (6.7) ima vrednost 0: $\sigma_{XY}^2 = 0$. Obratno ne velja — če je kovarianca nič, to ne pomeni, da sta slučajni spremenljivki neodvisni. Kovarianca dveh enakih slučajnih spremenljivk je varianca $\mathrm{cov}\{X, X\} = \mathrm{var}\{X\} = \sigma_X^2$.

6.2.3 Bayesovo pravilo

Bayesovo pravilo je eden od temeljnih stebrov teorije verjetnosti in ima veliko uporabno vrednost v mobilni robotiki. V zveznem prostoru je podano kot

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{6.8}$$

in za diskreten prostor kot

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{6.9}$$

Bayesovo pravilo omogoča, da izračunamo težko določljivo verjetnost na podlagi verjetnosti, ki jo je lažje določiti.

Primer 6.1

Predstavimo uvodni primer uporabe Bayesovega pravila (6.8) v mobilnih sistemih. Naj slučajna spremenljivka X predstavlja stanje mobilnega sistema, ki ga želimo oceniti (npr. razdalja mobilnega sistema od ovire) na podlagi meritve Y, ki je stohastično odvisna od slučajnega stanja X. Ker je izid meritve negotov, želimo izvedeti, kakšna je gostota verjetnosti ocenjenega stanja $X = x$ na podlagi meritve $Y = y$.

Rešitev

Zanima nas gostota verjetnosti $p(x|y)$, ki jo lahko izračunamo iz (6.8). Gostota verjetnosti $p(x)$ vsebuje znanje o slučajni spremenljivki X pred opravljeno meritvijo y, zato jo imenujemo **napovedna ocena** (angl. *a-priori estimate*). Pogojna gostota verjetnosti $p(x|y)$ podaja znanje o slučajni spremenljivki X po opravljeni meritvi in je zato znana tudi kot **trenutna ocena** (angl. *a posteriori estimate*). Gostota verjetnosti $p(y|x)$ vsebuje informacijo o vplivu stanja X na meritev Y, zato predstavlja model senzorja (npr. porazdelitev merjenja razdalje do ovire, če je mobilni robot na določeni razdalji od ovire). Gostota verjetnosti $p(y)$ vsebuje porazdelitev meritve y, torej zaupanje v opravljeno meritev, in jo lahko določimo s teoremom popolne verjetnosti $p(y) = \int p(y|x)p(x)\,\mathrm{d}x$. Zato lahko trenutno oceno $p(x|y)$ pridobimo s pomočjo znanega statističnega modela senzorja ($p(y|x)$) in napovedne ocene (gostote verjetnosti stanja $p(x)$).

Primer 6.2

Do ciljne lokacije vodijo tri različne poti. Mobilni sistem izbere prvo pot v sedmih od desetih primerov, drugo pot samo v enem od desetih, tretjo pa v enem od

petih primerov. Na prvi poti ima 5 %, na drugi 10 % in na tretji 8 % verjetnost naleta na oviro.

1. Kakšna je verjetnost, da mobilni sistem na poti do cilja naleti na oviro?
2. Mobilni sistem je naletel na oviro. Kakšna je verjetnost, da se je to zgodilo na prvi poti?

Rešitev

Označimo prvo, drugo in tretjo pot z A_1, A_2 in A_3; oviro na katerikoli poti pa z B. Verjetnost, da izberemo določeno pot je: $P(A_1) = 0{,}7$, $P(A_2) = 0{,}1$, $P(A_3) = 0{,}2$. Verjetnost naleta na oviro na prvi poti je $P(B|A_1) = 0{,}05$, na drugi poti $P(B|A_2) = 0{,}1$ in na tretji poti $P(B|A_3) = 0{,}08$.

1. Verjetnost, da mobilni sistem naleti na oviro, je

$$\begin{aligned} P(B) &= P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + P(B|A_3)P(A_3) = \\ &= 0{,}05 \cdot 0{,}7 + 0{,}1 \cdot 0{,}1 + 0{,}08 \cdot 0{,}2 = \\ &= 0{,}061 \end{aligned}$$

2. Verjetnost, da mobilni sistem obtiči na prvi poti, lahko izračunamo s pomočjo Bayesovega pravila (6.9):

$$\begin{aligned} P(A_1|B) &= \frac{P(B|A_1)P(A_1)}{P(B)} = \\ &= \frac{P(B|A_1)P(A_1)}{P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + P(B|A_3)P(A_3)} = \\ &= \frac{0{,}05 \cdot 0{,}7}{0{,}05 \cdot 0{,}7 + 0{,}1 \cdot 0{,}1 + 0{,}08 \cdot 0{,}2} = \\ &= 0{,}5738 \end{aligned}$$

Z Matlab kodo v programu 6.1 lahko izračunamo verjetnosti, da mobilni robot obtiči na katerikoli poti.

Program 6.1

./src/prb/example_three_paths.m

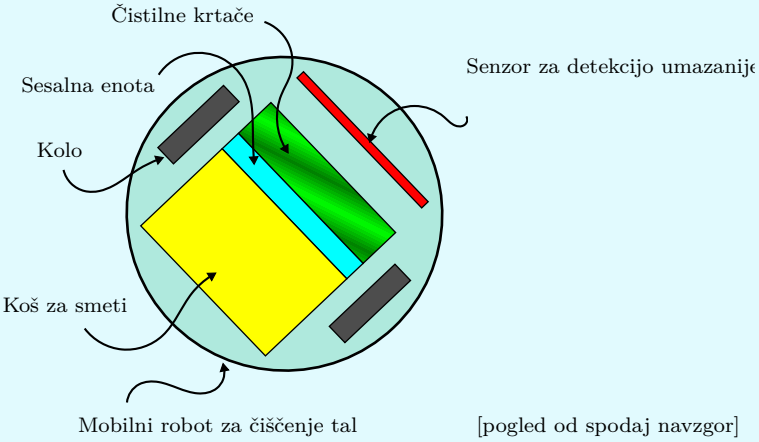
```
1 % Verjetnost izbire prve, druge ali tretje poti:
2 % p(A) = [P(A1), P(A2), P(A3)]
3 p_A = [0.7 0.1 0.2]
4 % Verjetnost ovire na prvi, drugi in tretji poti:
5 % p(B|A) = [P(B|A1), P(B|A2), P(B|A3)]
6 p_BA = [0.05 0.1 0.08]
7
```

```
% Verjetnost ovire: P(B)
P_B = p_BA*p_A.'

% Verjetnost zastoja na prvi, drugi in tretji poti:
% p(A|B) = [P(A1|B), P(A2|B), P(A3|B)]
p_AB = (p_BA.*p_A)./P_B
```

```
p_A =
    0.7000    0.1000    0.2000
p_BA =
    0.0500    0.1000    0.0800
P_B =
    0.0610
p_AB =
    0.5738    0.1639    0.2623
```

Primer 6.3

Mobilni robot za čiščenje tal je opremljen s senzorjem za zaznavanje umazanije, ki lahko zaznava čistočo tal pod mobilnim robotom (slika 6.5).



Slika 6.5: Mobilni robot za čiščenje tal

Na podlagi odčitkov senzorja želimo ugotoviti, ali so tla pod robotom čista ali ne, zato ima diskretna slučajna spremenljivka dve možni vrednosti. Verjetnost, da so tla čista, je 40 %. O senzorju za zaznavanje umazanije lahko trdimo naslednje: če so tla čista, senzor to pravilno zazna v 80 % primerov; in če so tla umazana, senzor to pravilno zazna v 90 % primerov. V primeru čistih tal je verjetnost nepravilne meritve majhna, saj bo senzor napačno določil stanje tal v enem od petih primerov. V primeru umazanih tal je verjetnost nepravilne meritve še manjša, saj bo senzor napačno zaznal samo v enem od desetih primerov. Kakšna

je verjetnost, da so tla čista, če senzor zazna, da so le-ta čista?

Rešitev

Označimo stanje tal — čista (*clean*) ali umazana (*dirty*) — z diskretno slučajno spremenljivko $X \in \{clean,\ dirty\}$ in meritev senzorja s slučajno spremenljivko $Z \in \{clean,\ dirty\}$. Verjetnost, da so tla čista, torej zapišemo kot $P(X = clean) = 0{,}4$ in model meritve senzorja lahko matematično predstavimo kot

$$\begin{aligned} P(Z = clean|X = clean) &= 0{,}8 \\ P(Z = dirty|X = dirty) &= 0{,}9 \end{aligned}$$

Uvedimo krajši zapis

$$\begin{aligned} P(x) &= P(X = clean) = 0{,}4 \\ P(\bar{x}) &= P(X = dirty) = 1 - P(x) = 0{,}6 \\ P(z|x) &= P(Z = clean|X = clean) = 0{,}8 \\ P(\bar{z}|x) &= P(Z = dirty|X = clean) = 1 - P(z|x) = 0{,}2 \\ P(\bar{z}|\bar{x}) &= P(Z = dirty|X = dirty) = 0{,}9 \\ P(z|\bar{x}) &= P(Z = clean|X = dirty) = 1 - P(\bar{z}|\bar{x}) = 0{,}1 \end{aligned}$$

S pomočjo Bayesovega pravila (6.9) določimo $P(x|z)$

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)}$$

Nadalje lahko s pomočjo teorema popolne verjetnosti izračunamo verjetnost, da senzor zazna tla kot čista

$$\begin{aligned} P(z) &= P(z|x)P(x) + P(z|\bar{x})P(\bar{x}) = \\ &= 0{,}8 \cdot 0{,}4 + 0{,}1 \cdot 0{,}6 = \\ &= 0{,}38 \end{aligned}$$

Tako je iskana verjetnost enaka

$$P(x|z) = \frac{0{,}8 \cdot 0{,}4}{0{,}38} = 0{,}8421$$

Dobljena verjetnost, da senzor zazna čista tla kot čista, je visoka. Vendar pa obstaja več kot 15 % verjetnost, da so tla dejansko umazana, zato lahko mobilni sistem napačno domneva, da čiščenje ni potrebno. Če mobilni robot v takšnem primeru ne izvede čiščenja, ostanejo tla umazana.

6.3 Ocenjevanje stanj

Ocenjevanje stanja je proces, v katerem so prava stanja sistema ocenjena na podlagi izmerjenih podatkov in predhodnega znanja o sistemu. Tudi če je možno stanja sistema neposredno izmeriti, izmerjeni podatki običajno vsebujejo šum in druge motnje. Zaradi tega so neobdelani izmerjeni podatki običajno neprimerni za nadaljnjo uporabo brez ustreznega filtriranja (npr. neposredna uporaba pri izračunu pogreška vodenja). V številnih primerih je možno oceniti stanja sistema, tudi če stanja niso neposredno merljiva. To lahko izvedemo, če je sistem *spoznaven*. Zato moramo pred ocenjevanjem stanj preveriti spoznavnost sistema. Najpomembnejši lastnosti algoritmov za ocenjevanje stanja sta: ocena konvergence in ocenjevalna pristranskost. V tem poglavju bomo opisali nekaj praktičnih vidikov, ki jih je potrebno upoštevati pred izvedbo določenega algoritma za ocenjevanje stanj.

6.3.1 Motnje in šum

Vso neupoštevano sistemsko dinamiko — kot so nemerljivi signali in pogreški modeliranja — lahko razumemo kot sistemske motnje. Pod predpostavko linearnosti lahko vse motnje predstavimo v enem samem členu $n(t)$, ki ga prištejemo pravemu signalu $y_0(t)$

$$y(t) = y_0(t) + n(t)$$

Motnje razvrstimo v več razredov: visokofrekvenčni kvazistacionarni stohastični signali (npr. merilni šum), nizkofrekvenčni nestacionarni signali (npr. lezenje), periodični signali ali kakšen drug tip signalov (npr. špice, osamelci). Eden od najpomembnejših stohastičnih signalov je **beli šum**.

Frekvenčni spekter in porazdelitev signala sta najpomembnejši lastnosti, ki opisujeta signal. Porazdelitev signala poda verjetnost, s katero amplituda zavzame določeno vrednost. Najpogostejši porazdelitvi signalov sta enakomerna porazdelitev in Gaussova (normalna) porazdelitev (glejte poglavje 6.2). Frekvenčni spekter signala predstavlja soodvisnost signala v vsakem trenutku, ki je povezan s porazdelitvijo frekvenčnih komponent signala. V primeru belega šuma so vse frekvenčne komponente enakomerno porazdeljene, zato je vrednost signala v vsakem časovnem trenutku neodvisna od prejšnjih vrednosti signala.

6.3.2 Ocena konvergence in pristranskosti

Kot smo že omenili, ocenjevanje stanj podaja ocene notranjih stanj glede na izmerjene vhodno-izhodne signale, razmerje med spremenljivkami (modela sistema), nekatere statistične lastnosti signalov (npr. varianca) in druge informacije o sistemu. Vse te podatke je treba združiti tako, da dobimo točno in natančno oceno notranjih stanj. Na žalost so meritve, model in vnaprej znane lastnosti

signalov same po sebi negotove zaradi šuma, motenj, parazitske dinamike, napačnih predpostavk o modelu sistema in drugih virov napak. Zato se ocene stanj običajno razlikujejo od dejanskih stanj.

Vse zgoraj omenjene težave povzročajo določeno stopnjo negotovosti signalov. Matematično lahko problem rešimo v stohastičnem okolju, kjer se signali obravnavajo kot slučajne spremenljivke. Signali so tako predstavljeni s pripadajočimi gostotami verjetnosti ali pa s srednjo vrednostjo in varianco. V stohastičnem okolju je pomembno vprašanje kakovost določene ocene stanja. Zlasti je potrebno analizirati konvergenco ocene stanja glede na pravo vrednost. Zastaviti si moramo dve pomembni vprašanji:

1. *Ali je matematično upanje ocene enako pravi vrednosti?* Če to drži, je ocena **nepristranska**. Ocena je **konsistentna**, če se izboljša s časom (večji interval opazovanja) in konvergira k pravi vrednosti s podaljševanjem intervala opazovanja proti neskončnosti.

2. *Ali varianca pogreška ocene konvergira proti* 0*, ko gre čas opazovanja proti neskončnosti?* Če to drži in je ocena konsistentna, je ocena **konsistentna v srednjekvadratični vrednosti**. To pomeni, da se z naraščanjem časa opazovanja točnost in natančnost ocen izboljšuje (vse ocene so v bližini prave vrednosti).

Na zgornji vprašanji lahko razmeroma enostavno odgovorimo, če so predpostavke o sistemu preproste (popoln model sistema, Gaussov šum itd.). Pri obravnavi zahtevnejšega problema je zelo težko najti odgovore in analitične rešitve niso več možne. Vendar se moramo zavedati, da ocenjevalniki stanj podajajo zadovoljive rezultate, če niso kršene nekatere pomembne predpostavke. Zato je izjemno pomembno, da pred uporabo določenega algoritma ocenjevanja stanj preberemo drobni tisk, saj lahko v nasprotnem primeru algoritem poda ocene, ki so daleč od dejanskih stanj. Problem je, da se tega ponavadi sploh ne zavedamo.

6.3.3 Spoznavnost

Stanja, ki jih je potrebno oceniti, so običajno skrita, saj navadno informacije o njih niso neposredno dostopne. Stanja lahko ocenimo na podlagi meritev izhodov sistema, ki so neposredno ali posredno odvisni od stanj. Preden izvedemo algoritem ocenjevanja, moramo odgovoriti na naslednje vprašanje: *Ali lahko enoličnо ocenimo stanja v končnem času na podlagi opazovanja izhodov sistema?* Odgovor na to vprašanje podaja **spoznavnost** sistema. Če je sistem spoznaven, lahko ocenimo stanja z opazovanjem njegovih izhodov. Sistem je le delno spoznaven, če lahko ocenimo samo podmnožico stanj sistema, pri čemer preostalih stanj morda ne bomo ocenili. V primeru popolnoma spoznavnega sistema so vsa stanja in izhodi povezani.

Preden začnemo z opredelitvijo spoznavnosti, moramo vpeljati koncept **nerazpoznavnih stanj** (angl. *indistinguishable state*). Predstavljajmo si splošen nelinearen sistem v obliki ($\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{u} \in \mathbb{R}^m$ in $\boldsymbol{y} \in \mathbb{R}^l$)

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{y}(t) &= \boldsymbol{h}(\boldsymbol{x}(t)) \end{aligned} \tag{6.10}$$

Stanji $\boldsymbol{x}_0$ in $\boldsymbol{x}_1$ sta nerazpoznavni, če za vsak dopusten vhod $\boldsymbol{u}(t)$ v končnem časovnem intervalu $t \in [t_0,\ t_1]$ dobimo enake izhode [2]

$$\boldsymbol{y}(t, \boldsymbol{x}_0) \equiv \boldsymbol{y}(t, \boldsymbol{x}_1) \quad \forall\ t \in [t_0,\ t_1]$$

Množica vseh nerazpoznavnih stanj iz stanja $\boldsymbol{x}_0$ je označena kot $\mathcal{I}(\boldsymbol{x}_0)$. Sedaj lahko rečemo, da je sistem **spoznaven pri** $\boldsymbol{x}_0$, če množica nerazpoznavnih stanj $\mathcal{I}(\boldsymbol{x}_0)$ vsebuje samo stanje $\boldsymbol{x}_0$, torej $\mathcal{I}(\boldsymbol{x}_0) = \{\boldsymbol{x}_0\}$. Sistem je **spoznaven**, če množica nerazpoznavnih stanj $\mathcal{I}(\boldsymbol{x})$ vsebuje samo stanje $\boldsymbol{x}$ za vsako stanje $\boldsymbol{x}$ v definicijskem območju, torej $\mathcal{I}(\boldsymbol{x}) = \{\boldsymbol{x}\}\ \forall\ \boldsymbol{x}$. Spoznavnost ne pomeni, da je ocena stanja $\boldsymbol{x}$ opazovanega izhoda možna za vsak vhod $\boldsymbol{u}(t)$, $t \in [t_0,\ t_1]$. V določenih primerih je za razlikovanje med stanji potreben dolg čas opazovanja. Poznamo različne načine spoznavnosti [2]: *lokalna spoznavnost* ali močnejši koncept spoznavnosti, *šibka spoznavnost* ali oslabljeni koncept spoznavnosti in *lokalna šibka spoznavnost*. V primeru avtonomnih linearnih sistemov so vse tri oblike spoznavnosti enakovredne.

Preverjanje spoznavnosti splošnih nelinearnih sistemov (6.10) zahteva napredno matematično analizo. **Lokalno šibko spoznavnost** sistema lahko preverimo s preprostim algebraičnim testom. V ta namen vpeljemo operator **Liejev odvod** kot časovni odvod izhodne funkcije $\boldsymbol{h}$ vzdolž trajektorije sistema $\boldsymbol{x}$

$$L_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] = \frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$$

Vzemimo avtonomen sistem (6.10) ($\boldsymbol{u}(t) = \boldsymbol{0}$ za vsak t). Lokalno šibko spoznavnost preverimo tako, da (večkrat) odvajamo sistemski izhod $\boldsymbol{y}$, dokler se ne poveča rang matrike $\boldsymbol{Q}$ (definicija sledi v nadaljevanju)

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{h}(\boldsymbol{x}) = L^0_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] \\ \dot{\boldsymbol{y}} &= \frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) = L_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] = L^1_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] \\ \ddot{\boldsymbol{y}} &= \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) \right) \boldsymbol{f}(\boldsymbol{x}) = L_{\boldsymbol{f}}[L_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})]] = L^2_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] \\ &\vdots \\ \frac{\mathrm{d}^i \boldsymbol{y}}{\mathrm{d}t^i} &= L^i_{\boldsymbol{f}}[\boldsymbol{h}(\boldsymbol{x})] \end{aligned} \tag{6.11}$$

Časovne odvode izhoda sistema (6.11) lahko zapišemo v matriko $\boldsymbol{L}(\boldsymbol{x})$

$$\boldsymbol{L}(\boldsymbol{x}) = \begin{bmatrix} L_{\boldsymbol{f}}^0[\boldsymbol{h}(\boldsymbol{x})] \\ L_{\boldsymbol{f}}^1[\boldsymbol{h}(\boldsymbol{x})] \\ L_{\boldsymbol{f}}^2[\boldsymbol{h}(\boldsymbol{x})] \\ \vdots \\ L_{\boldsymbol{f}}^i[\boldsymbol{h}(\boldsymbol{x})] \end{bmatrix} \tag{6.12}$$

Rang matrike $\boldsymbol{Q}(\boldsymbol{x}_0) = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{L}(\boldsymbol{x})|_{\boldsymbol{x}_0}$ določa lokalno šibko spoznavnost sistema pri $\boldsymbol{x}_0$. Če je rang matrike $\boldsymbol{Q}(\boldsymbol{x}_0)$ enak številu stanj, torej $\text{rang}(\boldsymbol{Q}(\boldsymbol{x}_0)) = n$, naj bi sistem zadostoval **spoznavnostnemu pogoju za rang pri** $\boldsymbol{x}_0$, ki je zadosten, ne pa potreben pogoj za lokalno šibko spoznavnost sistema pri $\boldsymbol{x}_0$. Če je za vsak $\boldsymbol{x}$ iz definicijskega območja izpolnjen spoznavnostni pogoj za rang, je sistem lokalno šibko spoznaven. Bolj podrobna študija na temo spoznavnosti je na voljo v [2–4].

Pogoj za rang spoznavnosti se lahko poenostavi za časovno nespremenljive linearne sisteme. Za sistem z n stanji v obliki $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$, $\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t)$ so Liejevi odvodi (6.12)

$$\begin{aligned} L_{\boldsymbol{f}}^0[\boldsymbol{h}(\boldsymbol{x})] &= \boldsymbol{C}\boldsymbol{x}(t) \\ L_{\boldsymbol{f}}^1[\boldsymbol{h}(\boldsymbol{x})] &= \boldsymbol{C}(\boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)) \\ L_{\boldsymbol{f}}^2[\boldsymbol{h}(\boldsymbol{x})] &= \boldsymbol{C}\boldsymbol{A}(\boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)) \\ &\vdots \end{aligned} \tag{6.13}$$

Delno odvajanje Liejevih odvodov (6.13) privede do **Kalmanove spoznavnostne matrike** $\boldsymbol{Q}$

$$\boldsymbol{Q}^T = \begin{bmatrix} \boldsymbol{C}^T & \boldsymbol{A}^T\boldsymbol{C}^T & \dots & (\boldsymbol{A}^T)^{n-1}\boldsymbol{C}^T \end{bmatrix}$$

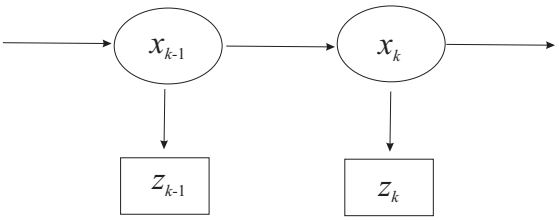
Sistem je spoznaven, če ima spoznavnostna matrika n neodvisnih vrstic, torej je rang spoznavnostne matrike enak številu stanj

$$\text{rang}(\boldsymbol{Q}) = n$$

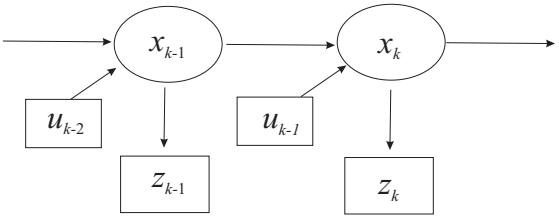
6.4 Bayesov filter

6.4.1 Markovove verige

Obravnavali bomo sisteme, za katere lahko predpostavimo, da je **stanje vsebovano**. To pomeni, da se vse informacije o sistemu v danem trenutku lahko pridobijo iz stanj sistema. Sistem lahko opišemo na podlagi trenutnih stanj, kar je lastnost Markovovega procesa. Na sliki 6.6 je prikazan *skriti Markovov proces*, kjer stanja niso neposredno dostopna, ampak jih lahko samo ocenimo iz meritev,



Slika 6.6: Skriti Markovov proces. Meritev z_k je stohastično odvisna od trenutnega stanja x_k, ki ni neposredno dostopno in je odvisno od prejšnjega stanja x_{k-1}



Slika 6.7: Skriti Markovov proces z zunanjimi vhodi. Meritev z_k je stohastično odvisna od trenutnega stanja x_k. Stanje x_k ni neposredno dostopno ter je odvisno od prejšnjega stanja x_{k-1} in trenutno aktualnega vhoda u_{k-1}.

ki so stohastično odvisne od trenutnih vrednosti stanj. Zaradi tega je trenutno stanje sistema odvisno le od prejšnjega stanja in ne celotne zgodovine stanj

$$p(x_k|x_0, \ldots, x_{k-1}) = p(x_k|x_{k-1})$$

Podobno je tudi meritev neodvisna od celotne zgodovine stanj sistema, če je znano le trenutno stanje

$$p(z_k|x_0, \ldots, x_k) = p(z_k|x_k)$$

Struktura skritega Markovovega procesa, kjer je trenutno stanje x_k odvisno od prejšnjega stanja x_{k-1} in vhoda sistema u_{k-1}, je prikazana na sliki 6.7. Vhod u_{k-1} je trenutno aktualen vhod, ki vpliva na notranja stanja od trenutka $k-1$ do k.

6.4.2 Ocenjevanje stanj iz opazovanj

Bayesov filter predstavlja najbolj splošen algoritem za izračun porazdelitve. Bayesov filter je močno statistično orodje, ki ga lahko uporabimo za oceno lokacije (stanj sistema) ob prisotnosti sistemskih in merilnih negotovosti [5].

Porazdelitev stanja po merjenju $p(x|z)$ lahko ocenimo, če sta znana statistični model senzorja $p(z|x)$ (porazdelitev izida meritve ob poznanem stanju) in porazdelitev meritve $p(z)$. To smo predstavili v primeru 6.3 za diskretno slučajno spremenljivko.

Poglejmo si primer ocenjevanja stanja x, ko imamo več zaporednih meritev z. Želimo oceniti porazdelitev $p(x_k|z_1, \ldots, z_k)$ stanja x v trenutku k, pri čemer upoštevamo zaporedje vseh meritev do aktualnega trenutka. Bayesovo formulo lahko podamo v rekurzivni obliki

$$p(x_k|z_1, \ldots, z_k) = \frac{p(z_k|x_k, z_1, \ldots, z_{k-1})p(x_k|z_1, \ldots, z_{k-1})}{p(z_k|z_1, \ldots, z_{k-1})}$$

kar lahko krajše zapišemo kot

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k, z_{1:k-1})p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{6.14}$$

Pomen posameznih členov v (6.14) je:

- $p(x_k|z_{1:k})$ je ocenjena porazdelitev stanja v trenutku k, posodobljenega z merilnimi podatki,
- $p(z_k|x_k, z_{1:k-1})$ je porazdelitev meritve v trenutku k, če poznamo trenutno stanje x_k in pretekle meritve do trenutka $k-1$,
- $p(x_k|z_{1:k-1})$ je napovedana porazdelitev stanja na osnovi preteklih meritev,
- $p(z_k|z_{1:k-1})$ je porazdelitev opravljene meritve (zaupanje v opravljeno meritev) v trenutku k.

Nadalje velja, da je trenutna meritev z_k v (6.14) neodvisna od preteklih meritev $z_{1:k-1}$ (stanje je vsebovano, Markovov proces), če poznamo stanje sistema x_k

$$p(z_k|x_k, z_{1:k-1}) = p(z_k|x_k)$$

Zato se enačba (6.14) poenostavi v

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{6.15}$$

Izpeljava enačbe (6.15) je podana v primeru 6.4.

Primer 6.4

Za vajo izpeljite (6.15) iz (6.14), pri čemer predpostavite vsebovano stanje ($p(z_k|x_k, z_{1:k-1}) = p(z_k|x_k)$).

Rešitev

Izpeljava porazdelitve (6.15)

$$\begin{aligned}
p(x_k|z_{1:k}) &= \frac{p(z_{1:k}|x_k)p(x_k)}{p(z_{1:k})} = \\
&= \frac{p(z_k, z_{1:k-1}|x_k)p(x_k)}{p(z_k, z_{1:k-1})} = \\
&= \frac{p(z_k|z_{1:k-1}, x_k)p(z_{1:k-1}|x_k)p(x_k)}{p(z_k|z_{1:k-1})p(z_{1:k-1})} = \\
&= \frac{p(z_k|z_{1:k-1}, x_k)p(x_k|z_{1:k-1})p(z_{1:k-1})p(x_k)}{p(z_k|z_{1:k-1})p(z_{1:k-1})p(x_k)} = \\
&= \frac{p(z_k|z_{1:k-1}, x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} = \\
&= \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})}
\end{aligned}$$

Rekurzivna enačba (6.15) za posodobitev stanja na podlagi preteklih meritev vsebuje tudi predikcijo $p(x_k|z_{1:k-1})$. Postopek ocenjevanja stanja lahko razdelimo na *predikcijski* in *korekcijski* korak.

Predikcijski korak

Predikcijo $p(x_k|z_{1:k-1})$ določimo kot

$$p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1}, z_{1:k-1})p(x_{k-1}|z_{1:k-1})\,\mathrm{d}x_{k-1}$$

kar lahko poenostavimo s predpostavko, da je stanje vsebovano

$$p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})\,\mathrm{d}x_{k-1} \tag{6.16}$$

kjer $p(x_k|x_{k-1})$ predstavlja porazdelitev prehajanja med stanji in $p(x_{k-1}|z_{1:k-1})$ je popravljena porazdelitev ocenjenega stanja iz prejšnjega časovnega trenutka.

Korekcijski korak

Ocenjena porazdelitev stanj po opravljeni meritvi v trenutku k in izračunana napovedana porazdelitev stanj v predikcijskem koraku je

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{6.17}$$

Verjetnost $p(z_k|z_{1:k-1})$, ki opisuje merilno zaupanje, lahko določimo s pomočjo relacije

$$p(z_k|z_{1:k-1}) = \int p(z_k|x_k)p(x_k|z_{1:k-1})\,\mathrm{d}x_k$$

Najverjetnejša ocena stanj

Kako lahko ocenjeno porazdelitev $p(x_k|z_{1:k})$ uporabimo pri oceni najverjetnejšega stanja x_k? Najbolj verjetna ocena stanja (matematično upanje) $\mathrm{E}\{x_k\}$ je podana kot vrednost, ki minimizira povprečni kvadratni pogrešek meritve

$$\mathrm{E}\{x_k\} = \int x_k p(x_k|z_{1:k})\,\mathrm{d}x_k$$

Ocenimo lahko tudi vrednost $x_{k_{max}}$, ki maksimira trenutno verjetnost $p(x_k|z_{1:k})$

$$x_{k_{max}} = \max_{x_k} p(x_k|z_{1:k})$$

Primer 6.5

Imamo senzor iz primera 6.3. Kakšna je verjetnost čistih tal, če senzor v trenutku $k = 2$ ponovno zazna čista tla?

Rešitev

V primeru 6.3 so bila v trenutku $k = 1$ tla čista, senzor pa je tudi zaznal, da so čista ($z_1 = clean$). Izračunali smo pogojno verjetnost

$$P(x_1|z_1) = \frac{0{,}8 \cdot 0{,}4}{0{,}38} = 0{,}8421$$

V naslednjem trenutku $k = 2$ senzor vrne $z_2 = clean$ z verjetnostjo pravilne zaznave senzorja $P(z_2|x_2) = 0{,}8$ in z verjetnostjo napačne zaznave senzorja $P(z_2|\bar{x}_2) = 0{,}1$ (predpostavljamo časovno nespremenljivo karakteristiko senzorja).

Najprej ovrednotimo predvideno verjetnost $P(x_2|z_1)$, torej da so tla umazana glede na prejšnjo meritev. V enačbi (6.16) zamenjamo integracijo z vsoto in dobimo

$$P(x_k|z_{1:k-1}) = \sum_{x_{k-1} \in X} P(x_k|x_{k-1}) P(x_{k-1}|z_{1:k-1})$$

Za zdaj predpostavimo, da mobilni sistem lahko le zazna stanje tal in ne more vplivati nanj, torej ne izvaja čiščenja tal in tal tudi ne more umazati. Tako je verjetnost prehajanja stanja preprosto $P(x_2|x_1) = 1$ in $P(x_2|\bar{x}_1) = 0$. Dobimo torej

$$\begin{aligned} P(x_2|z_{1:1}) &= P(x_2|x_1)P(x_1|z_1) + P(x_2|\bar{x}_1)P(\bar{x}_1|z_1) = \\ &= 1 \cdot 0{,}8421 + 0 \cdot 0{,}1579 = \\ &= 0{,}8421 \end{aligned}$$

kar je logičen rezultat. Če namreč nimamo meritve z_2, nismo pridobili novih informacij o sistemu. Zato ima stanje tal enako verjetnost v trenutku $k = 2$ kot v prejšnjem trenutku $k = 1$.

Z upoštevanjem razmerja (6.17) lahko v korekcijskem koraku združimo meritev s trenutno oceno

$$P(x_2|z_{1:2}) = \frac{P(z_2|x_2)P(x_2|z_{1:1})}{P(z_2|z_{1:1})} = \\ = \frac{0{,}8 \cdot 0{,}8421}{P(z_2|z_{1:1})}$$

kjer moramo izračunati vrednost normirnega člena, ki podaja verjetnost čistih tal v trenutku $k = 2$. Faktor lahko določimo z vsoto vseh možnih kombinacij stanj, ki vodijo do trenutne meritve z_k ob upoštevanju izidov preteklih meritev $z_{1:k-1}$

$$P(z_k|z_{1:k-1}) = \sum_{x_k \in X} P(z_k|x_k)P(x_k|z_{1:k-1})$$

V našem primeru zapišemo

$$P(z_2|z_1) = P(z_2|x_2)P(x_2|z_1) + P(z_2|\bar{x}_2)P(\bar{x}_2|z_1) = \\ = 0{,}8 \cdot 0{,}8421 + 0{,}1 \cdot 0{,}1579 = \\ = 0{,}6895$$

Verjetnost, da so tla čista, če je senzor dvakrat zapored zaznal čista tla, je

$$P(x_2|z_{1:2}) = 0{,}9771$$

Primer 6.6

Za primer 6.3 določite, kako se spremeni verjetnost stanja, če so tla čista, senzor pa opravi tri meritve $z_{1:3} = (clean,\ clean,\ dirty)$.

Rešitev

Prvi dve pogojni verjetnosti smo že izračunali v primeru 6.3 in 6.5, tretja pa je

$$P(x_3|z_{1:3}) = \frac{P(z_3|x_3)P(x_3|z_{1:2})}{P(z_3|z_{1:2})} = \\ = \frac{0{,}2 \cdot 0{,}9771}{0{,}2 \cdot 0{,}9771 + 0{,}9 \cdot (1 - 0{,}9771)} = \\ = 0{,}9046$$

kjer je $P(Z = dirty|X = clean) = 1 - P(Z = clean|X = clean)$. Verjetnosti stanj v treh zaporednih meritvah, opravljenih v časovnih trenutkih $i = 1, 2, 3$, so

$$P(x_k|z_{1:i}) = (0{,}8421,\ 0{,}9771,\ 0{,}9046)$$

Porazdelitev trenutnega stanja za trenutke $k = 1, 2, 3$ je prikazana na sliki 6.8. Implementacija rešitve v programskem okolju Matlab je prikazana v programu 6.2.

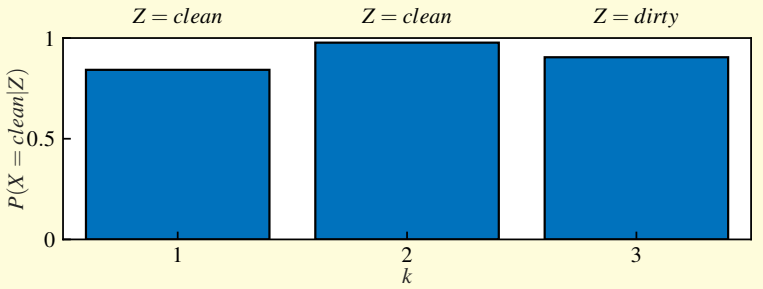
Program 6.2

./src/prb/example_nocleaning.m

```
% Verjetnost čistih (clean) in umazanih (dirty) tal
P_Xc = 0.4    % P(X=clean)
P_Xd = 1-P_Xc % P(X=dirty)
% Pogojna verjetnost merjenja čistoče
P_ZcXc = 0.8      % P(Z=clean|X=clean)
P_ZdXc = 1-P_ZcXc % P(Z=dirty|X=clean)
P_ZdXd = 0.9      % P(Z=dirty|X=dirty)
P_ZcXd = 1-P_ZdXd % P(Z=clean|X=dirty)

disp('Korak k = 1: Z=clean')
% Verjetnost meritve v primeru, da zaznamo čista tla
P_Zc_k1 = P_ZcXc*P_Xc + P_ZcXd*P_Xd
% Verjetnost čistih tal po opravljeni meritvi (Bayesovo pravilo)
P_XcZc_k1 = P_ZcXc*P_Xc/P_Zc_k1
P_XdZc_k1 = 1-P_XcZc_k1;

disp('Korak k = 2: Z=clean')
% Verjetnost meritve v primeru, da zaznamo čista tla
P_Zc_k2 = P_ZcXc*P_XcZc_k1 + P_ZcXd*P_XdZc_k1
% Verjetnost čistih tal po opravljeni meritvi (Bayesovo pravilo)
P_XcZc_k2 = P_ZcXc*P_XcZc_k1/P_Zc_k2
P_XdZc_k2 = 1-P_XcZc_k2;

disp('Korak k = 3: Z=dirty')
% Verjetnost meritve v primeru, da zaznamo umazana tla
P_Zd_k3 = P_ZdXc*P_XcZc_k2 + P_ZdXd*P_XdZc_k2
% Verjetnost čistih tal po opravljeni meritvi (Bayesovo pravilo)
P_XcZd_k3 = P_ZdXc*P_XcZc_k2/P_Zd_k3
P_XdZd_k3 = 1-P_XcZd_k3;
```

```
P_Xc =
    0.4000
P_Xd =
    0.6000
P_ZcXc =
    0.8000
P_ZdXc =
    0.2000
P_ZdXd =
    0.9000
P_ZcXd =
    0.1000
Korak k = 1: Z=clean
P_Zc_k1 =
    0.3800
P_XcZc_k1 =
    0.8421
Korak k = 2: Z=clean
P_Zc_k2 =
    0.6895
P_XcZc_k2 =
    0.9771
Korak k = 3: Z=dirty
P_Zd_k3 =
```

```
    0.2160
P_XcZd_k3 =
    0.9046
```



Slika 6.8: Porazdelitev trenutnega stanja v treh trenutkih iz primera 6.6

6.4.3 Ocenjevanje stanj iz opazovanj in akcij

V poglavju 6.4.2 smo ocenjevali stanje sistema samo na podlagi opazovanja okolice. Običajno pa akcije mobilnega sistema vplivajo na okolico, torej lahko spreminjajo tudi stanja sistema (npr. mobilni sistem se premakne, izvaja čiščenje itd.). Vsaka akcija mobilnega sistema ima neko lastno negotovost, zato izid akcije ni deterministično, ampak je podan z neko verjetnostjo. Gostota verjetnosti $p(x_k|x_{k-1}, u_{k-1})$ opisuje verjetnost prehoda iz prejšnjega v naslednje stanje pri znani akciji. Akcija u_{k-1} nastopi v trenutku $k-1$ in vpliva na sistem do trenutka k, zato jo običajno poimenujemo kar *trenutna akcija*. V splošnem akcije v okolici povečujejo stopnjo negotovosti našega znanja o okolici, meritve v okolici pa običajno zmanjšujejo stopnjo negotovosti.

Poglejmo si, kako akcije in meritve vplivajo na naše znanje o stanjih. Želimo ugotoviti gostoto verjetnosti $p(x_k|z_{1:k}, u_{0:k-1})$, kjer so z meritve in u akcije.

Kot v poglavju 6.4.2 lahko zapišemo Bayesov teorem

$$p(x_k|z_{1:k}, u_{0:k-1}) = \frac{p(z_k|x_k, z_{1:k-1}, u_{0:k-1})p(x_k|z_{1:k-1}, u_{0:k-1})}{p(z_k|z_{1:k-1}, u_{0:k-1})} \tag{6.18}$$

kjer so:

- $p(x_k|z_{1:k}, u_{0:k-1})$ je ocena porazdelitve stanja v trenutku k, posodobljena z znanimi meritvami in opravljenimi akcijami,
- $p(z_k|x_k, z_{1:k-1}, u_{0:k-1})$ je porazdelitev meritve v trenutku k, če poznamo trenutno stanje x_k, opravljene akcije in pretekle meritve do trenutka $k-1$,

- $p(x_k|z_{1:k-1}, u_{0:k-1})$ je napoved porazdelitve stanja na osnovi preteklih meritev in opravljenih akcij do trenutka k,
- $p(z_k|z_{1:k-1}, u_{0:k-1})$ je porazdelitev opravljene meritve (zaupanje v opravljeno meritev) v trenutku k.

Indeksi akcij so v razponu od 0 do $k-1$, ker akcije v preteklih trenutkih vplivajo na obnašanje stanja sistema v trenutku k.

Nadalje velja, da lahko trenutno meritev z_k v (6.18) opišemo le z znanim stanjem sistema x_k, saj pretekle meritve in akcije ne prinašajo dodatnih informacij o sistemu (stanje je vsebovano, Markovov proces)

$$p(z_k|x_k, z_{1:k-1}, u_{0:k-1}) = p(z_k|x_k)$$

Zato lahko (6.18) poenostavimo v

$$p(x_k|z_{1:k}, u_{0:k-1}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1}, u_{0:k-1})}{p(z_k|z_{1:k-1}, u_{0:k-1})} \tag{6.19}$$

Rekurzivno pravilo za posodobitev verjetnosti ocene stanja (6.19) na podlagi preteklih meritev in akcij vključuje tudi predikcijo $p(x_k|z_{1:k-1}, u_{0:k-1})$, kjer porazdelitev ocene stanja napovemo na podlagi preteklih meritev $z_{1:k-1}$ in vseh akcij $u_{0:k-1}$. Tako lahko postopek ocenjevanja stanj razdelimo na *predikcijski* in *korekcijski korak*. V predikcijskem koraku še ni znana zadnja meritev, je pa znana trenutna akcija, zato lahko na podlagi modela sistema napovemo porazdelitev stanj. Ko je na voljo nova meritev, izvedemo še korekcijski korak.

Predikcija

Predikcijo $p(x_k|z_{1:k-1}, u_{0:k-1})$ lahko ovrednotimo s teoremom popolne verjetnosti

$$p(x_k|z_{1:k-1}, u_{0:k-1}) = \int p(x_k|x_{k-1}, z_{1:k-1}, u_{0:k-1})p(x_{k-1}|z_{1:k-1}, u_{0:k-1})\,\mathrm{d}x_{k-1}$$

kjer zaradi vsebovanosti stanja velja

$$p(x_k|x_{k-1}, z_{1:k-1}, u_{0:k-1}) = p(x_k|x_{k-1}, u_{k-1})$$

nadalje ugotovimo, da najnovejša akcija u_{k-1} ni potrebna za oceno stanja v prejšnjem trenutku

$$p(x_{k-1}|z_{1:k-1}, u_{0:k-1}) = p(x_{k-1}|z_{1:k-1}, u_{0:k-2})$$

Zapišemo končni izraz za izračun predikcije

$$p(x_k|z_{1:k-1}, u_{0:k-1}) = \int p(x_k|x_{k-1}, u_{k-1})p(x_{k-1}|z_{1:k-1}, u_{0:k-2})\,\mathrm{d}x_{k-1} \tag{6.20}$$

kjer je $p(x_k|x_{k-1}, u_{k-1})$ porazdelitev prehajanja med stanji in $p(x_{k-1}|z_{1:k-1}, u_{0:k-2})$ korekcijska ocena porazdelitve stanja iz prejšnjega trenutka.

Korekcija

Ocena stanj, ko je na voljo meritev v trenutku k in predhodno izračunana predikcija, je

$$p(x_k|z_{1:k}, u_{0:k-1}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1}, u_{0:k-1})}{p(z_k|z_{1:k-1}, u_{0:k-1})} \tag{6.21}$$

Verjetnost $p(z_k|z_{1:k-1}, u_{0:k-1})$, ki predstavlja zaupanje v opravljeno meritev, pa je

$$p(z_k|z_{1:k-1}, u_{0:k-1}) = \int p(z_k|x_k)p(x_k|z_{1:k-1}, u_{0:k-1})\,\mathrm{d}x_k$$

Splošni algoritem za Bayesov filter

Splošna oblika Bayesovega filtra je podana v psevdokodi v algoritmu 3. Pogojna verjetnost korekcijskega koraka $p(x_k|z_{1:k}, u_{0:k-1})$, ki daje oceno porazdelitve stanja na podlagi znanih akcij in meritev, je znana kot **zaupanje** in jo zapišemo kot

$$bel(x_k) = p(x_k|z_{1:k}, u_{0:k-1})$$

pogojno verjetnost predikcije $p(x_k|z_{1:k-1}, u_{0:k-1})$ pa označimo kot

$$bel_p(x_k) = p(x_k|z_{1:k-1}, u_{0:k-1})$$

Bayesov filter ocenjuje porazdelitev ocene stanja. V trenutku, ko je znana informacija o trenutni akciji u_{k-1}, lahko izvedemo predikcijski korak in ko je na voljo tudi nova meritev, lahko izvedemo korekcijski korak. Vpeljemo še normirni faktor $\eta = \frac{1}{\alpha} = \frac{1}{p(z_k|z_{1:k-1}, u_{0:k-1})}$. Algoritem za Bayesov filter (algoritem 3) najprej izvede predikcijo, nato pa še korekcijo, ki je ustrezno normirana.

Algorithm 3 Bayesov filter

function BAYESOV_FILTER($bel(x_{k-1})$, u_{k-1}, z_k)
  $\alpha \leftarrow 0$
  for all x_k **do**
    $bel_p(x_k) \leftarrow \int p(x_k|x_{k-1}, u_{k-1})bel(x_{k-1})\,\mathrm{d}x_{k-1}$
    $bel'(x_k) \leftarrow p(z_k|x_k)bel_p(x_k)$
    $\alpha \leftarrow \alpha + bel'(x_k)$
  end for
  for all x_k **do**
    $bel(x_k) \leftarrow \frac{1}{\alpha}bel'(x_k)$
  end for
  return $bel(x_k)$
end function

Kot je razvidno iz algoritma 3, moramo rešiti integral, da določimo porazdelitev v predikcijskem koraku. Izvedba algoritma je torej omejena na enostavne zvezne

primere, kjer je možna eksplicitna rešitev integrala, in diskretne primere s končnim številom stanj, kjer lahko integral zamenja vsota.

Primer 6.7

Mobilni robot je opremljen s senzorjem, ki lahko zazna, ali so tla čista ali ne ($Z \in \{clean,\ dirty\}$), ter čistilnim sistemom (komplet krtač, vakuumska črpalka in posoda za prah) za čiščenje tal. Čiščenje se izvaja samo v primeru, ko robot meni, da je tla potrebno očistiti ($U \in \{clean,\ null\}$). Zanima nas, ali so tla čista ali ne ($X \in \{clean,\ dirty\}$).

Začetna verjetnost (zaupanje), da so tla čista, je

$$bel(X_0 = clean) = 0{,}5$$

Veljavnost meritev senzorja je podana s statističnim modelom senzorja

$$P(Z_k = clean|X_k = clean) = 0{,}8 \qquad P(Z_k = dirty|X_k = clean) = 0{,}2$$
$$P(Z_k = dirty|X_k = dirty) = 0{,}9 \qquad P(Z_k = clean|X_k = dirty) = 0{,}1$$

Verjetnosti izida, če se robot odloči za čiščenje tal, je

$$P(X_k = clean|X_{k-1} = clean, U_{k-1} = clean) = 1$$
$$P(X_k = dirty|X_{k-1} = clean, U_{k-1} = clean) = 0$$
$$P(X_k = clean|X_{k-1} = dirty, U_{k-1} = clean) = 0{,}8$$
$$P(X_k = dirty|X_{k-1} = dirty, U_{k-1} = clean) = 0{,}2$$

V kolikor pa čistilni sistem ni aktiviran, lahko domnevamo naslednje verjetnosti izida

$$P(X_k = clean|X_{k-1} = clean, U_{k-1} = null) = 1$$
$$P(X_k = dirty|X_{k-1} = clean, U_{k-1} = null) = 0$$
$$P(X_k = clean|X_{k-1} = dirty, U_{k-1} = null) = 0$$
$$P(X_k = dirty|X_{k-1} = dirty, U_{k-1} = null) = 1$$

Predpostavimo, da mobilni sistem najprej opravi akcijo in šele nato prejme meritev. Določite zaupanje $bel_p(x_k)$ na osnovi opravljene akcije (predikcija) in zaupanje $bel(x_k)$ na osnovi meritve $bel(x_k)$ (korekcija) za naslednje sekvence opravljenih akcij in zaznanih meritev

k	U_{k-1}	Z_k
1	*null*	*dirty*
2	*clean*	*clean*
3	*clean*	*clean*

Rešitev

Za boljšo preglednost bomo označevali $X_k \in \{clean,\ dirty\}$ kot $X_k \in \{x_k,\ \bar{x}_k\}$, $Z_k \in \{clean,\ dirty\}$ kot $Z_k \in \{z_k,\ \bar{z}_k\}$ in $U_k \in (clean,\ null)$ kot $U_k \in \{u_k,\ \bar{u}_k\}$.

Uporabimo algoritem 3. Za trenutek $k = 1$, ko je opravljena akcija $\bar{u}_0 = null$, lahko določimo predikcijo zaupanja, da so tla čista

$$\begin{aligned} bel_p(x_1) &= \sum_{x_0 \in X} P(x_1|x_0, \bar{u}_0) bel(x_0) = \\ &= P(x_1|\bar{x}_0, \bar{u}_0) bel(\bar{x}_0) + P(x_1|x_0, \bar{u}_0) bel(x_0) = \\ &= 0 \cdot 0{,}5 + 1 \cdot 0{,}5 = \\ &= 0{,}5 \end{aligned}$$

in predikcijo zaupanja, da so tla umazana

$$\begin{aligned} bel_p(\bar{x}_1) &= \sum_{x_0 \in X} P(\bar{x}_1|x_0, \bar{u}_0) bel(x_0) = \\ &= P(\bar{x}_1|\bar{x}_0, \bar{u}_0) bel(\bar{x}_0) + P(\bar{x}_1|x_0, \bar{u}_0) bel(x_0) = \\ &= 1 \cdot 0{,}5 + 0 \cdot 0{,}5 = \\ &= 0{,}5 \end{aligned}$$

Ker nismo izvedli nobene akcije, so verjetnosti stanja nespremenjene. Glede na meritev $\bar{z}_1 = dirty$ lahko določimo korekcijo zaupanja

$$bel(x_1) = \eta p(\bar{z}_1|x_1) bel_p(x_1) = \eta 0{,}2 \cdot 0{,}5 = \eta 0{,}1$$

in

$$bel(\bar{x}_1) = \eta p(\bar{z}_1|\bar{x}_1) bel_p(\bar{x}_1) = \eta 0{,}9 \cdot 0{,}5 = \eta 0{,}45$$

Ocenimo še normirni faktor η

$$\eta = \frac{1}{0{,}1 + 0{,}45} = 1{,}82$$

in končne vrednosti zaupanj

$$bel(x_1) = 0{,}182 \qquad bel(\bar{x_1}) = 0{,}818$$

Postopek ponovimo še za trenutek $k = 2$, kjer je $u_1 = clean$ in $z_2 = clean$

$$\begin{aligned} bel_p(x_2) &= 0{,}8364 & bel_p(\bar{x}_2) &= 0{,}1636 \\ bel(x_2) &= 0{,}9761 & bel(\bar{x}_2) &= 0{,}0239 \end{aligned}$$

in trenutek $k = 3$, kjer je $u_2 = clean$ in $z_3 = clean$

$$\begin{aligned} bel_p(x_3) &= 0{,}9952 & bel_p(\bar{x}_3) &= 0{,}0048 \\ bel(x_3) &= 0{,}9994 & bel(\bar{x}_3) &= 0{,}0006 \end{aligned}$$

Rešitev primera 6.7 v programskem okolju Matlab je podana v programu 6.3.

Program 6.3

./src/prb/example_cleaning.m

```
% Notacija: X === X(k), X' === X(k-1)
disp('Začetno zaupanje v čista in umazana tla')
bel_Xc = 0.5; % bel(X=clean)
bel_X = [bel_Xc 1-bel_Xc] % bel(X=clean), bel(X=dirty)

disp('Pogojne verjetnosti meritev senzorja')
P_ZcXc = 0.8;      % P(Z=clean|X=clean)
P_ZdXc = 1-P_ZcXc; % P(Z=dirty|X=clean)
P_ZdXd = 0.9;      % P(Z=dirty|X=dirty)
P_ZcXd = 1-P_ZdXd; % P(Z=clean|X=dirty)
p_ZX = [P_ZcXc, P_ZcXd; ...
        P_ZdXc, P_ZdXd]

disp('Pogojne verjetnosti v primeru čiščenja')
P_XcXcUc = 1;          % P(X=clean|X'=clean,U'=clean)
P_XdXcUc = 1-P_XcXcUc; % P(X=dirty|X'=clean,U'=clean)
P_XcXdUc = 0.8;        % P(X=clean|X'=dirty,U'=clean)
P_XdXdUc = 1-P_XcXdUc; % P(X=dirty|X'=dirty,U'=clean)
p_ZXUc = [P_XcXcUc, P_XdXcUc; ...
          P_XcXdUc, P_XdXdUc]

disp('Pogojne verjetnosti, če ni nobene akcije')
P_XcXcUn = 1;          % P(X=clean|X'=clean,U'=null)
P_XdXcUn = 1-P_XcXcUn; % P(X=dirty|X'=clean,U'=null)
P_XcXdUn = 0;          % P(X=clean|X'=dirty,U'=null)
P_XdXdUn = 1-P_XcXdUn; % P(X=dirty|X'=dirty,U'=null)
p_ZXUn = [P_XcXcUn, P_XdXcUn; ...
          P_XcXdUn, P_XdXdUn]

U = {'null',  'clean', 'clean'};
Z = {'dirty', 'clean', 'clean'};
for k=1:length(U)
    fprintf('Predikcijski korak: U(%d)=%s\n', k-1, U{k})
    if strcmp(U(k), 'clean')
        belp_X = bel_X*p_ZXUc
    else
        belp_X = bel_X*p_ZXUn
    end

    fprintf('Korekcijski korak: Z(%d)=%s\n', k, Z{k})
    if strcmp(Z(k), 'clean')
        bel_X = p_ZX(1,:).*belp_X;
    else
        bel_X = p_ZX(2,:).*belp_X;
    end
    bel_X = bel_X/sum(bel_X)
end
```

```
Začetno zaupanje v čista in umazana tla
bel_X =
    0.5000    0.5000
Pogojne verjetnosti meritev senzorja
p_ZX =
    0.8000    0.1000
    0.2000    0.9000
Pogojne verjetnosti v primeru čiščenja
p_ZXUc =
    1.0000         0
    0.8000    0.2000
Pogojne verjetnosti, če ni nobene akcije
```

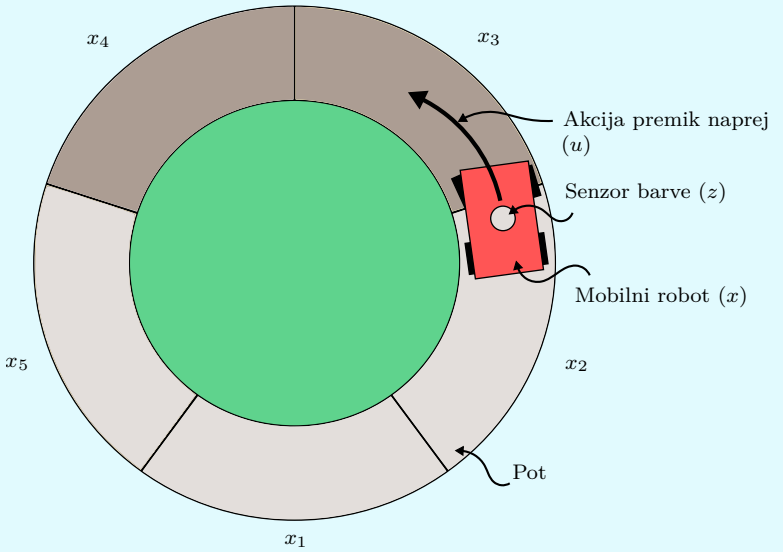
```
p_ZXUn =
     1     0
     0     1
Predikcijski korak: U(0)=null
belp_X =
    0.5000    0.5000
Korekcijski korak: Z(1)=dirty
bel_X =
    0.1818    0.8182
Predikcijski korak: U(1)=clean
belp_X =
    0.8364    0.1636
Korekcijski korak: Z(2)=clean
bel_X =
    0.9761    0.0239
Predikcijski korak: U(2)=clean
belp_X =
    0.9952    0.0048
Korekcijski korak: Z(3)=clean
bel_X =
    0.9994    0.0006
```

6.4.4 Primer lokalizacije

Na preprostem primeru si poglejmo princip lokalizacije, ki predstavlja osnovno idejo algoritma za lokalizacijo *Monte Carlo*. Mobilni robot se premika v okolju in zaznava svojo lego s senzorjem. Algoritem za lokalizacijo mora na podlagi meritev senzorja in izvedenih premikov določiti lego robota v okolju. Pri tem uporabimo teorem popolne verjetnosti in Bayesovo pravilo, ki predstavljata osnovo za izvedbo Bayesovega filtra. V nadaljevanju je predstavljen proces zaznavanja ob prisotnosti negotovosti senzorja, ki mu sledi proces izvajanja akcije ob prisotnosti negotovosti aktuatorja. Mobilni robot izvaja akcije z namenom spreminjanja stanja v okolju (lega mobilnega sistema se spreminja z njegovim premikanjem).

Primer 6.8

Mobilni robot se vozi po krožni poti v smeri naprej ali nazaj. Pot je sestavljena iz končnega števila svetlih in temnih ploščic v naključnem vrstnem redu, kjer širina ploščice ustreza širini poti. Mobilni sistem lahko namestimo na katerokoli oštevilčeno ploščico. Brez izgube splošnosti predpostavimo, da je pot sestavljena iz petih ploščic, kot je prikazano na sliki 6.9.



Slika 6.9: Pot v okolju je sestavljena iz petih črnih in belih ploščic. Mobilni sistem se lahko premika med njimi in zaznava barvo trenutne ploščice.

Vsaka ploščica predstavlja celico, v katero lahko postavimo mobilni sistem, torej imamo diskretno predstavitev okolja. Mobilni sistem pozna zemljevid okolice, torej pozna zaporedje svetlih in temnih ploščic, vendar ne ve, v kateri celici se nahaja. Začetno zaupanje v pozicijo mobilnega sistema je podano z enakomerno porazdelitvijo, saj je vsaka celica enako verjetna. Mobilni sistem ima senzor za zaznavanje svetlih in temnih ploščic, vendar so njegove meritve negotove. Mobilni sistem se lahko premika za želeno število ploščic v smeri naprej ali nazaj, vendar je gibanje samo po sebi negotovo (lahko se premakne premalo ali preveč). Na omenjenem primeru bosta razložena procesa *zaznavanja okolja* in *gibanja v okolju*.

6.4.5 Zaznavanje okolja

S pomočjo meritve, izvedene v okolici, lahko izboljšamo oceno stanja X (npr. lokacija) v tem okolju. Predstavljajmo si, da sredi noči hodimo v spanju in tavamo po hiši. Ko se zbudimo, lahko s svojimi čuti (vid, dotik itd.) ugotovimo, kje se nahajamo.

Začetno znanje o okolici lahko matematično opišemo s porazdelitvijo $p(x)$. To porazdelitev lahko izboljšamo ob nastopu nove meritve z (z zaupanjem $p(z|x)$), če je verjetnost po opravljeni meritvi $p(x|z)$. To lahko dosežemo z Bayesovim

pravilom $p(x|z) = bel(x) = \frac{p(z|x)p(x)}{p(z)}$. Verjetnost $p(z|x)$ predstavlja statistični model senzorja, $bel(x)$ pa je zaupanje v oceno stanja po opravljeni meritvi. V procesu zaznavanja se torej izvede korekcijski korak Bayesovega filtra.

Primer 6.9

Za primer 6.8 predpostavimo, da mobilni sistem zazna temno celico $Z = dark$. Mobilni sistem pravilno zazna temno celico z verjetnostjo 0,6, napako pa naredi z verjetnostjo 0,2, kjer svetlo celico razpozna kot temno. Tako zapišemo

$$p(Z = dark|X = x_d) = 0{,}6 \quad d \in \{3,\ 4\}$$
$$p(Z = dark|X = x_b) = 0{,}2 \quad b \in \{1,\ 2,\ 5\}$$

kjer indeks b označuje svetle celice, indeks d pa temne. Na začetku mobilni sistem ne pozna svoje pozicije, kar opišemo z enakomerno porazdelitvijo $P(X = x_i) = bel(x_i) = 0{,}2$, $i \in \{1, \ldots, 5\}$. Kakšna je porazdelitev lokacije po opravljeni meritvi?

Rešitev

Želimo določiti porazdelitev pogojne verjetnosti $\boldsymbol{p}(X_1|Z = dark)$, kar je zaupanje v oceno stanja po opravljeni meritvi. Želeno verjetnost lahko določimo s pomočjo korekcijskega koraka Bayesovega filtra

$$\begin{aligned}\boldsymbol{p}(X_1|Z = dark) &= \frac{\boldsymbol{p}(Z = dark|X_1) \star \boldsymbol{p}(X_1)}{P(Z = dark)} = \\ &= \frac{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2]^T \star [0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T}{P(Z = dark)} = \\ &= \frac{[0{,}04,\ 0{,}04,\ 0{,}12,\ 0{,}12,\ 0{,}04]^T}{P(Z = dark)}\end{aligned}$$

kjer operator $\star$ predstavlja množenje istoležnih elementov vektorja.

Izračunati moramo verjetnost zaznave temne celice $P(Z = dark)$. Zato je potrebno oceniti polno verjetnost, tj. verjetnost zaznave temne celice ob upoštevanju vseh celic

$$\begin{aligned}P(Z = dark) &= \sum_i P(Z = dark|X_1 = x_i)P(X_1 = x_i) = \\ &= \boldsymbol{p}^T(Z = dark|X_1)\boldsymbol{p}(X_1) = \\ &= [0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = \\ &= 0{,}36\end{aligned}$$

Iskana (posteriorna) porazdelitev je torej

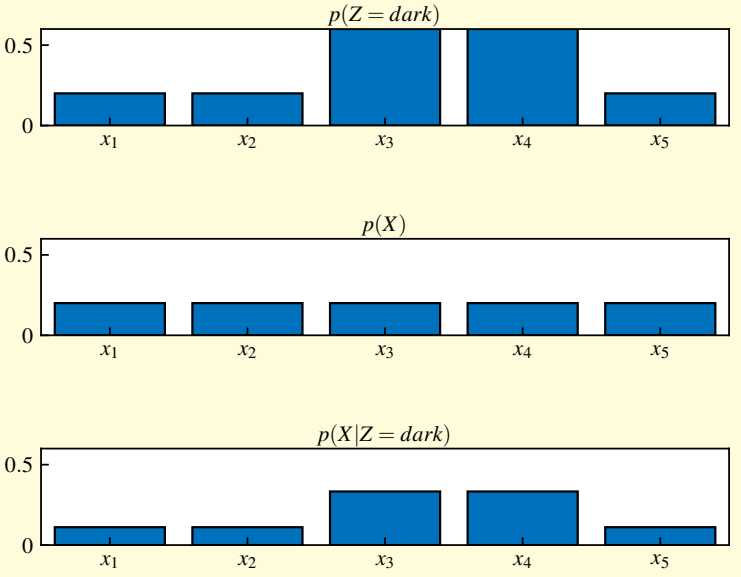
$$\boldsymbol{p}(X_1|Z = dark) = [0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T$$

od koder lahko sklepamo, da se mobilni robot s trikrat večjo verjetnostjo nahaja v celici 3 ali 4, kot pa v preostalih treh. Porazdelitve so grafično prikazane na sliki 6.10, rešitev tega primera pa je podana v programu 6.4.

Program 6.4
./src/prb/example_sensing.m

```
disp('Porazdelitev meritev senzorja p(Z=dark|X)')
p_ZdX = [0.2 0.2 0.6 0.6 0.2]
disp('Porazdelitev meritev senzorja p(Z=bright|X)')
p_ZbX = 1-p_ZdX
disp('Začetna porazdelitev p(X)')
p_X = ones(1,5)/5

disp('Verjetnost detekcije temne celice P(Z=dark)')
P_Zd = p_ZdX*p_X.'

disp('Porazdelitev p(X|Z=dark)')
p_XZd = p_ZdX.*p_X/P_Zd
```

```
Porazdelitev meritev senzorja p(Z=dark|X)
p_ZdX =
    0.2000    0.2000    0.6000    0.6000    0.2000
Porazdelitev meritev senzorja p(Z=bright|X)
p_ZbX =
    0.8000    0.8000    0.4000    0.4000    0.8000
Začetna porazdelitev p(X)
p_X =
    0.2000    0.2000    0.2000    0.2000    0.2000
Verjetnost detekcije temne celice P(Z=dark)
P_Zd =
    0.3600
Porazdelitev p(X|Z=dark)
p_XZd =
    0.1111    0.1111    0.3333    0.3333    0.1111
```



Slika 6.10: Porazdelitve iz primera 6.9

Primer 6.10

Za primer 6.9 odgovorite na naslednja vprašanja:

1. Ali lahko večkratne zaporedne meritve izboljšajo oceno pozicije mobilnega robota (robot med meritvami miruje)?
2. Kakšna je porazdelitev pozicije mobilnega robota, če robot dvakrat zapored zazna ploščico kot temno?
3. Kakšna je porazdelitev pozicije mobilnega robota, če robot najprej zazna ploščico kot temno nato pa kot svetlo?
4. Kakšna je porazdelitev pozicije mobilnega robota, če robot najprej zazna ploščico kot temno, nato kot svetlo, nato pa spet kot temno?

Rešitev

1. Večkratne zaporedne meritve lahko izboljšajo oceno pozicije mobilnega robota, če je verjetnost pravilne meritve večja od verjetnosti napake pri meritvi.

2. Če senzor zazna celico kot temno dvakrat zapored, je porazdelitev

$$\begin{aligned}\boldsymbol{p}(X_2|Z_1 = dark, Z_2 = dark) &= p(X_2|z_1, z_2) = \\ &= \frac{\boldsymbol{p}(z_2|X_2) \star \boldsymbol{p}(X_2|z_1)}{P(z_2|z_1)} = \\ &= \frac{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2]^T \star [0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T}{P(z_2|z_1)}\end{aligned}$$

kjer je j-ti element v porazdelitvi $\boldsymbol{p}(X_2|z_1)$ podan s $P(X_2 = x_j|z_1) = \boldsymbol{p}^T(X_2 = x_j|X_1)\boldsymbol{p}(X_1|z_1) = P(X_1 = x_j|z_1)$, ker nimamo vpliva na stanja (glejte primer 6.5), ampak jih le merimo (opazujemo). Pogojna verjetnost v imenovalcu je

$$\begin{aligned}P(z_2|z_1) &= \sum_{x_i} P(z_2|X_2 = x_i)P(X_2 = x_i|z_1)) = \\ &= \boldsymbol{p}^T(z_2|X_2)\boldsymbol{p}(X_2|z_1)) = \\ &= [0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2][0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T = \\ &= 0{,}4667\end{aligned}$$

Končna rešitev je

$$\begin{aligned}\boldsymbol{p}(X_2|z_1, z_2) &= \frac{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2]^T \star [0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T}{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2][0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T} = \\ &= [0{,}0476,\ 0{,}0476,\ 0{,}4286,\ 0{,}4286,\ 0{,}0476]^T\end{aligned}$$

3. Svetla celica je pravilno zaznana z verjetnostjo $p(Z = bright|X = bright) = 1 - p(Z = dark|X = bright) = 0{,}8$ in nepravilno z verjetnostjo $p(Z = bright|X = dark) = 1 - p(Z = dark|X = dark) = 0{,}4$. Drugo meritev lahko izvedemo na osnovi porazdelitve $\boldsymbol{p}(X_2|Z_1 = dark)$

$$\begin{aligned}\boldsymbol{p}(X_2|Z_1 = dark, Z_2 = bright) &= \boldsymbol{p}(X_2|z_1, z_2) = \\ &= \frac{[0{,}8,\ 0{,}8,\ 0{,}4,\ 0{,}4,\ 0{,}8]^T \star [0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T}{P(Z_2 = bright|Z_1 = dark)}\end{aligned}$$

$$\begin{aligned}&P(Z_2 = bright|Z_1 = dark) = \\ &= \sum_{x_i} P(Z_2 = bright|X_2 = x_i)P(X_2 = x_i|Z_1 = dark) = \\ &= \boldsymbol{p}^T(Z_2 = bright|X_2)\boldsymbol{p}(X_2|Z_1 = dark) = \\ &= [0{,}8,\ 0{,}8,\ 0{,}4,\ 0{,}4,\ 0{,}8][0{,}11,\ 0{,}11,\ 0{,}33,\ 0{,}33,\ 0{,}11]^T = 0{,}533\end{aligned}$$

$$\boldsymbol{p}(X_2|Z_1 = dark, Z_2 = bright) = [0{,}167,\ 0{,}167,\ 0{,}25,\ 0{,}25,\ 0{,}167]^T$$

4. Porazdelitev stanja po treh meritvah je

$$\begin{aligned}\boldsymbol{p}(X_3|Z_1 = dark, Z_2 = bright, Z_3 = dark) = \\ = [0{,}083,\ 0{,}083,\ 0{,}375,\ 0{,}375,\ 0{,}083]^T\end{aligned}$$

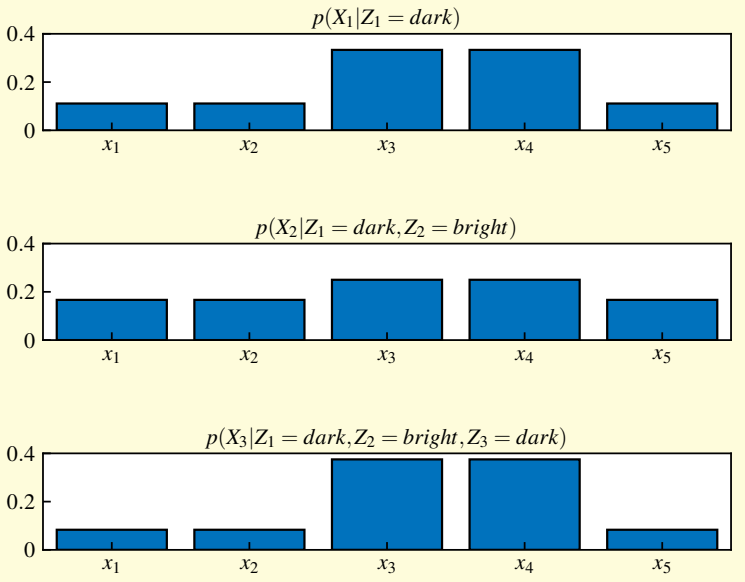
V programu 6.5 je prikazana Matlab koda rešitve. Porazdelitev trenutnega stanja za tri časovne trenutke je grafično predstavljena na sliki 6.11.

Program 6.5

./src/prb/example_sensing_ans34.m

```
p_ZdX = [0.2 0.2 0.6 0.6 0.2];
p_ZbX = 1-p_ZdX;
p_X = ones(1,5)/5;

disp('Verjetnost detekcije temne celice P(Z1=dark)')
P_z1 = p_ZdX*p_X.'
disp('Porazdelitev p(X1|Z1=dark)')
p_Xz1 = p_ZdX.*p_X/P_z1

disp('Verjetnost detekcije svetle celice P(Z2=bright|Z1=dark)')
P_z2 = p_ZbX*p_Xz1.'
disp('Porazdelitev p(X2|Z1=dark,Z2=bright)')
p_Xz2 = p_ZbX.*p_Xz1/P_z2

disp('Verjetnost detekcije temne celice P(Z3=dark|Z1=dark,Z2=bright)')
P_z3 = p_ZdX*p_Xz2.'
disp('Porazdelitev p(X3|Z1=dark,Z2=bright,Z3=dark)')
p_Xz3 = p_ZdX.*p_Xz2/P_z3
```

```
Verjetnost detekcije temne celice P(Z1=dark)
P_z1 =
    0.3600
Porazdelitev p(X1|Z1=dark)
p_Xz1 =
    0.1111    0.1111    0.3333    0.3333    0.1111
Verjetnost detekcije svetle celice P(Z2=bright|Z1=dark)
P_z2 =
    0.5333
Porazdelitev p(X2|Z1=dark,Z2=bright)
p_Xz2 =
    0.1667    0.1667    0.2500    0.2500    0.1667
Verjetnost detekcije temne celice P(Z3=dark|Z1=dark,Z2=bright)
P_z3 =
    0.4000
Porazdelitev p(X3|Z1=dark,Z2=bright,Z3=dark)
p_Xz3 =
    0.0833    0.0833    0.3750    0.3750    0.0833
```



Slika 6.11: Porazdelitve trenutnega stanja v treh trenutkih iz primera 6.10

6.4.6 Gibanje v okolju

Gibanje mobilnih sistemov v okolju je izvedeno s pomočjo aktuatorjev (npr. motorna kolesa) in regulacijskega sistema (algoritem). Pri vsakem gibanju je prisotna manjša ali večja negotovost, zato gibanje mobilnega sistema povečuje negotovost stanja mobilnega sistema (lege) v okolju.

Predstavljajmo si, da stojimo v znanem okolju. Z zaprtimi očmi naredimo nekaj korakov. Približno vemo, kako velike korake smo naredili in tudi v katero smer, zato si lahko predstavljamo, kje v okolici se nahajamo. Vendar pa dolžine naših korakov niso natančno znane, prav tako težko ocenimo smeri korakov, zato se naše znanje o legi v prostoru sčasoma zmanjšuje, saj naredimo vedno več korakov.

V primeru gibanja brez zaznavanja stanj preko meritev lahko enačbo (6.20) preuredimo

$$p(x_k|u_{0:k-1}) = \int_{-\infty}^{+\infty} p(x_k|x_{k-1}, u_{k-1}) p(x_{k-1}|u_{1:k-2}) \, \mathrm{d}x_{k-1}$$

Zaupanje v novo stanje $p(x_k|u_{0:k-1})$ je odvisno od zaupanja v prejšnjem trenutku $p(x_{k-1}|u_{0:k-2})$ in pogojne verjetnosti prehoda med stanji $p(x_k|x_{k-1}, u_{k-1})$. Porazdelitev $p(x_k|, u_{0:k-1})$ določimo z integracijo (ali seštevanjem v diskretnem

primeru) vseh možnih verjetnosti prehodov $p(x_k|x_{k-1}, u_{k-1})$ iz predhodnih stanj x_{k-1} v stanje x_k pri poznani akciji u_{k-1}.

Primer 6.11

Za primer 6.8 predpostavimo, da je začetna pozicija mobilnega robota v prvi celici ($X_0 = x_1$), torej lahko začetno stanje podamo s porazdelitvijo $p(X_0) = [1,\ 0,\ 0,\ 0,\ 0]$. Mobilni sistem se lahko premika med celicami, kjer je izid akcije premika pravilen v 80 %. V 10 % se robot premakne za eno celico premalo in v 10 % za eno preveč, kot je potrebno. To lahko opišemo z naslednjimi verjetnostmi prehoda med stanji

$$\begin{aligned} P(X_k = x_i|X_{k-1} = x_j, U_{k-1} = u) &= 0{,}8 \qquad \text{za } i = j + u \\ P(X_k = x_i|X_{k-1} = x_j, U_{k-1} = u) &= 0{,}1 \qquad \text{za } i = j + u - 1 \\ P(X_k = x_i|X_{k-1} = x_j, U_{k-1} = u) &= 0{,}1 \qquad \text{za } i = j + u + 1 \end{aligned}$$

Mobilni robot se mora premakniti za dve celici v nasprotni smeri urinega kazalca ($U_0 = 2$). Določite zaupanje v pozicijo mobilnega sistema po premiku.

Rešitev

Porazdelitev (zaupanje) po premiku lahko določimo tako, da izračunamo verjetnosti pozicije mobilnega robota v vsaki celici (popolna verjetnost). Mobilni sistem lahko prispe v prvo celico samo iz celice 3 (premik preveč), celice 4 (pravi premik) in celice 5 (premik premalo). Tako dobimo porazdelitev prehoda v prvo celico $\boldsymbol{p}(X_1 = x_1|X_0, U_0 = 2) = [0,\ 0,\ 0{,}1,\ 0{,}8,\ 0{,}1]^T$. Po premiku se mobilni sistem nahaja v prvi celici z verjetnostjo

$$\begin{aligned} P(X_1 = x_1|U_0 = 2) &= \sum_{x_i} P(X_1 = x_1|X_0 = x_i, U_0 = 2)P(X_0 = x_i) = \\ &= \boldsymbol{p}^T(X_1 = x_1|X_0, U_0 = 2)\boldsymbol{p}(X_0) = \\ &= [0,\ 0,\ 0{,}1,\ 0{,}8,\ 0{,}1][1,\ 0,\ 0,\ 0,\ 0]^T = 0 \end{aligned}$$

Verjetnost, da se mobilni sistem po premiku nahaja v drugi celici, je

$$\begin{aligned} P(X_1 = x_2|U_0 = 2) &= \sum_{x_i} P(X_1 = x_2|X_0 = x_i, U_0 = 2)P(X_0 = x_i) = \\ &= \boldsymbol{p}^T(X_1 = x_2|X_0, U_0 = 2)\boldsymbol{p}(X_0) = \\ &= [0{,}1,\ 0,\ 0,\ 0{,}1,\ 0{,}8][1,\ 0,\ 0,\ 0,\ 0]^T = 0{,}1 \end{aligned}$$

Podobno lahko izračunamo še verjetnosti za ostale celice

$$\begin{aligned} P(X_1 = x_3|U_0 = 2) &= [0{,}8,\ 0{,}1,\ 0,\ 0,\ 0{,}1][1,\ 0,\ 0,\ 0,\ 0]^T = 0{,}8 \\ P(X_1 = x_4|U_0 = 2) &= [0{,}1,\ 0{,}8,\ 0{,}1,\ 0,\ 0][1,\ 0,\ 0,\ 0,\ 0]^T = 0{,}1 \\ P(X_1 = x_5|U_0 = 2) &= [0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0][1,\ 0,\ 0,\ 0,\ 0]^T = 0 \end{aligned}$$

Zato je zaupanje v pozicijo po premiku

$$\boldsymbol{p}(X_1|U_0 = 2) = [0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T$$

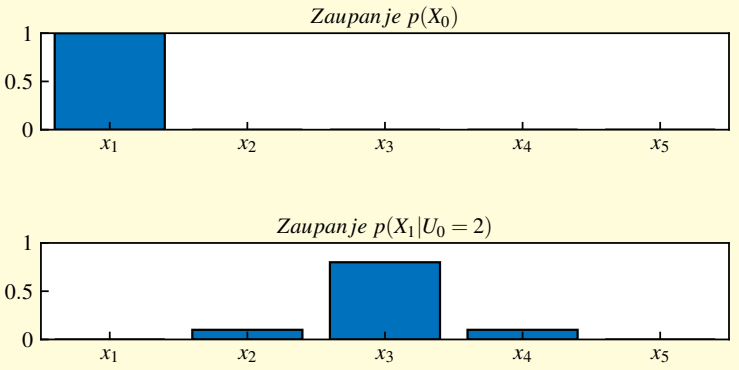
Porazdelitev trenutnega (a posteriori) stanja je grafično predstavljena na sliki 6.12. V programu 6.6 je prikazana Matlab koda rešitve.

Program 6.6

./src/prb/example_motion.m

```
disp('Začetno zaupanje p(X0)');
p_X0 = [1 0 0 0 0]

P_xxu_null = 0.8; % P(X=i|X'=j,U'=u), i=j+u
P_xxu_less = 0.1; % P(X=i|X'=j,U'=u), i=j+u-1
P_xxu_more = 0.1; % P(X=i|X'=j,U'=u), i=j+u+1

disp('Zaupanje p(X1|U0=2)');
p_xXu = [0 0 P_xxu_more P_xxu_null P_xxu_less]; % Za U=2
p_Xu = zeros(1,5);
for i=1:5
    p_Xu(i) = p_xXu*p_X0.';
    p_xXu = p_xXu([end 1:end-1]);
end
p_X1 = p_Xu
```

```
Začetno zaupanje p(X0)
p_X0 =
     1     0     0     0     0
Zaupanje p(X1|U0=2)
p_X1 =
         0    0.1000    0.8000    0.1000         0
```



Slika 6.12: Porazdelitvi trenutnega stanja v dveh trenutkih iz primera 6.11

Primer 6.12

Kakšno je zaupanje v pozicijo mobilnega robota, če robot po premiku iz primera 6.11 izvede premik za eno celico v nasprotni smeri urinega kazalca ($U_1 = 1$)?

Rešitev

Porazdelitev (zaupanje) po premiku ponovno določimo tako, da za vsako celico izračunamo verjetnost pozicije mobilnega robota v vsaki celici (popolna verjetnost). V primeru premika za eno celico je prva celica dosegljiva iz celice 1 (premik premalo), celice 4 (premik preveč) in celice 5 (pravi premik). Mobilni sistem lahko prispe v drugo celico iz celic 1, 2 ter 5 in tako naprej. Po opravljenem premiku lahko izračunamo naslednje verjetnosti

$$P(X_2 = x_1|U_0 = 2, U_1 = 1) = [0{,}1,\ 0,\ 0,\ 0{,}1,\ 0{,}8][0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T = 0{,}01$$
$$P(X_2 = x_2|U_0 = 2, U_1 = 1) = [0{,}8,\ 0{,}1,\ 0,\ 0,\ 0{,}1][0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T = 0{,}01$$
$$P(X_2 = x_3|U_0 = 2, U_1 = 1) = [0{,}1,\ 0{,}8,\ 0{,}1,\ 0,\ 0][0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T = 0{,}16$$
$$P(X_2 = x_4|U_0 = 2, U_1 = 1) = [0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0][0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T = 0{,}66$$
$$P(X_2 = x_5|U_0 = 2, U_1 = 1) = [0,\ 0,\ 0{,}1,\ 0{,}8,\ 0{,}1][0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0]^T = 0{,}16$$

Torej je zaupanje v pozicijo po drugem premiku

$$\boldsymbol{p}(X_2|U_0 = 2, U_1 = 1) = [0{,}01,\ 0{,}01,\ 0{,}16,\ 0{,}66,\ 0{,}16]^T$$

kar je prikazano na sliki 6.13. Opazimo, da je mobilni sistem najverjetneje v celici 4. Vendar pa porazdelitev nima tako izrazitega maksimuma, kot pred izvedbo drugega premika (primerjajte srednjo s spodnjo porazdelitvijo na sliki 6.13). To je v skladu z izjavo, da vsak premik poveča negotovost stanj v okolici.

Implementacija rešitve v programskem okolju Matlab je prikazana v programu 6.7.

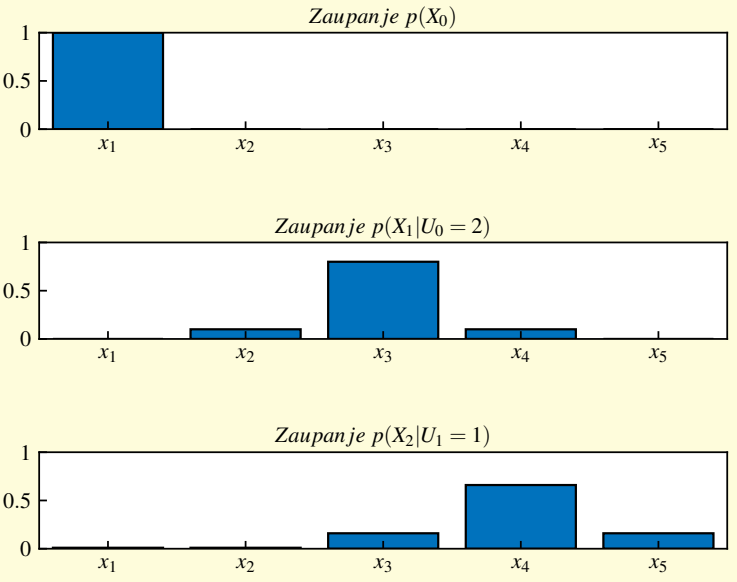
Program 6.7

./src/prb/example_motion2.m

```
disp('Začetno zaupanje p(X0)')
p_X0 = [1 0 0 0 0]

P_xxu_null = 0.8; % P(X=i|X'=j,U'=u), i=j+u
P_xxu_less = 0.1; % P(X=i|X'=j,U'=u), i=j+u-1
P_xxu_more = 0.1; % P(X=i|X'=j,U'=u), i=j+u+1

disp('Zaupanje p(X1|U0=2)');
p_xXu = [0 0 P_xxu_more P_xxu_null P_xxu_less]; % Za U=2
p_Xu = zeros(1,5);
for i=1:5
    p_Xu(i) = p_xXu*p_X0.';
    p_xXu = p_xXu([end 1:end-1]);
end
```

```
p_X1 = p_Xu

disp('Zaupanje p(X2|U1=1)');
p_xXu = [P_xxu_less 0 0 P_xxu_more P_xxu_null]; % Za U=1
p_Xu = zeros(1,5);
for i=1:5
    p_Xu(i) = p_xXu*p_X1.';
    p_xXu = p_xXu([end 1:end-1]);
end
p_X2 = p_Xu
```

```
Začetno zaupanje p(X0)
p_X0 =
     1     0     0     0     0
Zaupanje p(X1|U0=2)
p_X1 =
         0    0.1000    0.8000    0.1000         0
Zaupanje p(X2|U1=1)
p_X2 =
    0.0100    0.0100    0.1600    0.6600    0.1600
```



Slika 6.13: Porazdelitve stanj v treh časovnih trenutkih iz primera 6.12

Primer 6.13

Mobilni robot iz primera 6.11 se na začetku nahaja v prvi celici $p(X_0) = [1, 0, 0, 0, 0]$. Nato se v vsakem časovnem trenutku premakne za eno celico v nasprotni smeri urinega kazalca.

1. Kakšno je zaupanje v pozicijo mobilnega sistema po desetem premiku?
2. H kateri vrednosti konvergira zaupanje po neskončnem številu premikov?

Rešitev

1. Zaupanje v stanje po desetih premikih je

$$\boldsymbol{p}(X_{10}|U_{0:9}) = [0{,}29,\ 0{,}22,\ 0{,}13,\ 0{,}13,\ 0{,}22]^T$$

2. Po neskončnem številu premikov dobimo enakomerno porazdelitev, kjer so vse celico enako verjetne

$$\boldsymbol{p}(X_{\infty}|U_{0:\infty}) = [0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T$$

Te rezultate smo potrdili tudi v programskem okolju Matlab (program 6.8). Rezultati so grafično prikazani na sliki 6.14.

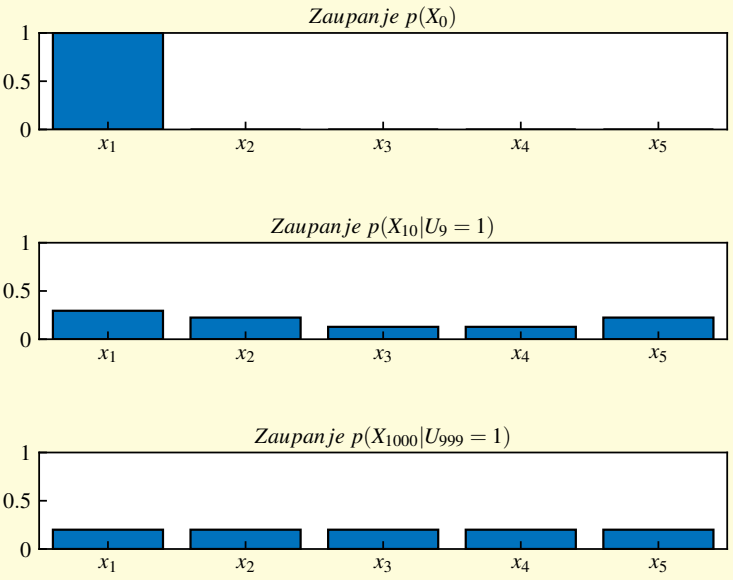
Program 6.8

./src/prb/example_motion3.m

```
disp('Začetno zaupanje p(X0)')
p_X0 = [1 0 0 0 0]

P_xxu_null = 0.8; % P(X=i|X'=j,U'=u), i=j+u
P_xxu_less = 0.1; % P(X=i|X'=j,U'=u), i=j+u-1
P_xxu_more = 0.1; % P(X=i|X'=j,U'=u), i=j+u+1

p_X = p_X0;
for k=1:1000
    p_xXu = [P_xxu_less 0 0 P_xxu_more P_xxu_null]; % Za U=1
    p_Xu = zeros(1,5);
    for i=1:5
        p_Xu(i) = p_xXu*p_X.';
        p_xXu = p_xXu([end 1:end-1]);
    end
    p_X = p_Xu;
    if k==10
        disp('Zaupanje p(X10|U9=1)');
        p_X10 = p_X
    elseif k==1000
        disp('Zaupanje p(X1000|U999=1)');
        p_X1000 = p_X
    end
end
```

```
Začetno zaupanje p(X0)
p_X0 =
     1     0     0     0     0
Zaupanje p(X10|U9=1)
```

```
p_X10 =
    0.2949    0.2243    0.1283    0.1283    0.2243
Zaupanje p(X1000|U999=1)
p_X1000 =
    0.2000    0.2000    0.2000    0.2000    0.2000
```



Slika 6.14: Porazdelitve trenutnega stanja v treh časovnih trenutkih iz primera 6.13

6.4.7 Lokalizacija v okolju

Ko mobilni robot pozna zemljevid okolja, lahko oceni svojo lokacijo v okolju, tudi če ne pozna svoje začetne lokacije. Lokacijo mobilnega sistema lahko natančno določimo s porazdelitvijo. Proces ugotavljanja lokacije v okolju imenujemo **lokalizacija**. Lokalizacija združuje proces zaznavanja (meritev) in akcije (premik). Kot smo že omenili, meritve v okolju povečujejo znanje o lokaciji, gibanje mobilnega sistema v okolju pa to znanje zmanjšuje.

Lokalizacija je postopek, pri katerem mobilni sistem stalno posodablja porazdelitev, ki predstavlja njegovo znanje o svoji lokaciji v okolju. Maksimum porazdelitve (če obstaja) predstavlja najverjetnejšo lokacijo mobilnega sistema.

Pri procesu lokalizacije v bistvu izvajamo Bayesov filter (algoritem 3), ki združuje procesa premikanja in zaznavanja.

Primer 6.14

Mobilni sistem se premika v okolici iz primera 6.8, kjer najprej izvede premik, nato pa zaznava okolico. Njegova začetna lega ni znana, kar lahko opišemo z enakomerno porazdelitvijo $p(X_0) = bel(X_0) = [0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]$.

Premik za u_k celic v nasprotni smeri urinega kazalca je točen v 80 %, v 10 % pa je za eno celico prekratek ali predolg

$$\begin{aligned} p(X_k = x_i | X_{k-1} = x_j, U_{k-1} = u_{k-1}) &= 0{,}8 \qquad \text{za } i = j + u_{k-1} \\ p(X_k = x_i | X_{k-1} = x_j, U_{k-1} = u_{k-1}) &= 0{,}1 \qquad \text{za } i = j + u_{k-1} - 1 \\ p(X_k = x_i | X_{k-1} = x_j, U_{k-1} = u_{k-1}) &= 0{,}1 \qquad \text{za } i = j + u_{k-1} + 1 \end{aligned}$$

Mobilni robot pravilno zazna temno celico z verjetnostjo 0,6, svetlo celico pa pravilno zazna z verjetnostjo 0,8. To lahko v matematični obliki zapišemo kot

$$\begin{aligned} P(Z = dark | X = dark) &= 0{,}6 \qquad & P(Z = bright | X = dark) &= 0{,}4 \\ P(Z = bright | X = bright = 0{,}8 \qquad & & P(Z = dark | X = bright = 0{,}2 \end{aligned}$$

V vsakem časovnem trenutku dobi mobilni robot ukaz za premik za eno celico v nasprotni smeri urinega kazalca ($u_{k-1} = 1$). Zaporedje prvih treh meritev je $z_{1:3} = [bright,\ dark,\ dark]$.

1. Kakšno je zaupanje v prvem trenutku $k = 1$?
2. Kakšno je zaupanje v drugem trenutku $k = 2$?
3. Kakšno je zaupanje v tretjem trenutku $k = 3$?
4. V kateri celici se mobilni robot nahaja z največjo verjetnostjo po tretjem koraku?

Rešitev

Po vsakem premiku izvedemo predikcijski korak Bayesovega filtra (algoritem 3), po meritvi (zaznavi) pa korekcijski korak.

1. Predikcijski korak izvedemo na osnovi izvedenega premika. Mali x_i, $i \in \{1, \ldots, 5\}$, označuje, da se lokacija (stanje) mobilnega sistema nahaja v

celici i, veliki X_k pa označuje vektor vseh možnih stanj v trenutku k

$$\begin{aligned}
bel_p(X_1 = x_1) &= \sum_{x_i} P(X_1 = x_1|X_0 = x_i, u_0) bel(X_0 = x_i) = \\
&= \boldsymbol{p}^T(X_1 = x_1|X_0, u_0)\boldsymbol{bel}(X_0) = \\
&= [0{,}1,\ 0,\ 0,\ 0{,}1,\ 0{,}8][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = 0{,}2 \\
bel_p(X_1 = x_2) &= [0{,}8,\ 0{,}1,\ 0,\ 0,\ 0{,}1][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = 0{,}2 \\
bel_p(X_1 = x_3) &= [0{,}1,\ 0{,}8,\ 0{,}1,\ 0,\ 0][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = 0{,}2 \\
bel_p(X_1 = x_4) &= [0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = 0{,}2 \\
bel_p(X_1 = x_5) &= [0,\ 0,\ 0{,}1,\ 0{,}8,\ 0{,}1][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T = 0{,}2
\end{aligned}$$

Torej je celotna porazdelitev (zaupanje) predikcijskega koraka

$$\boldsymbol{bel}_p(X_1) = [0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T$$

Na osnovi meritve se oceni korekcijski korak Bayesovega filtra

$$\begin{aligned}
bel(X_1 = x_1) &= \eta\, p(Z_1 = bright|x_1)\, bel_p(X_1 = x_1) = \eta\, 0{,}8 \cdot 0{,}2 = \eta\, 0{,}16 \\
bel(X_1 = x_2) &= \eta\, p(Z_1 = bright|x_2)\, bel_p(X_1 = x_2) = \eta\, 0{,}8 \cdot 0{,}2 = \eta\, 0{,}16 \\
bel(X_1 = x_3) &= \eta\, p(Z_1 = bright|x_3)\, bel_p(X_1 = x_3) = \eta\, 0{,}4 \cdot 0{,}2 = \eta\, 0{,}08 \\
bel(X_1 = x_4) &= \eta\, p(Z_1 = bright|x_4)\, bel_p(X_1 = x_4) = \eta\, 0{,}4 \cdot 0{,}2 = \eta\, 0{,}08 \\
bel(X_1 = x_5) &= \eta\, p(Z_1 = bright|x_5)\, bel_p(X_1 = x_5) = \eta\, 0{,}8 \cdot 0{,}2 = \eta\, 0{,}16
\end{aligned}$$

Ko upoštevamo še normirni faktor

$$\eta = \frac{1}{0{,}16 + 0{,}16 + 0{,}08 + 0{,}08 + 0{,}16} = 1{,}56$$

dobimo posodobljeno porazdelitev (zaupanje)

$$\boldsymbol{bel}(X_1) = [0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T$$

Enak rezultat lahko dobimo iz

$$\begin{aligned}
\boldsymbol{bel}(X_1) &= \frac{\boldsymbol{p}^T(Z_1 = bright|X_1) \star \boldsymbol{bel}_p^T(X_1)}{\boldsymbol{p}^T(Z_1 = bright|X_1)\boldsymbol{bel}_p(X_1)} = \\
&= \frac{[0{,}8,\ 0{,}8,\ 0{,}4,\ 0{,}4,\ 0{,}8] \star [0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]}{[0{,}8,\ 0{,}8,\ 0{,}4,\ 0{,}4,\ 0{,}8][0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2,\ 0{,}2]^T} = \\
&= [0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T
\end{aligned}$$

2. Postopek iz prvega primera ponovimo na zadnjem rezultatu, da dobimo zaupanje v stanje v trenutku $k = 1$. Najprej ponovimo predikcijski korak

$$\begin{aligned}
bel_p(X_2 = x_1) &= [0{,}1,\ 0,\ 0,\ 0{,}1,\ 0{,}8][0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T = 0{,}237 \\
bel_p(X_2 = x_2) &= [0{,}8,\ 0{,}1,\ 0,\ 0,\ 0{,}1][0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T = 0{,}25 \\
bel_p(X_2 = x_3) &= [0{,}1,\ 0{,}8,\ 0{,}1,\ 0,\ 0][0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T = 0{,}237 \\
bel_p(X_2 = x_4) &= [0,\ 0{,}1,\ 0{,}8,\ 0{,}1,\ 0][0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T = 0{,}138 \\
bel_p(X_2 = x_5) &= [0,\ 0,\ 0{,}1,\ 0{,}8,\ 0{,}1][0{,}25,\ 0{,}25,\ 0{,}125,\ 0{,}125,\ 0{,}25]^T = 0{,}138
\end{aligned}$$

Celotna porazdelitev za predikcijski korak je

$$\boldsymbol{bel}_p(X_2) = [0{,}237,\ 0{,}25,\ 0{,}237,\ 0{,}138,\ 0{,}138]^T$$

za korekcijski korak pa

$$\boldsymbol{bel}(X_2) = \frac{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2]^T \star [0{,}237,\ 0{,}25,\ 0{,}237,\ 0{,}138,\ 0{,}138]^T}{[0{,}2,\ 0{,}2,\ 0{,}6,\ 0{,}6,\ 0{,}2][0{,}237,\ 0{,}25,\ 0{,}237,\ 0{,}138,\ 0{,}138]^T} = \\ = [0{,}136,\ 0{,}143,\ 0{,}407,\ 0{,}236,\ 0{,}079]^T$$

3. Podobno kot v prejšnjih dveh primerih lahko dobimo porazdelitev oz. zaupanje za trenutek $k = 3$

$$\boldsymbol{bel}_p(X_3) = [0{,}1,\ 0{,}131,\ 0{,}167,\ 0{,}363,\ 0{,}237]^T$$
$$\boldsymbol{bel}(X_3) = [0{,}048,\ 0{,}063,\ 0{,}245,\ 0{,}528,\ 0{,}115]^T$$

4. Po tretjem koraku se mobilni robot najverjetneje nahaja v četrti celici, z verjetnostjo 52,8 %. Druga najverjetnejša celica je tretja celica, kjer se nahaja z verjetnostjo 24,5 %.

Zaupanja stanj za vse tri časovne trenutke so grafično predstavljena na sliki 6.15. V programu 6.9 je prikazana Matlab koda rešitve.

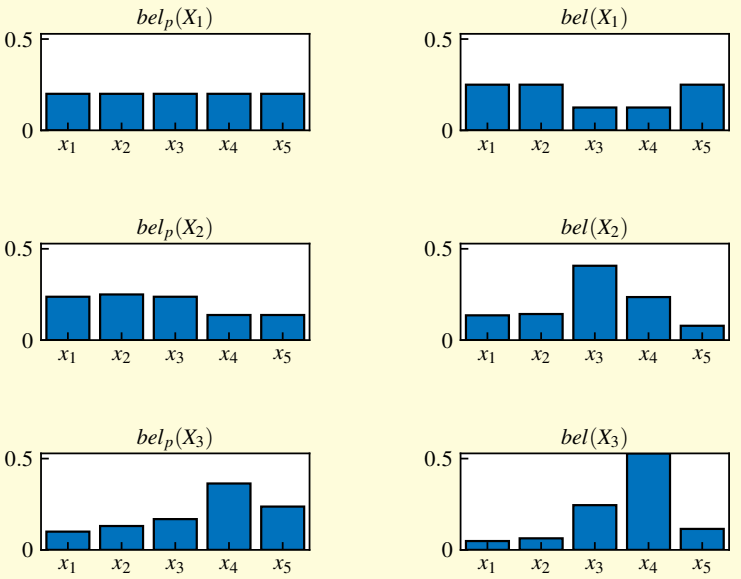
Program 6.9

./src/prb/example_localization.m

```
disp('Začetno zaupanje p(X0)')
bel_X0 = ones(1,5)/5

P_xxu_null = 0.8; % P(X=i|X'=j,U'=u), i=j+u
P_xxu_less = 0.1; % P(X=i|X'=j,U'=u), i=j+u-1
P_xxu_more = 0.1; % P(X=i|X'=j,U'=u), i=j+u+1

p_ZdX = [0.2 0.2 0.6 0.6 0.2]; % p(Z=dark|X)
p_ZbX = 1-p_ZdX;               % p(Z=bright|X)

bel_X = bel_X0;
for k=1:3
    % Predikcijski korak
    p_xXu = [P_xxu_less 0 0 P_xxu_more P_xxu_null]; % Za U=1
    belp_X = zeros(1,5);
    for i=1:5
        belp_X(i) = p_xXu*bel_X.';
        p_xXu = p_xXu([end 1:end-1]);
    end

    % Korekcijski korak
    if k==1
        bel_X = p_ZbX.*belp_X;
    else
        bel_X = p_ZdX.*belp_X;
    end
    bel_X = bel_X/sum(bel_X);

```

```
    if k==1
        disp('Zaupanji belp_X1 in bel_X1')
        belp_X1 = belp_X
        bel_X1 = bel_X
    elseif k==2
        disp('Zaupanji belp_X2 in bel_X2')
        belp_X2 = belp_X
        bel_X2 = bel_X
    elseif k==3
        disp('Zaupanji belp_X3 in bel_X3')
        belp_X3 = belp_X
        bel_X3 = bel_X
        disp('Najmanj do najbolj verjeten položaj')
        [m,mi] = sort(bel_X)
    end
end
```

```
Začetno zaupanje p(X0)
bel_X0 =
    0.2000    0.2000    0.2000    0.2000    0.2000
Zaupanji belp_X1 in bel_X1
belp_X1 =
    0.2000    0.2000    0.2000    0.2000    0.2000
bel_X1 =
    0.2500    0.2500    0.1250    0.1250    0.2500
Zaupanji belp_X2 in bel_X2
belp_X2 =
    0.2375    0.2500    0.2375    0.1375    0.1375
bel_X2 =
    0.1357    0.1429    0.4071    0.2357    0.0786
Zaupanji belp_X3 in bel_X3
belp_X3 =
    0.1000    0.1307    0.1686    0.3636    0.2371
bel_X3 =
    0.0484    0.0633    0.2450    0.5284    0.1149
Najmanj do najbolj verjeten položaj
m =
    0.0484    0.0633    0.1149    0.2450    0.5284
mi =
     1     2     5     3     4
```



Slika 6.15: Porazdelitev trenutnega stanja v treh trenutkih iz primera 6.14

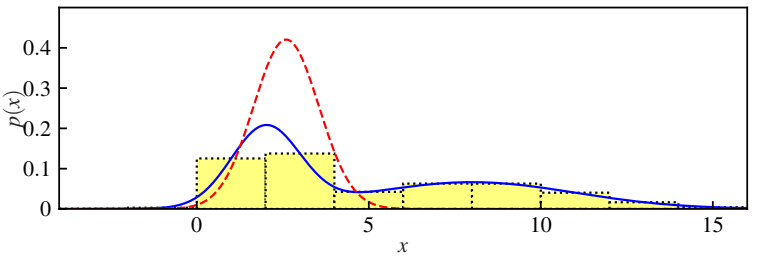
6.5 Kalmanov filter

Kalmanov filter [6] je eden najpomembnejših algoritmov za ocenjevanje in napovedovanje stanj, ki se uporablja v številnih aplikacijah na različnih inženirskih področjih, tudi v avtonomnih mobilnih sistemih. Zasnovan je kot ocenjevalnik stanj linearnih sistemov, kjer lahko signali sistema vsebujejo šum. Algoritem sestavljata dva (tipična) koraka, predikcijski in korekcijski korak, ki se izvedeta v vsakem časovnem trenutku. V predikcijskem koraku napovemo najnovejše stanje skupaj z njegovimi negotovostmi. Ko je nova meritev na voljo, se izvede korekcijski korak, kjer se stohastična meritev utežno združi s napovedno oceno stanja, pri čemer imajo manj negotove vrednosti večjo utež. Algoritem je rekurziven in omogoča sprotno ocenjevanje trenutnega stanja sistema ob upoštevanju negotovosti sistema in meritve.

Klasičen Kalmanov filter predvideva normalno porazdeljene šume, torej je porazdelitev šuma Gaussova funkcija

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

kjer je μ srednja vrednost (matematično upanje) in σ^2 varianca. Gaussova funkcija je unimodalna (levo in desno od (edinega) maksimuma funkcija monotono



Slika 6.16: Primer porazdelitve zvezne spremenljivke x (polna krivulja), aproksimacija z Gaussovo funkcijo (črtkana krivulja) in aproksimacija s histogramom (pikčasta krivulja)

pada proti 0) za razliko od splošnih porazdelitev, ki so običajno večmodalne (obstaja več lokalnih maksimumov). Pri predpostavki, da je porazdelitev zveznih spremenljivk unimodalna, lahko Kalmanov filter uporabimo za optimalno ocenjevanje stanj. V kolikor pa niso vse spremenljivke unimodalne, je ocena dobljenih stanj suboptimalna. Poleg tega je vprašljiva tudi konvergenca ocene k pravi vrednosti. Bayesov filter nima omenjenih problemov, vendar je njegova uporabnost omejena na enostavna zvezna ali diskretna okolja s končnim številom stanj.

Na sliki 6.16 je prikazan primer zvezne porazdelitve, ki ni unimodalna. Zvezna porazdelitev je aproksimirana z Gaussovo funkcijo in histogramom (prostor je razdeljen na diskretne intervale). Aproksimacija z Gaussovo funkcijo se uporablja pri Kalmanovem filtru, histogram pa pri Bayesovem filtru.

Bistvo korekcijskega koraka (glejte Bayesov filter (6.21)) je združevanje informacij iz dveh neodvisnih virov, to je iz meritve senzorja in napovedi stanja na osnovi predhodnih ocen stanja. Na primeru 6.15 si poglejmo, kako lahko optimalno združimo dve neodvisni oceni iste spremenljivke x, če poznamo vrednost in varianco (zaupanje) obeh virov.

Primer 6.15

Imamo dve neodvisni oceni spremenljivke x. Vrednost prve ocene je x_1 z varianco σ_1^2, vrednost druge ocene pa je x_2 z varianco σ_2^2. Kakšna je optimalna linearna kombinacija teh dveh ocen, ki predstavljata oceno stanja $\hat{x}$ z minimalno varianco?

Rešitev

Ocena optimalne vrednosti spremenljivke x je linearna kombinacija dveh meritev

$$\hat{x} = \omega_1 x_1 + \omega_2 x_2$$

kjer sta parametra ω_1 in ω_2 iskani uteži, ki izpolnjujeta pogoj $\omega_1 + \omega_2 = 1$. Optimalni vrednosti uteži minimizirata varianco σ^2 optimalne ocene $\hat{x}$. Varianca ocene je torej

$$\begin{aligned}
\sigma^2 &= \mathrm{E}\{(\hat{x} - \mathrm{E}\{\hat{x}\})^2\} \\
&= \mathrm{E}\{(\omega_1 x_1 + \omega_2 x_2 - \mathrm{E}\{\omega_1 x_1 + \omega_2 x_2\})^2\} \\
&= \mathrm{E}\{(\omega_1 x_1 + \omega_2 x_2 - \omega_1 \mathrm{E}\{x_1\} - \omega_2 \mathrm{E}\{x_2\})^2\} \\
&= \mathrm{E}\Big\{(\omega_1 (x_1 - \mathrm{E}\{x_1\}) + \omega_2 (x_2 - \mathrm{E}\{x_2\}))^2\Big\} \\
&= \mathrm{E}\Big\{\omega_1^2 (x_1 - \mathrm{E}\{x_1\})^2 + \omega_2^2 (x_2 - \mathrm{E}\{x_2\})^2 + 2\omega_1\omega_2 (x_1 - \mathrm{E}\{x_1\})(x_2 - \mathrm{E}\{x_2\})\Big\} \\
&= \omega_1^2 \mathrm{E}\{(x_1 - \mathrm{E}\{x_1\})^2\} + \omega_2^2 \mathrm{E}\{(x_2 - \mathrm{E}\{x_2\})^2\} + 2\omega_1\omega_2 \mathrm{E}\{(x_1 - \mathrm{E}\{x_1\})(x_2 - \mathrm{E}\{x_2\})\} \\
&= \omega_1^2 \sigma_1^2 + \omega_2^2 \sigma_2^2 + 2\omega_1\omega_2 \mathrm{E}\{(x_1 - \mathrm{E}\{x_1\})(x_2 - \mathrm{E}\{x_2\})\}
\end{aligned}$$

Ker sta spremenljivki x_1 in x_2 neodvisni, sta neodvisni tudi razliki $x_1 - \mathrm{E}\{x_1\}$ in $x_2 - \mathrm{E}\{x_2\}$ ter velja $\mathrm{E}\{(x_1 - \mathrm{E}\{x_1\})(x_2 - \mathrm{E}\{x_2\})\} = 0$. Torej lahko zapišemo

$$\sigma^2 = \omega_1^2 \sigma_1^2 + \omega_2^2 \sigma_2^2$$

ali po uvedbi $\omega_2 = \omega$ in $\omega_1 = 1 - \omega$

$$\sigma^2 = (1 - \omega)^2 \sigma_1^2 + \omega^2 \sigma_2^2$$

Iščemo vrednost uteži ω, ki minimizira varianco, in jo lahko pridobimo s pomočjo odvoda variance

$$\frac{\partial}{\partial \omega} \sigma^2 = -2(1 - \omega)\sigma_1^2 + 2\omega\sigma_2^2 = 0$$

kjer je rešitev

$$\omega = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

Končna ocena (z minimalno varianco) je

$$\hat{x} = \frac{\sigma_2^2 x_1 + \sigma_1^2 x_2}{\sigma_1^2 + \sigma_2^2} \tag{6.22}$$

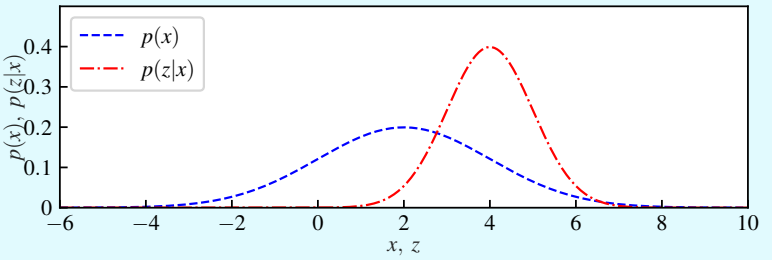
najmanjša varianca ocene pa

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2} = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-1} \tag{6.23}$$

Dobljeni rezultati potrjujejo, da vir z manjšo varianco (večje zaupanje) bolj prispeva h končni oceni in obratno.

Primer 6.16

V določenem trenutku je podana začetna ocena stanja $x = 2$ z varianco $\sigma^2 = 4$. Nato s senzorjem izmerimo vrednost stanja $z = 4$ z varianco senzorja $\sigma_z^2 = 1$. Gaussovi porazdelitvi stanja in meritve sta prikazani na sliki 6.17.



Slika 6.17: Porazdelitev stanja (črtkana krivulja) in meritve (krivulja *črta-pika*)

Kakšna je posodobljena optimalna ocena stanja, ki združuje informacijo predhodne ocene stanja in trenutne meritve? Kakšna je porazdelitev posodobljene optimalne ocene stanja?

Rešitev

Na podlagi slike 6.17 lahko predvidimo, da bo srednja vrednost posodobljenega stanja x' bližje srednji vrednosti meritve, ker ima le-ta manjšo varianco (negotovost). Z upoštevanjem (6.22) dobimo posodobljeno oceno stanja

$$x' = \frac{\sigma_z^2 x + \sigma^2 z}{\sigma^2 + \sigma_z^2} = 3{,}6$$

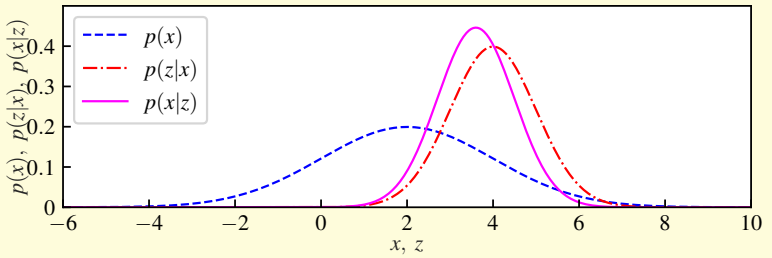
Varianca posodobljene ocene σ'^2 je manjša od obeh predhodnih varianc, saj integracija informacij predhodne ocene in meritve zmanjšuje negotovost posodobljene ocene. Varianca posodobljene ocene, izračunana s pomočjo (6.23), je

$$\sigma'^2 = \left(\frac{1}{\sigma^2} + \frac{1}{\sigma_z^2}\right)^{-1} = 0{,}8$$

in standardna deviacija je

$$\sigma' = \sqrt{\sigma'^2} = 0{,}894$$

Posodobljena porazdelitev $p(x|z)$ stanja po opravljeni korekciji na osnovi meritve je prikazana na sliki 6.18.



Slika 6.18: Porazdelitev začetnega stanja (črtkana krivulja), meritve (krivulja *črta-pika*) in posodobljenega stanja (polna krivulja)

Za izpeljavo algoritma rekurzivnega ocenjevanja stanja uporabimo ugotovitve iz primera 6.15. V vsakem časovnem trenutku s pomočjo senzorja pridobimo novo meritev stanja $z(k) = x(k) + n(k)$, kjer je $n(k)$ šum meritve. Predpostavimo, da je varianca meritve $\sigma_z^2(k)$ znana. Posodobljena optimalna ocena stanja je kombinacija prejšnje ocene stanja $\hat{x}(k)$ in trenutne meritve $z(k)$

$$\hat{x}(k+1) = (1-\omega)\,\hat{x}(k) + \omega(k)z(k) = \hat{x}(k) + \omega(z(k) - \hat{x}(k))$$

Varianca posodobljenega stanja je

$$\sigma^2(k+1) = \frac{\sigma^2(k)\sigma_z^2(k)}{\sigma^2(k) + \sigma_z^2(k)} = (1-\omega)\sigma^2(k)$$

kjer je

$$\omega = \frac{\sigma^2(k)}{\sigma^2(k) + \sigma_z^2(k)}$$

Glede na podano začetno oceno stanja x, $\hat{x}(0)$ in njeno varianco $\sigma^2(0)$ lahko optimalno združimo meritve $z(1), z(2), \ldots, z(k)$ tako, da ocenimo trenutno vrednost stanja in njegovo varianco. To predstavlja osnovno idejo korekcijskega koraka Kalmanovega filtra.

Predikcijski korak Kalmanovega filtra podaja napoved stanja ob znani vhodni akciji. Izhodiščna ocena stanja $\hat{x}(k)$ ima porazdelitev z varianco $\sigma^2(k)$. Na enak način ima akcija $u(k)$, ki izvede premik iz stanja $x(k)$ v $x(k+1)$, porazdelitev (negotovost premika) $\sigma_u^2(k)$. Na primeru 6.17 si poglejmo vrednost stanja in njegove variance po izvedeni akciji (premiku).

Primer 6.17

Imamo izhodiščno oceno stanja $\hat{x}(k)$ z varianco $\sigma^2(k)$. Nato izvedemo akcijo $u(k)$, ki predstavlja neposreden premik stanja z negotovostjo (varianco) $\sigma_u^2(k)$. Kakšna je vrednost ocene stanja in njene negotovosti po premiku?

Rešitev

Posodobljena ocena stanja po premiku je

$$\hat{x}(k+1) = \hat{x}(k) + u(k)$$

in negotovost te ocene je

$$\begin{aligned}
\sigma^2(k+1) &= \mathrm{E}\Big\{\left(\hat{x}(k+1) - \mathrm{E}\{\hat{x}(k+1)\}\right)^2\Big\} \\
&= \mathrm{E}\Big\{\left(\hat{x}(k) + u(k) - \mathrm{E}\{\hat{x}(k) + u(k)\}\right)^2\Big\} \\
&= \mathrm{E}\Big\{\left((\hat{x}(k) - \mathrm{E}\{\hat{x}(k)\}) + (u(k) - \mathrm{E}\{u(k)\})\right)^2\Big\} \\
&= \mathrm{E}\big\{(\hat{x}(k) - \mathrm{E}\{\hat{x}(k)\})^2 + (u(k) - \mathrm{E}\{u(k)\})^2\big\} + \\
&\quad + \mathrm{E}\{2(\hat{x}(k) - \mathrm{E}\{\hat{x}(k)\})(u(k) - \mathrm{E}\{u(k)\})\} \\
&= \sigma^2(k) + \sigma_u^2(k) + \\
&\quad + \mathrm{E}\{2\left(\hat{x}(k) - \mathrm{E}\{\hat{x}(k)\}\right)(u(k) - \mathrm{E}\{u(k)\}))\}
\end{aligned} \tag{6.24}$$

Ker sta $\hat{x}$ in u neodvisna, velja $\mathrm{E}\{2\left(\hat{x}(k) - \mathrm{E}\{\hat{x}(k)\}\right)\left(u(k) - \mathrm{E}\{u(k)\}\right)\} = 0$ in (6.24) se poenostavi v

$$\sigma^2(k+1) = \sigma^2(k) + \sigma_u^2(k)$$

Algoritem za poenostavljeno izvedbo Kalmanovega filtra

Kalmanov filter za enostaven primer z enim stanjem je podan v algoritmu 4, kjer veličine z oznako $(\cdot)_{k|k-1}$ predstavljajo ocenjene vrednosti iz predikcijskega koraka in veličine z oznako $(\cdot)_{k|k}$ vrednosti iz korekcijskega koraka. Zavoljo boljše preglednosti uporabimo zapis $u(k-1) = u_{k-1}$ in $z(k) = z_k$.

Kalmanov filter je sestavljen iz dveh korakov (predikcija in korekcija), ki se izvajata eden za drugim v zanki. V predikcijskem koraku uporabimo samo znano akcijo in določimo (napovemo) vrednost stanja v naslednjem časovnem trenutku. Torej iz začetnega zaupanja določimo novo zaupanje, katerega negotovost je večja od začetne. V korekcijskem koraku pa uporabimo meritev za izboljšanje napovedanega zaupanja tako, da ima ocena novega (popravljenega) stanja manjšo negotovost kot prejšnje zaupanje. V obeh korakih sta potrebna samo dva vhoda: v predikcijskem koraku morata biti znana vrednost predhodnega stanja $\hat{x}_{k-1|k-1}$

Algorithm 4 Kalmanov filter za eno stanje

function KALMAN_FILTER($\hat{x}_{k-1|k-1}$, u_{k-1}, z_k, $\sigma^2_{k-1|k-1}$, $\sigma_{u^2_{k-1}}$, $\sigma_{z^2_k}$)

Predikcijski korak:

$\hat{x}_{k|k-1} \leftarrow \hat{x}_{k-1|k-1} + u_{k-1}$

$\sigma^2_{k|k-1} \leftarrow \sigma^2_{k-1|k-1} + \sigma_{u^2_{k-1}}$

Korekcijski korak:

$\omega_k \leftarrow \frac{\sigma^2_{k|k-1}}{\sigma^2_{k|k-1}+\sigma_{z^2_k}}$

$\hat{x}_{k|k} \leftarrow \hat{x}_{k|k-1} + \omega_k(z_k - \hat{x}_{k|k-1})$

$\sigma^2_{k|k} \leftarrow (1-\omega_k)\sigma^2_{k|k-1}$

return $\hat{x}_{k|k}$, $\sigma^2_{k|k}$

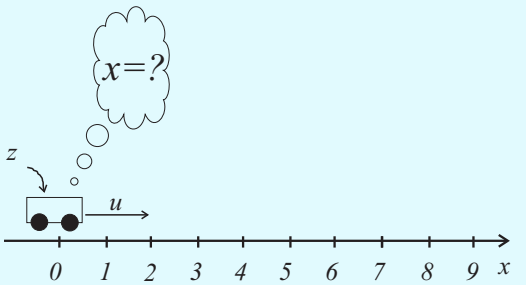
end function

in izvedena akcija u_{k-1}, v predikcijskem koraku pa napovedano stanje $\hat{x}_{k|k-1}$ in meritev z_k. Podana mora biti tudi varianca premika stanja $\sigma^2_{k-1|k-1}$, varianca vhodne akcije $\sigma_{u^2_{k-1}}$ in varianca meritve $\sigma_{z^2_k}$.

Primer 6.18

Imamo mobilnega robota, ki se lahko premika samo v eni dimenziji. Njegova začetna pozicija je neznana (slika 6.19). Predpostavimo začetno pozicijo $\hat{x}_0 = 3$ z veliko varianco $\sigma^2_0 = 100$ (dejanske pozicije $x_0 = 0$ ne poznamo).

Mobilni robot se v vsakem trenutku $k = 0, \ldots, 4$ premakne za $u_{0:4} = (2, 3, 2, 1, 1)$ enot, nato pa izvedemo meritve pozicije robota $z_{1:5} = (2, 5, 7, 8, 9)$ v trenutkih $k = 1, \ldots, 5$. Pomik in meritev sta motena z normalno porazdeljenim belim šumom, kar lahko opišemo s konstantno negotovostjo pomika $\sigma^2_u = 2$ in negotovostjo meritve $\sigma^2_z = 4$.



Slika 6.19: Lokalizacija mobilnega robota v enodimenzionalnem prostoru z neznano začetno pozicijo

Kakšna je ocenjena pozicija robota in negotovost te ocene?

Rešitev

Dani problem lokalizacije mobilnega robota rešimo z izvajanjem algoritma 4. V prvem časovnem trenutku ($k = 1$) najprej izračunamo predikcijo stanja in varianco

$$\hat{x}_{1|0} = \hat{x}_{0|0} + u_0 = 3 + 2 = 5$$
$$\sigma^2_{1|0} = \sigma^2_{0|0} + {\sigma_u}^2 = 100 + 2 = 102$$

nato pa izvedemo korekcijski korak Kalmanovega filtra v prvem časovnem trenutku $k = 1$ in dobimo

$$\omega_1 = \frac{\sigma^2_{1|0}}{\sigma^2_{1|0} + {\sigma_z}^2} = \frac{102}{102 + 4} = 0{,}962$$
$$\hat{x}_{1|1} = \hat{x}_{1|0} + \omega_1(z_1 - \hat{x}_{1|0}) = 5 + 0{,}962(2 - 5) = 2{,}113$$
$$\sigma^2_{1|1} = (1 - \omega_1)\sigma^2_{1|0} = (1 - 0{,}962)102 = 3{,}849$$

Predikcijo in korekcijo izvedemo še za ostale časovne trenutke. Za predikcijske korake dobimo

$$\hat{x}_{1:5|0:4} = (5{,}00,\ 5{,}11,\ 7{,}05,\ 8{,}02,\ 9{,}01)$$
$$\sigma^2_{1:5|0:4} = (102,\ 5{,}85,\ 4{,}38,\ 4{,}09,\ 4{,}02)$$

in za korekcijske korake

$$\hat{x}_{1:5|1:5} = (2{,}11,\ 5{,}05,\ 7{,}02,\ 8{,}01,\ 9{,}01)$$
$$\sigma^2_{1:5|1:5} = (3{,}85,\ 2{,}38,\ 2{,}09,\ 2{,}02,\ 2{,}01)$$

Dobljeni rezultati kažejo, da lahko pozicijo mobilnega robota določimo po nekaj korakih z negotovostjo 2, kar ustreza negotovosti predikcije in meritve $\left(\frac{1}{\sigma^2_{5|4}} + \frac{1}{\sigma^2_z}\right)^{-1} = 2{,}01$. Negotovost predikcijske ocene pozicije konvergira proti 4, kar je v skladu z negotovostjo korekcije iz prejšnjega časovnega trenutka in meritve $\sigma^2_{4|4} + \sigma^2_u = 4{,}02$.

6.5.1 Kalmanov filter v matrični obliki

Sisteme z več vhodi, stanji in izhodi lahko za večjo preglednost podamo v matrični obliki. Splošni linearni sistem zapišemo v prostoru stanj kot

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) + \boldsymbol{F}\boldsymbol{w}(k) \\ \boldsymbol{z}(k) &= \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{v}(k) \end{aligned} \tag{6.25}$$

kjer je $\boldsymbol{x}$ vektor stanj, $\boldsymbol{u}$ je vhodni vektor (akcija), $\boldsymbol{z}$ je izhodni vektor (meritev), $\boldsymbol{A}$ je sistemska matrika, $\boldsymbol{B}$ je vhodna matrika, $\boldsymbol{F}$ je vhodna matrika šuma, $\boldsymbol{C}$

je izhodna matrika, $\boldsymbol{w}(k)$ je vektor procesnega šuma in $\boldsymbol{v}$ je vektor izhodnega (merilnega) šuma. V kolikor se šum $\boldsymbol{w}$ doda vhodu sistema $\boldsymbol{u}$, velja $\boldsymbol{F} = \boldsymbol{B}$. Predpostavimo, da sta procesni šum $\boldsymbol{w}(k)$ in merilni šum $\boldsymbol{v}(k)$ medsebojno neodvisna bela šuma z ničelno srednjo vrednostjo in kovariančnima matrikama $\boldsymbol{Q}_k = \mathrm{E}\{\boldsymbol{w}(k)\boldsymbol{w}^T(k)\}$ in $\boldsymbol{R}_k = \mathrm{E}\{\boldsymbol{v}(k)\boldsymbol{v}^T(k)\}$.

Porazdelitev stanj $\boldsymbol{x}$, ki so motena z belim Gaussovim šumom, podamo v matrični obliki

$$p(\boldsymbol{x}) = \det(2\pi\boldsymbol{P})^{-\frac{1}{2}}\, e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{P}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

kjer je $\boldsymbol{P}$ kovariančna matrika napake ocene stanj.

Kalmanov filter predstavlja pristop za filtriranje in ocenjevanje zveznih stanj linearnih sistemov, ki so motena z normalnim šumom. Porazdelitev šuma je podana z Gaussovo funkcijo (Gaussov šum). Vhodni in merilni šum vplivata na notranja stanja sistema, ki jih želimo oceniti. V primeru linearnega modela sistema je tudi filtriran šum preko linearnega modela (npr. od vhodov do stanj) Gaussov šum. Torej mora biti sistem linearen, saj to zagotavlja Gaussovo porazdelitev šuma na stanjih, kar je izhodišče pri izpeljavi Kalmanovega filtra. Kalmanov filter bo konvergiral k pravi oceni stanj le v primeru linearnih sistemov, ki so moteni z Gaussovim šumom.

Kalmanov filter za linearni sistem (6.25) ima predikcijski korak

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{A}\boldsymbol{P}_{k-1|k-1}\boldsymbol{A}^T + \boldsymbol{F}\boldsymbol{Q}_{k-1}\boldsymbol{F}^T \end{aligned} \tag{6.26}$$

in korekcijski korak

$$\begin{aligned} \boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\left(\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R}_k\right)^{-1} \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1}) \\ \boldsymbol{P}_{k|k} &= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} \end{aligned} \tag{6.27}$$

V predikcijskem koraku določimo napovedno oceno $\hat{\boldsymbol{x}}_{k|k-1}$, ki temelji na predhodni oceni $\hat{\boldsymbol{x}}_{k-1|k-1}$, dobljeni iz meritev do trenutka $k-1$, in vhodu $\boldsymbol{u}(k-1)$. V korekcijskem koraku pa izračunamo trenutno oceno $\hat{\boldsymbol{x}}_{k|k}$, ki temelji na meritvah do trenutka k. Korekcijo stanja izvedemo z izračunom razlike med dejansko in ocenjeno meritvijo $\boldsymbol{z}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1}$. Ta razlika je znana tudi kot *inovacija* ali residuum meritve. Celotna korekcija stanja se izračuna kot produkt Kalmanovega ojačenja $\boldsymbol{K}_k$ in inovacije. Predikcijski korak lahko ovrednotimo vnaprej, medtem ko čakamo na novo meritev v trenutku k. Opazimo podobnost matričnega zapisa (6.26) in (6.27) z zapisom v algoritmu 4.

Izpeljimo izraz za kovariančno matriko napake ocene stanj v predikcijskem koraku

Kalmanovega filtra

$$\begin{aligned}
\boldsymbol{P}_{k|k-1} &= \mathrm{E}\big\{(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})^T\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1}\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} + \boldsymbol{F}\boldsymbol{w}_{k-1} - \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1|k-1} - \boldsymbol{B}\boldsymbol{u}_{k-1}\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{F}\boldsymbol{w}_{k-1} - \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1|k-1}\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{A}(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1}) + \boldsymbol{F}\boldsymbol{w}_{k-1}\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{A}(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1})\big\} + \mathrm{cov}\{\boldsymbol{F}\boldsymbol{w}_{k-1}\} \\
&= \mathrm{E}\big\{(\boldsymbol{A}(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1}))(\boldsymbol{A}(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1}))^T\big\} + \\
&\quad + \mathrm{E}\big\{(\boldsymbol{F}\boldsymbol{w}_{k-1})(\boldsymbol{F}\boldsymbol{w}_{k-1})^T\big\} \\
&= \mathrm{E}\big\{(\boldsymbol{A}(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1})(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1})^T\boldsymbol{A}^T\big\} + \\
&\quad + \mathrm{E}\big\{\boldsymbol{F}\boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T\boldsymbol{F}^T\big\} \\
&= \boldsymbol{A}\boldsymbol{P}_{k-1|k-1}\boldsymbol{A}^T + \boldsymbol{F}\boldsymbol{Q}_{k-1}\boldsymbol{F}^T
\end{aligned}$$

kjer smo v šesti vrstici upoštevali, da je procesni (vhodni) šum $\boldsymbol{w}_k$ v trenutku k neodvisen od napake ocene stanj v prejšnjem trenutku $(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1})$.

Izpeljimo še izraz za kovariančno matriko napake ocene stanj v korekcijskem delu Kalmanovega filtra

$$\begin{aligned}
\boldsymbol{P}_{k|k} &= \mathrm{E}\big\{(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})^T\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k}\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1} - \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1})\big\} \\
&= \mathrm{cov}\big\{\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1} - \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1})\big\} \\
&= \mathrm{cov}\big\{(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1}) - \boldsymbol{K}_k\boldsymbol{v}_k\big\} \\
&= \mathrm{cov}\big\{(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})\big\} + \mathrm{cov}\{\boldsymbol{K}_k\boldsymbol{v}_k\} \\
&= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{P}_{k|k-1}(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T
\end{aligned}$$

kjer smo v šesti vrstici upoštevali, da je merilni šum $\boldsymbol{v}_k$ nekoreliran z ostalimi členi. Dobljeni izraz za kovariančno matriko $\boldsymbol{P}_{k|k}$ je splošen in ga lahko uporabimo za poljubno ojačenje $\boldsymbol{K}_k$. Vendar pa izraz za $\boldsymbol{P}_{k|k}$ v (6.27) velja le za optimalno ojačenje (Kalmanovo ojačenje), ki minimizira povprečni kvadratni pogrešek korekcije $\mathrm{E}\Big\{\big|\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k}\big|^2\Big\}$. To je ekvivalentno minimizaciji vsote vseh diagonalnih elementov kovariančne matrike korekcije $\boldsymbol{P}_{k|k}$.

Splošni izraz za $\boldsymbol{P}_{k|k}$ lahko razširimo in preuredimo

$$\begin{aligned}
\boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})\boldsymbol{P}_{k|k-1}(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C})^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T \\
&= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T \\
&= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k(\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R}_k)\boldsymbol{K}_k^T \\
&= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k\boldsymbol{S}_k\boldsymbol{K}_k^T
\end{aligned}$$

kjer $\boldsymbol{S}_k = \boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R}_k$ predstavlja kovariančno matriko inovacije ($\boldsymbol{S}_k = \mathrm{cov}\big\{\boldsymbol{z}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1}\big\}$). Vsota diagonalnih členov $\boldsymbol{P}_{k|k}$ je minimalna, ko je odvod

$\boldsymbol{P}_{k|k}$ po $\boldsymbol{K}_k$ enak 0

$$\frac{\partial \boldsymbol{P}_{k|k}}{\partial \boldsymbol{K}_k} = -2(\boldsymbol{C}\boldsymbol{P}_{k|k-1})^T + 2\boldsymbol{K}_k\boldsymbol{S}_k = \boldsymbol{0}$$

kar vodi do optimalnega ojačenja v (6.27)

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{S}_k^{-1} = \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R}_k)^{-1}$$

Kovariančno matriko korekcije pri optimalnem ojačenju lahko izpeljemo, če optimalno ojačenje z desne strani pomnožimo s $\boldsymbol{S}_k\boldsymbol{K}_k^T$ in vstavimo v izraz za $\boldsymbol{P}_{k|k}$

$$\begin{aligned}\boldsymbol{P}_{k|k} &= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{K}_k\boldsymbol{S}_k\boldsymbol{K}_k^T \\ &= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} - \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T + \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\boldsymbol{K}_k^T \\ &= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1}\end{aligned}$$

Primer 6.19

Mobilni robot se vozi po ravnini in meri svojo pozicijo z GPS-om desetkrat v sekundi (čas vzorčenja $T_s = 0{,}1\,\mathrm{s}$). Meritev položaja je motena z Gaussovim šumom, ki ima varianco $10\,\mathrm{m}^2$. Robot se premika s hitrostjo $1\,\mathrm{m/s}$ v smeri x in s hitrostjo $0\,\mathrm{m/s}$ v smeri y. Varianca Gaussovega šuma hitrosti je $0{,}1\,\mathrm{m}^2/\mathrm{s}^2$. Na začetku opazovanja se mobilni robot nahaja v izhodišču $\boldsymbol{x} = [0,\ 0]^T$, vendar je naša ocena začetne pozicije $\hat{\boldsymbol{x}} = [3,\ 3]^T$ z začetno varianco

$$\boldsymbol{P}_0 = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

Kakšen je časovni potek ocene pozicije in njene variance?

Rešitev

Do rešitve lahko pridemo s pomočjo simulacije v okolju Matlab. Določimo model gibanja robota, kjer ocena stanja predstavlja pozicijo robota na ravnini $\hat{\boldsymbol{x}}_{k|k-1} = [x_k,\ y_k]^T$, vhod $\boldsymbol{u} = [v_x,\ v_y]^T$ pa predstavlja hitrost robota v smereh x in y. Torej je model za predikcijo stanja sistema

$$\hat{\boldsymbol{x}}_{k|k-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\hat{\boldsymbol{x}}_{k-1|k-1} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix}\boldsymbol{u}$$

Model meritve pozicije z GPS-om pa je

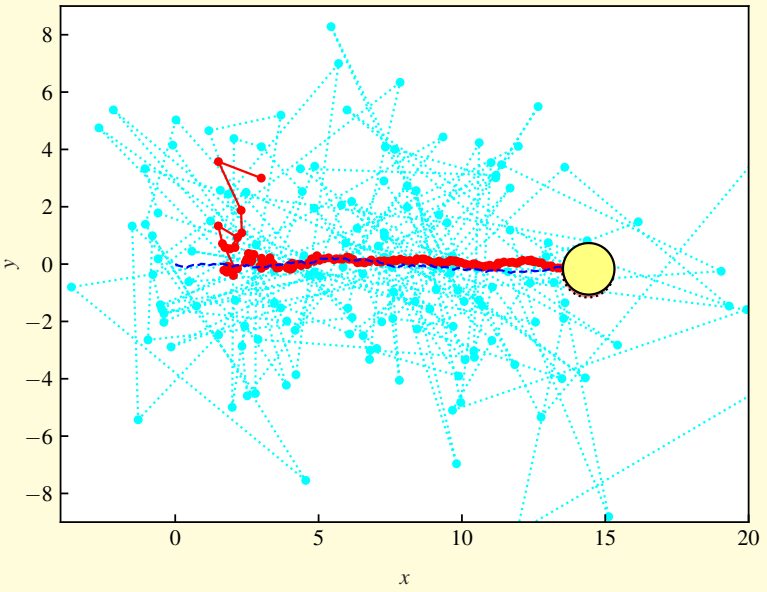
$$\hat{\boldsymbol{z}}_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\hat{\boldsymbol{x}}_{k|k-1}$$

Matlab koda rešitve je podana v programu 6.10, rezultati simulacije pa so grafično prikazani na slikah 6.20, 6.21 in 6.22. Rezultati simulacije potrjujejo, da

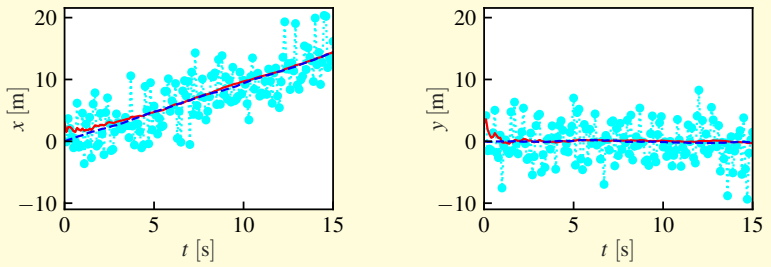
dobljena ocena pozicije konvergira k pravi oceni pozicije robota iz napačne začetne ocene. Rezultati so pričakovani, saj je srednja vrednost merjenega šuma nič. Vendar se varianca ocene s časom zmanjšuje in pade veliko pod varianco meritve (kinematičnemu modelu zaupamo bolj). Kalmanov filter omogoča optimalno združitev podatkov iz različnih virov (notranji kinematični model in zunanji model meritve), če so znane variance teh virov. Bralci lahko prosto eksperimentirate s parametri sistema in opazujete potek variance ocene in stopnjo konvergence.

Program 6.10: Rešitev primera 6.19
./src/prb/example_kf1.m

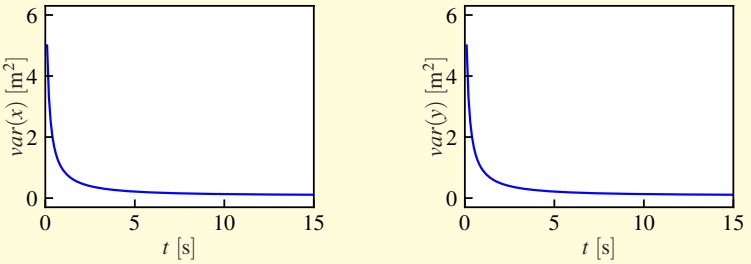
```
% Linearni model sistema v prostoru stanj
Ts = 0.1; % Računski korak
A = [1 0; 0 1];
B = [Ts 0; 0 Ts];
C = [1 0; 0 1];
F = B; % Šum je dodan na vhod.

xTrue = [0; 0]; % Prava začetna vrednost stanj
x = [3; 3]; % Začetna ocena stanj
P = diag([10 10]); % Varianca začetne ocene stanj
Q = diag([1 1]/10); % Varianca šuma modela gibanja
R = diag([10 10]); % Varianca šuma meritev GPS

% Zanka
N = 150;
for k = 1:N
    u = [1; 0]; % Ukazi za gibanje

    % Simulacija pravega položaja robota in pravih meritev
    xTrue = A*xTrue + B*u + F*sqrt(Q)*randn(2, 1);
    zTrue = C*xTrue + sqrt(R)*randn(2, 1);

    % Ocena položaja na podlagi znanih ukazov in meritev
    %%% Predikcija
    xPred = A*x + B*u;
    PPred = A*P*A.' + F*Q*F.';

    %%% Korekcija
    K = PPred*C.'/(C*PPred*C.' + R);
    x = xPred + K*(zTrue - C*xPred);
    P = PPred - K*C*PPred;
end
```



Slika 6.20: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija z meritvami (pikice na pikčasti krivulji) iz primera 6.19. Končna pozicija robota je označena s krogom.



Slika 6.21: Prava pozicija (črtkana krivulja) in ocena pozicije (polna krivulja) mobilnega robota na podlagi meritev (pikčasta krivulja) iz primera 6.19



Slika 6.22: Časovni potek variance pozicije mobilnega robota iz primera 6.19

6.5.2 Razširjeni Kalmanov filter

Kalmanov filter je razvit za linearne sisteme, vse motnje in šumi pa morajo biti opisljivi z (normalno) Gaussovo porazdelitvijo. Šum z Gaussovo porazdelitvijo je invarianten za linearne transformacije. Če Gaussov šum transformiramo z linearno funkcijo, je dobljeni šum še vedno Gaussov, spremenili so se samo njegovi parametri, ki jih lahko eksplicitno izračunamo iz znane linearne funkcije. Ravno zaradi tega je Kalmanov filter računsko učinkovit. V primeru transformacije vhodnega Gaussovega šuma z nelinearno funkcijo, izhodni šum ni več Gaussov, čeprav ga lahko še vedno aproksimiramo z Gaussovim šumom.

Če je katerokoli prehajanje stanja ali izhodna enačba sistema nelinearna funkcija, osnovni Kalmanov filter ne zagotavlja več optimalne ocene stanja. Problem nelinearnosti lahko rešimo z uporabo **razširjenega Kalmanovega filtra** (EKF, angl. *extended Kalman filter*), kjer nelinearnosti modela aproksimiramo z lokalnimi linearnimi modeli. Lokalni linearni model pridobimo iz nelinearnega sistema s pomočjo linearizacije (razvoj v Taylorjevo vrsto prvega reda) okoli trenutne ocene stanja. Z linearizacijo dobimo občutljivostne matrike (Jacobijeve matrike) za trenutne vrednosti ocenjenih stanj in meritev. Dobljeni linearni model omogoča izračun približka šuma, ki ni nujno Gaussov, z Gaussovo porazdelitvijo.

Uporaba linearizacije pri modeliranju šuma omogoča računsko učinkovito izvajanje razširjenega Kalmanovega filtra, kar je razlog za njegovo pogosto uporabo v praksi. Točnost linearne aproksimacije je odvisna od variance šuma (pri velikih negotovostih ali amplitudah šuma je linearni približek slabši, saj je signal (morda) izven linearnega območja) in stopnje nelinearnosti. Zaradi pogreška, ki ga v sistem vnaša linearizacija, se lahko konvergenca filtra poslabša ali pa ocena sploh ne konvergira k pravi rešitvi.

Nelinearni sistem lahko zapišemo v splošni obliki

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}\right) \\ \boldsymbol{z}_k &= \boldsymbol{h}\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k \end{aligned} \tag{6.28}$$

kjer se lahko šum $\boldsymbol{w}_k$ pojavi na vhodu sistema ali pa vpliva neposredno na stanja.

Razširjeni Kalmanov filter za nelinearni sistem (6.28) je podan s predikcijskim korakom

$$\begin{aligned} \hat{\boldsymbol{x}}_{k|k-1} &= \boldsymbol{f}\left(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}\right) \\ \boldsymbol{P}_{k|k-1} &= \boldsymbol{A}\boldsymbol{P}_{k-1|k-1}\boldsymbol{A}^T + \boldsymbol{F}\boldsymbol{Q}_{k-1}\boldsymbol{F}^T \end{aligned} \tag{6.29}$$

in korekcijskim korakom

$$\begin{aligned} \boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T\left(\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R}_k\right)^{-1} \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k) \\ \boldsymbol{P}_{k|k} &= \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{C}\boldsymbol{P}_{k|k-1} \end{aligned} \tag{6.30}$$

V predikcijskem koraku (6.29) uporabimo nelinearni model prehajanja stanj za izračun ocene predikcije stanja. Za izračun kovariančne matrike šuma iz

modela stanj (6.28) določimo Jacobijevo matriko $\boldsymbol{A}$, ki opisuje prehajanje šuma iz prejšnjih na trenutna stanja, in Jacobijevo matriko $\boldsymbol{F}$, ki opisuje širjenje šuma od vhodov na stanja

$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}}{\partial \hat{\boldsymbol{x}}}\right|_{\left(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}\right)}$$
$$\boldsymbol{F} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}\right|_{\left(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}\right)}$$

V korekcijskem koraku (6.30) določimo oceno meritve na podlagi predikcijske ocene stanja $\hat{\boldsymbol{z}}_k = \boldsymbol{h}\left(\hat{\boldsymbol{x}}_{k|k-1}\right)$. Določimo tudi Jacobijevo matriko $\boldsymbol{C}$, ki opisuje širjenje šuma iz stanj na izhode (meritve)

$$\boldsymbol{C} = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\left(\hat{\boldsymbol{x}}_{k|k-1}\right)} \tag{6.31}$$

Kovariančni matriki šuma sta $\boldsymbol{Q}_k = \mathrm{E}\{\boldsymbol{w}(k)\boldsymbol{w}^T(k)\}$ in $\boldsymbol{R}_k = \mathrm{E}\{\boldsymbol{v}(k)\boldsymbol{v}^T(k)\}$. V mnogih aplikacijah so uporabili razširjeni Kalmanov filter za reševanje problema lokalizacije kolesnih mobilnih robotov [7, 8] in gradnje zemljevida [9].

Primer 6.20

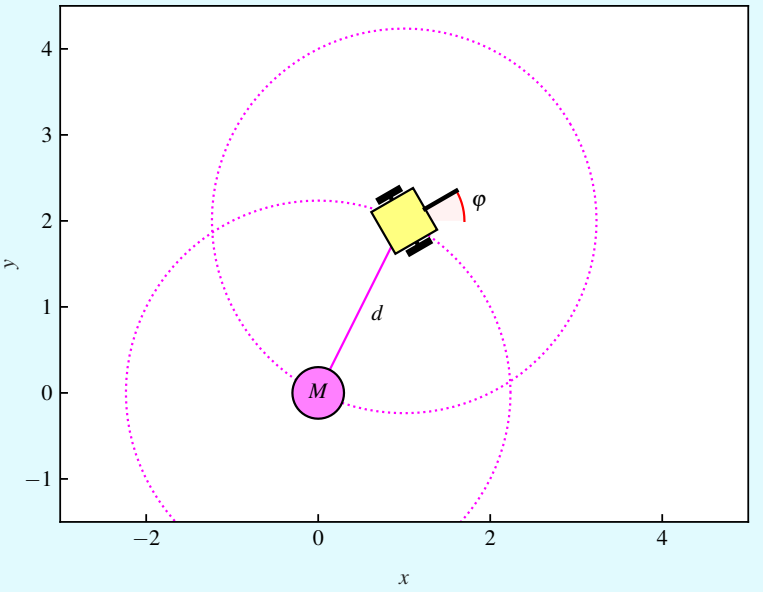
Kolesni mobilni robot z diferencialnim pogonom se premika po ravnini. Vhoda sta translatorna hitrost v_k in kotna hitrost ω_k, ki sta motena z Gaussovim šumom z variancama $\mathrm{var}\{v_k\} = 0{,}1\,\mathrm{m}^2/\mathrm{s}^2$ in $\mathrm{var}\{\omega_k\} = 0{,}1\,\mathrm{rad}^2/\mathrm{s}^2$.

Mobilni robot ima senzor, s katerim lahko izmeri razdaljo do značke, ki se nahaja v izhodišču globalnega koordinatnega sistema. Robot je opremljen tudi s kompasom, ki omogoča merjenje orientacije mobilnega robota (odklon). Predstavljena situacija je prikazana na sliki 6.23. Meritev razdalje je motena z Gaussovim šumom z varianco $0{,}5\,\mathrm{m}^2$, meritev kota pa z Gaussovim šumom z varianco $0{,}3\,\mathrm{rad}^2$.

Na začetku opazovanja je prava lega robota $\boldsymbol{x}_0 = [1,\ 2,\ \pi/6]^T$, ocenjena lega robota pa je $\hat{\boldsymbol{x}}_0 = [3,\ 0,\ 0]^T$ z začetno varianco

$$\boldsymbol{P}_0 = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0{,}6 \end{bmatrix}$$

Kakšen je časovni potek ocene lege mobilnega robota ($\hat{\boldsymbol{x}}_{k|k-1} = [x_k,\ y_k,\ \varphi_k]^T$) in varianca ocene, če ob vsakem času vzorčenja $T_s = 0{,}1\,\mathrm{s}$ pošljemo robotu ukaz $\boldsymbol{u} = [v_k,\ \omega_k]^T = [0{,}5,\ 0{,}5]^T$?



Slika 6.23: Postavitev, predstavljena v primeru 6.20. Mobilni robot ima senzor za merjenje razdalje do značke M (nameščena v izhodišču globalnega koordinatnega sistema) in kompas za določitev orientacije robota v globalnem koordinatnem sistemu (odklon).

Rešitev

Do rešitve lahko pridemo s pomočjo simulacije v okolju Matlab. Določimo model premikanja mobilnega robota. V tem primeru je kinematični model nelinearen

$$\hat{\boldsymbol{x}}_{k|k-1} = \hat{\boldsymbol{x}}_{k-1|k-1} + \begin{bmatrix} T_s v_{k-1} \cos(\varphi_{k-1}) \\ T_s v_{k-1} \sin(\varphi_{k-1}) \\ T_s \omega_{k-1} \end{bmatrix}$$

Model meritve razdalje in kota pa je

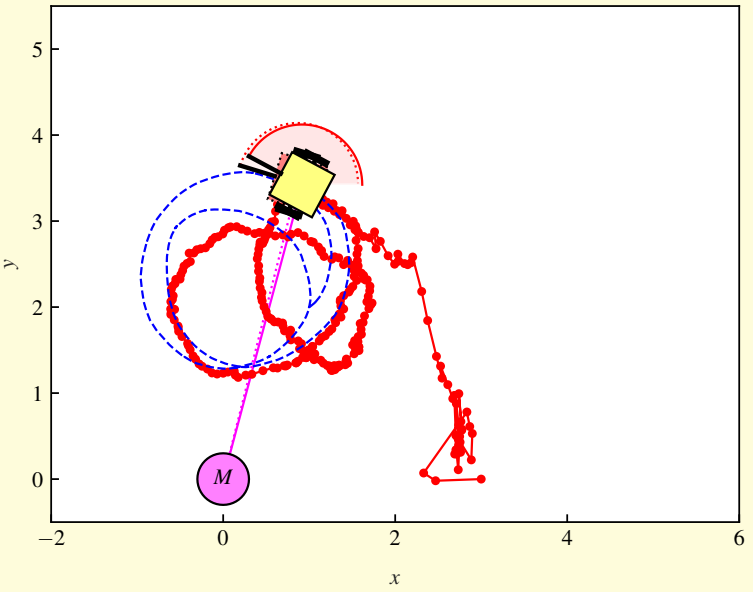
$$\hat{\boldsymbol{z}}_k = \begin{bmatrix} \sqrt{x_k^2 + y_k^2} \\ \varphi_k \end{bmatrix}$$

Matlab koda rešitve je predstavljena v programu 6.11. Rezultati simulacije so prikazani na slikah 6.24 – 6.28. Rezultati potrjujejo, da ocenjena lega konvergira k pravi legi mobilnega robota, čeprav je bila prvotna ocena pristranska. Členi inovacije se ustalijo okoli ničle.

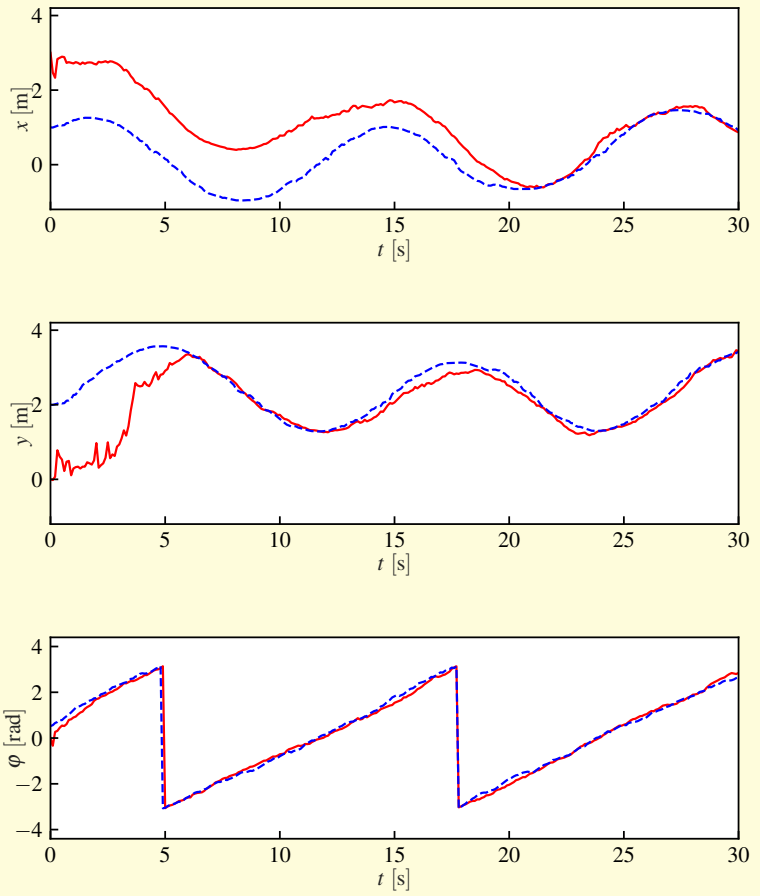
Program 6.11: Izvedba rešitve primera 6.20

./src/prb/example_ekf1default.m

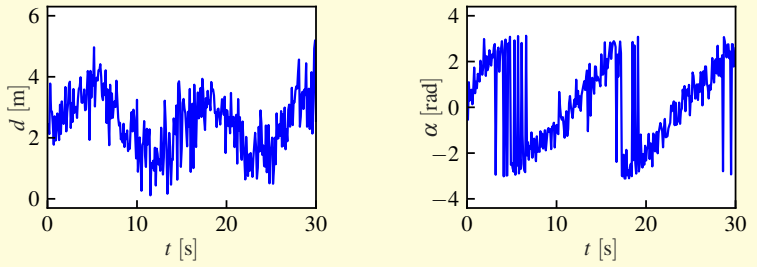
```
Ts = 0.1; % Računski korak
xTrue = [1; 2; pi/6]; % Prava začetna lega
x = [3; 0; 0]; % Ocena začetne lege
P = diag([9 9 0.6]); % Začetna kovariančna matrika ocene lege
Q = diag([0.1 0.1]); % Kovariančna matrika šuma modela gibanja
R = diag([0.5 0.3]); % Kovariančna matrika šuma merjenja razdalje in
                     % kota
enableNoise = 1; % Omogoči šum: 0 ali 1
N = 300; % Število simulacijskih korakov

% Zanka
for k = 1:N
    u = [0.5; 0.5]; % Ukazi (translatorna in kotna hitrost)
    uNoisy = u + sqrt(Q)*randn(2, 1)*enableNoise;

    % Simulacija pravih stanj (lege) robota
    xTrue = xTrue + Ts*[uNoisy(1)*cos(xTrue(3)); ...
                        uNoisy(1)*sin(xTrue(3)); ...
                        uNoisy(2)];
    xTrue(3) = wrapToPi(xTrue(3));

    % Simulacija meritev s šumom
    zTrue = [sqrt(xTrue(1)^2 + xTrue(2)^2 ); ...
             xTrue(3)] + sqrt(R)*randn(2, 1)*enableNoise;
    zTrue(1) = abs(zTrue(1));
    zTrue(2) = wrapToPi(zTrue(2));

    %%% Predikcija (ocena lege in hitrosti glede na znane vhode)
    xPred = x + Ts*[u(1)*cos(x(3)); ...
                    u(1)*sin(x(3)); ...
                    u(2)];
    xPred(3) = wrapToPi(xPred(3));

    % Jacobijeve matrike
    A = [1 0 -Ts*u(1)*sin(x(3)); ...
         0 1  Ts*u(1)*cos(x(3)); ...
         0 0  1];
    F = [Ts*cos(x(3)) 0; ...
         Ts*sin(x(3)) 0; ...
         0            Ts];
    PPred = A*P*A.' + F*Q*F.';

    % Ocenjene meritve
    z = [sqrt(xPred(1)^2 + xPred(2)^2); ...
         xPred(3)];

    %%% Korekcija
    d = sqrt(xPred(1)^2 + xPred(2)^2);
    C = [xPred(1)/d xPred(2)/d 0;...
         0          0          1];
    K = PPred*C.'/(C*PPred*C.' + R);
    inov = zTrue - z;
    inov(2) = wrapToPi(inov(2));
    x = xPred + K*inov;
    P = PPred - K*C*PPred;
end
```



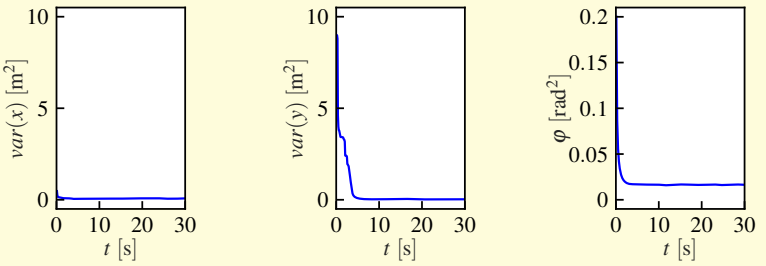
Slika 6.24: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija iz primera 6.20



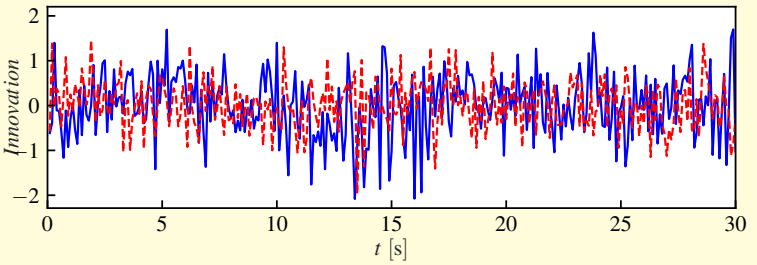
Slika 6.25: Ocenjena lega (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.20



Slika 6.26: Meritvi razdalje in kota iz primera 6.20

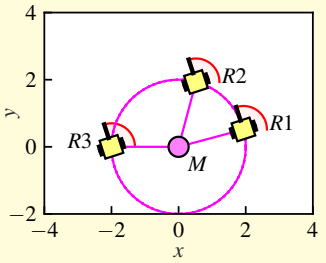


Slika 6.27: Variance ocene lege robota iz primera 6.20

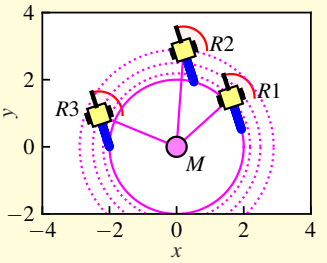


Slika 6.28: Časovni potek inovacije iz primera 6.20

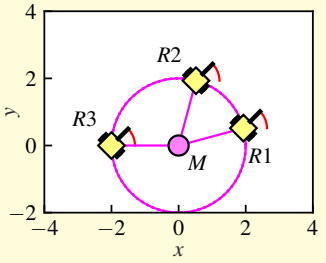
Če stanja sistema niso neposredno merljiva (kot v tem primeru), se pojavi vprašanje spoznavnosti sistema. Analiza spoznavnosti lahko običajno zagotovi zadostne pogoje za spoznavnost sistema. Vendar moramo analizo opraviti pred načrtovanjem ocenjevanja stanj, saj nam to lahko pomaga pri izbiri ustrezne množice merilnih signalov, ki omogočajo ocenjevanje. Spoznavnost sistema lahko preverimo z uporabo naprednih matematičnih orodij ali pa izberemo preprost grafični pristop, ki temelji na definiciji nerazpoznavnih stanj (glejte poglavje 6.3.3). Analiza spoznavnosti sistema iz tega primera je prikazana na sliki 6.29, iz katere je razvidno, da so stanja sistema običajno razpoznavna, razen v nekaterih posebnih primerih (slika 6.29d). Če stanja sistema opazujemo dovolj dolgo in so regulirne veličine mobilnega robota ustrezno vzbujene, je sistem spoznaven.



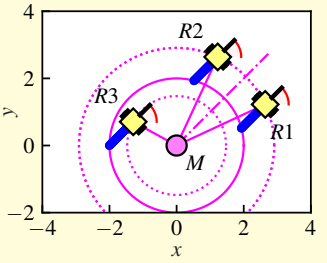
(a) Tri začetne lege robotov, ki imajo enako smer in oddaljenost od značke



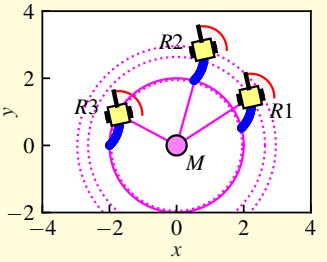
(b) Posodobljena situacija s slike (a) po tem, ko so roboti prepotovali enako razdaljo v smeri naprej. Z vidika meritev so lege robotov razpoznavne.



(c) Tri začetne lege robotov, ki imajo enako smer in oddaljenost od značke – poseben primer



(d) Posodobljena situacija s slike (c) po tem, ko so roboti prepotovali enako razdaljo v smeri naprej. Z vidika meritev je robot v legi 1 nerazpoznaven od robota v legi 2, robot v legi 3 pa je razpoznaven od ostalih dveh.

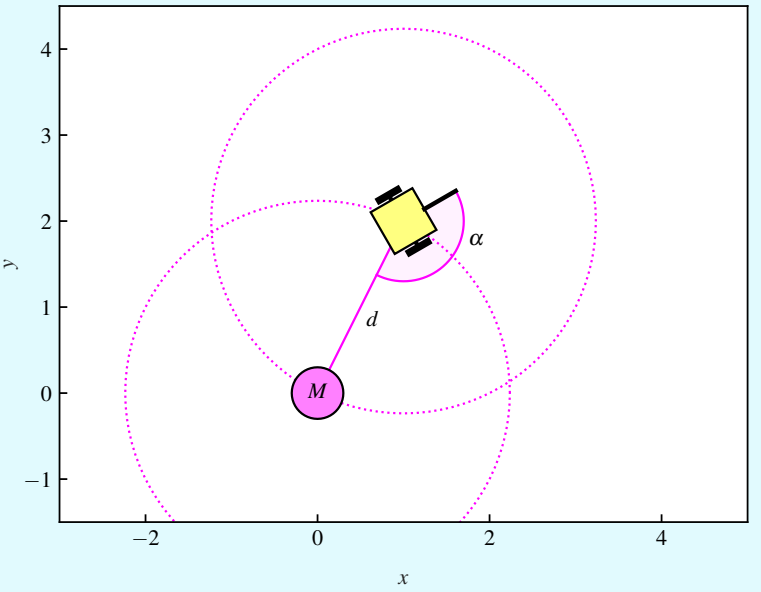


(e) Posodobljena situacija s slike (c) po tem, ko so roboti prepotovali isto neravno pot. Z vidika meritev so vsi roboti razpoznavni.

Slika 6.29: Analiza spoznavnosti sistema iz primera 6.20

Primer 6.21

Mobilni robot iz primera 6.20 naj ima malce drugačen senzor, ki meri razdaljo in kot do značke v izhodišču globalnega koordinatnega sistema. Meritve kota so znotraj intervala $\alpha \in [-\pi,\ \pi]$. Meritev razdalje je motena z Gaussovim šumom z varianco $0{,}5\,\mathrm{m}^2$, meritev kota pa z Gaussovim šumom z varianco $0{,}3\,\mathrm{rad}^2$.



Slika 6.30: Postavitev iz primera 6.21. Mobilni robot ima senzor za merjenje razdalje in kota do značke M, ki se nahaja v izhodišču globalnega koordinatnega sistema.

Kakšen je časovni potek ocene lege robota ($\hat{\boldsymbol{x}}_{k|k-1} = [x_k,\ y_k,\ \varphi_k]^T$) in varianca ocene, če ob vsakem času vzorčenja $T_s = 0{,}1\,\mathrm{s}$ pošljemo robotu ukaz $\boldsymbol{u} = [v_k,\ \omega_k]^T = [0{,}5,\ 0{,}5]^T$?

Rešitev

Do rešitve lahko pridemo s pomočjo simulacije v okolju Matlab. Določimo model gibanja mobilnega robota. Kinematični model kolesnega mobilnega robota je enak kot v primeru 6.20

$$\hat{\boldsymbol{x}}_{k|k-1} = \hat{\boldsymbol{x}}_{k-1|k-1} + \begin{bmatrix} T_s v_{k-1} \cos(\varphi_{k-1}) \\ T_s v_{k-1} \sin(\varphi_{k-1}) \\ T_s \omega_{k-1} \end{bmatrix}$$

Model meritve razdalje in kota je

$$\hat{\boldsymbol{z}}_k = \begin{bmatrix} \sqrt{x_k^2 + y_k^2} \\ \operatorname{atan2}(0 - y_k, 0 - x_k) - \varphi_k \end{bmatrix}$$

pri čemer je štirikvadratna inverzna funkcija tangens definirana v (2.11).

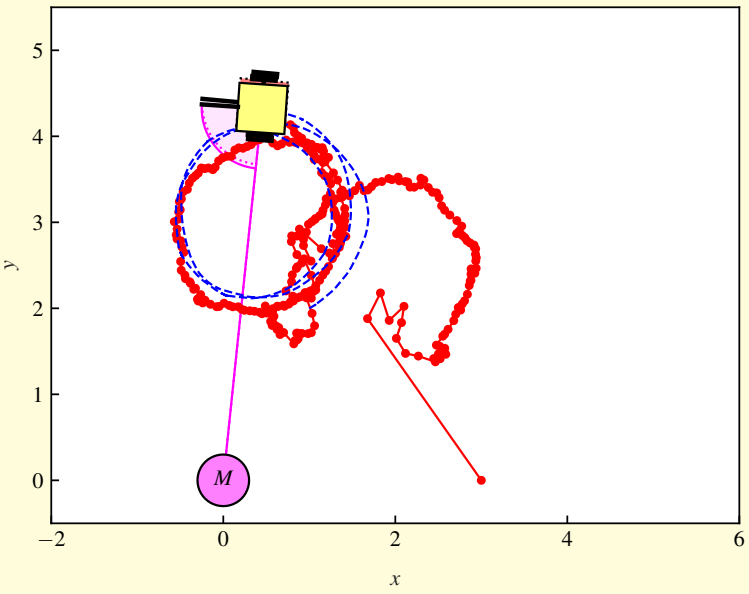
Matlab koda rešitve je podana v programu 6.12. Rezultati simulacije, prikazani na slikah 6.31 – 6.35, kažejo, da ocenjena stanja konvergirajo k pravi legi robota. Čeprav je ta primer podoben primeru 6.20, se lahko zgodi, da ocena ne konvergira k pravi legi robota, če so okoljski pogoji nekoliko drugačni. Primer konvergence k napačni rešitvi prikazujejo slike 6.36 – 6.40. Ker sta oba izhoda senzorja (razdalja in kot) relativni meritvi, ocenjena stanja morda ne konvergirajo k pravi rešitvi, čeprav je inovacija (razlika med meritvijo in predikcijo meritve) blizu ničelne vrednosti (slika 6.40).

Program 6.12: Izvedba rešitve iz primera 6.21

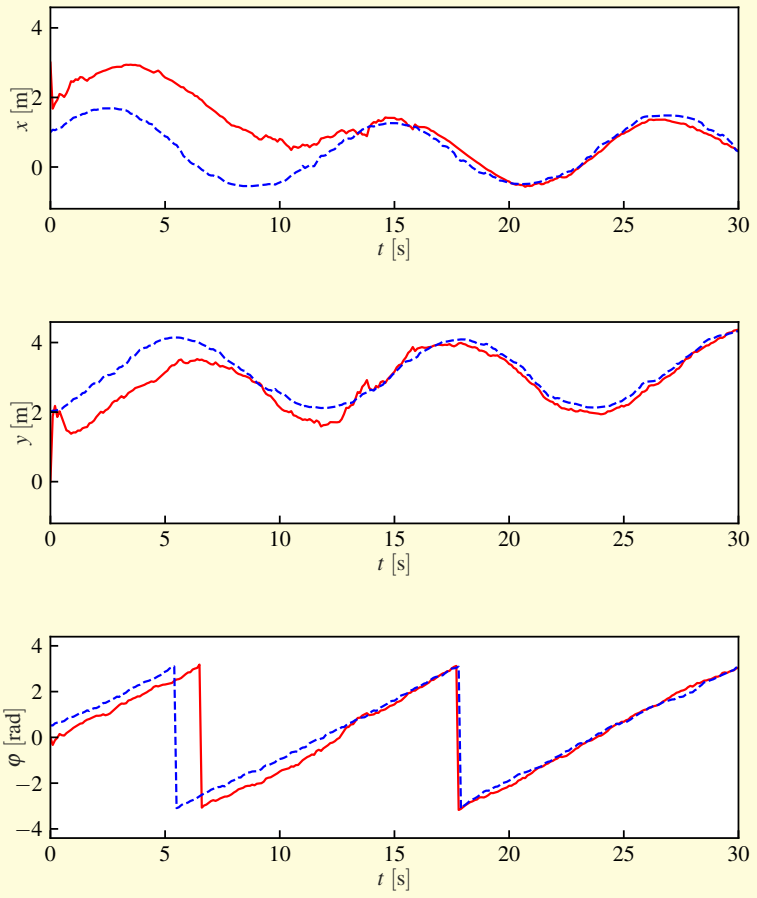
./src/prb/example_ekf2default.m

```
Ts = 0.1; % Računski korak
xTrue = [1; 2; pi/6]; % Prava začetna lega
x = [3; 0; 0]; % Ocena začetne lege
P = diag([9 9 0.6]); % Začetna kovariančna matrika ocene lege
Q = diag([0.1 0.1]); % Kovariančna matrika šuma modela gibanja
R = diag([0.5 0.3]); % Kovariančna matrika šuma merjenja razdalje in
                     % kota
enableNoise = 1; % Omogoči šum: 0 ali 1
N = 300; % Število simulacijskih korakov

% Loop
for k = 1:N
    u = [0.5; 0.5]; % Ukazi (translatorna in kotna hitrost)
    uNoisy = u + sqrt(Q)*randn(2, 1)*enableNoise;

    % Simulacija pravih stanj (lege) robota
    xTrue = xTrue + Ts*[uNoisy(1)*cos(xTrue(3)); ...
                        uNoisy(1)*sin(xTrue(3)); ...
                        uNoisy(2)];
    xTrue(3) = wrapToPi(xTrue(3));

    % Simulacija meritev s šumom (razdalja in kot)
    zTrue = [sqrt(xTrue(1)^2 + xTrue(2)^2); ...
             atan2(0-xTrue(2), 0-xTrue(1))-xTrue(3)] + ...
             sqrt(R)*randn(2, 1)*enableNoise;
    zTrue(1) = abs(zTrue(1));
    zTrue(2) = wrapToPi(zTrue(2));

    %%% Predikcija (ocena lege in hitrosti glede na znane vhode)
    xPred = x + Ts*[u(1)*cos(x(3)); ...
                    u(1)*sin(x(3)); ...
                    u(2)];
    xPred(3) = wrapToPi(xPred(3));

    % Jacobijeve matrike
    A = [1 0 -Ts*u(1)*sin(x(3)); ...
         0 1  Ts*u(1)*cos(x(3)); ...
         0 0  1];
```

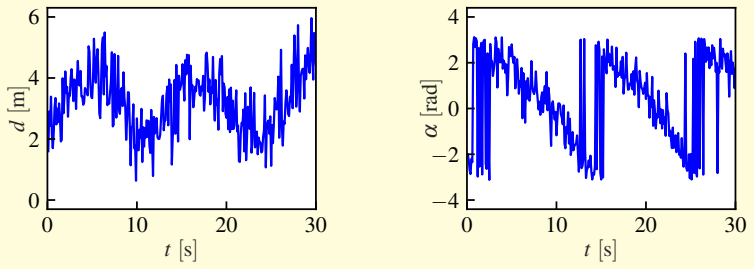
```
39      F = [Ts*cos(x(3)) 0; ...
40           Ts*sin(x(3)) 0; ...
41           0            Ts];
42      PPred = A*P*A.' + F*Q*F.';
43
44
45      % Ocenjene meritve
46      z= [sqrt(xPred(1)^2 + xPred(2)^2); ...
47          atan2(0-xPred(2), 0-xPred(1)) - xPred(3)];
48      z(2) = wrapToPi(z(2));
49
50      %%% Korekcija
51      d = sqrt(xPred(1)^2 + xPred(2)^2);
52      C = [xPred(1)/d   xPred(2)/d    0; ...
53          -xPred(2)/d^2 xPred(1)/d^2 -1];
54      K = PPred*C.'/(C*PPred*C.' + R);
55      inov = zTrue - z;
56
57      % Izbira primerne inovacije, zaradi šuma in cikličnosti kota
58      inov(2) = wrapToPi(inov(2));
59
60      x = xPred + K*inov;
61      P = PPred - K*C*PPred;
62  end
```



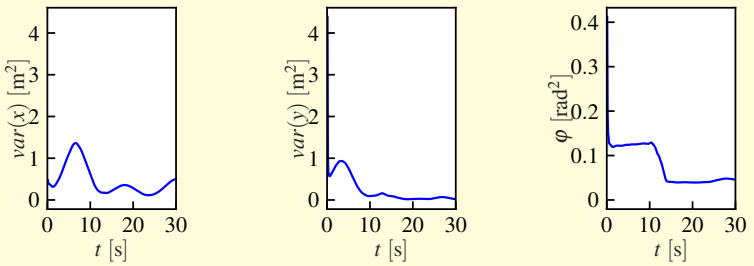
Slika 6.31: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija iz primera 6.21



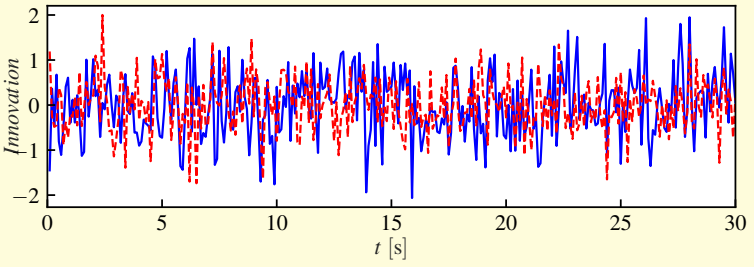
Slika 6.32: Ocena lege (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.21



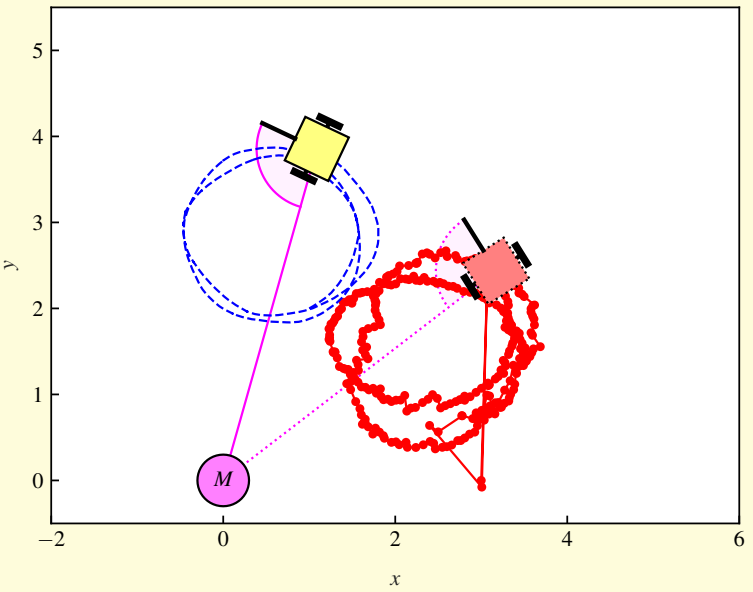
Slika 6.33: Meritvi razdalje in kota iz primera 6.21



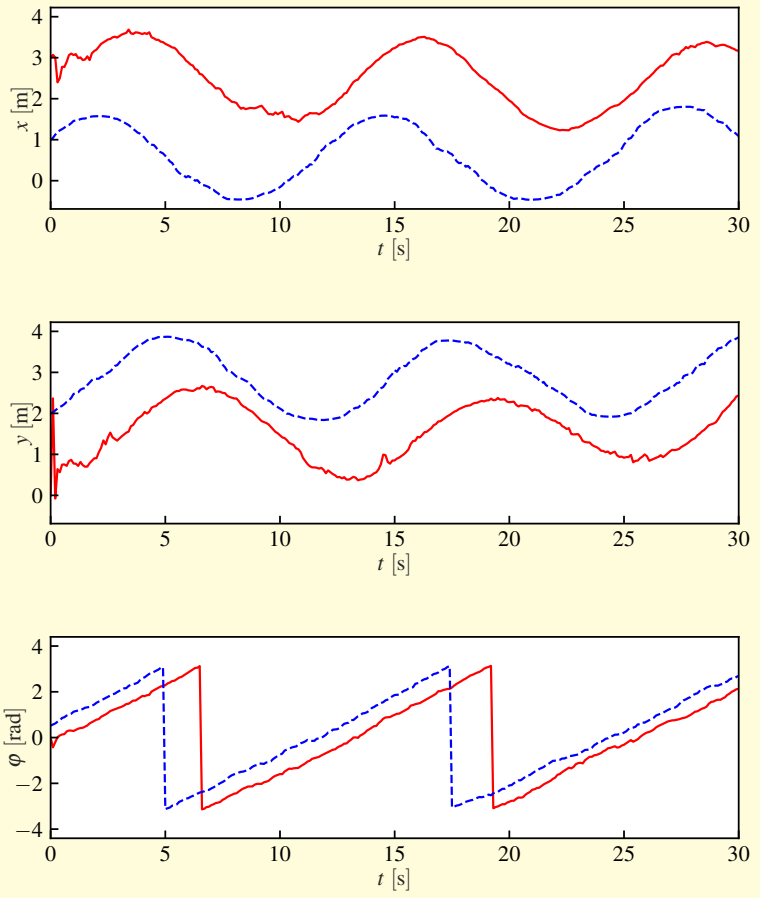
Slika 6.34: Variance ocene lege robota iz primera 6.21



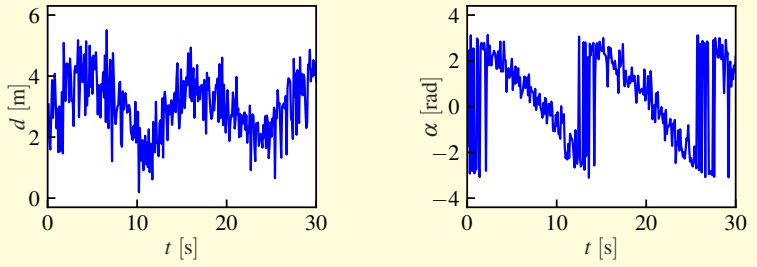
Slika 6.35: Časovni potek inovacije iz primera 6.21



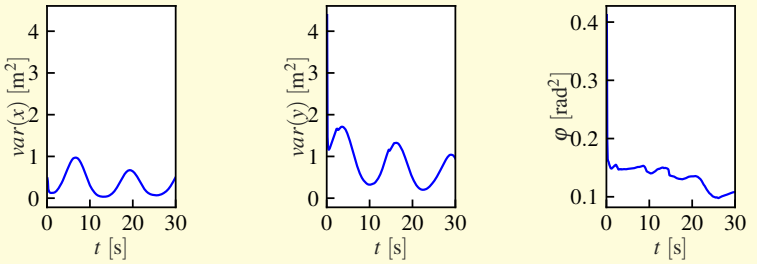
Slika 6.36: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija iz primera 6.21 (reprezentativni primer)



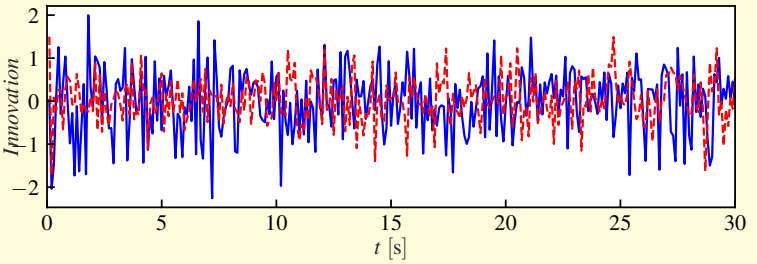
Slika 6.37: Ocena lege (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.21 (reprezentativni primer)



Slika 6.38: Meritvi razdalje in kota iz primera 6.21 (reprezentativni primer)

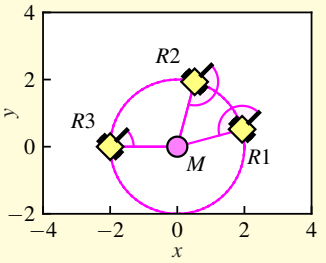


Slika 6.39: Variance ocene lege robota iz primera 6.21 (reprezentativni primer)

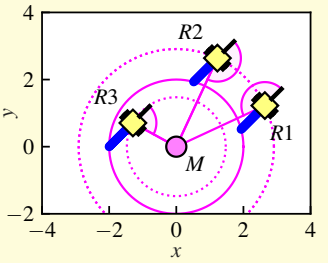


Slika 6.40: Časovni potek inovacije iz primera 6.21 (reprezentativni primer)

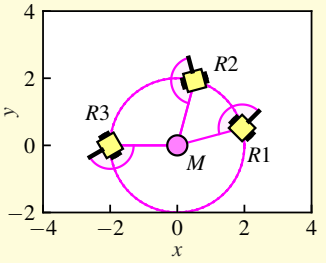
Grafični prikaz spoznavnosti na sliki 6.41 omogoča vpogled v razloge za pristranskost ocene. Analiza kaže, da obstajajo stanja, ki so z vidika meritev nerazpoznavna, ne glede na regulirne veličine. Poleg tega za vsako meritev obstaja neskončna množica stanj, zato sistem ni spoznaven. Pristranskost ocene lahko odpravimo s hkratnim opazovanjem več značk, saj nam to zagotovi dovolj informacij za izbiro ustrezne rešitve, kot je prikazano v primeru 6.22.



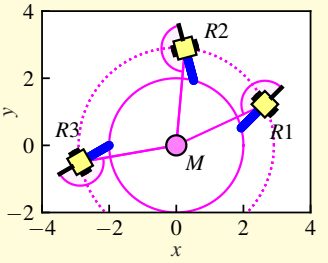
(a) Tri začetne lege robotov z enako oddaljenostjo od značke



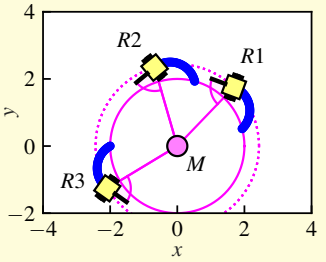
(b) Posodobljena situacija s slike (a) po tem, ko so roboti prepotovali enako razdaljo v smeri naprej. Z vidika meritev so lege robotov razpoznavne.



(c) Tri začetne lege robotov, ki imajo enak kot in oddaljenost do značke — poseben primer



(d) Posodobljena situacija s slike (c) po tem, ko so roboti prepotovali enako razdaljo v smeri naprej. Z vidika meritev so vse tri lege robotov nerazpoznavne.

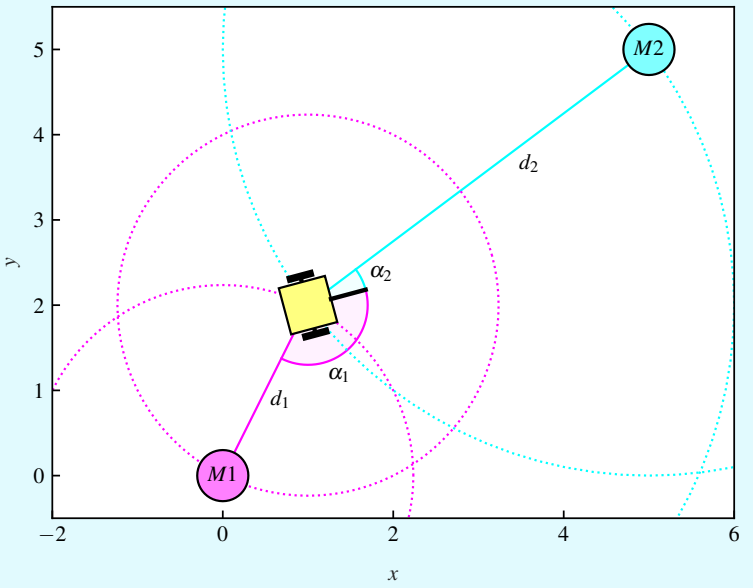


(e) Posodobljena situacija s slike (c) po tem, ko so roboti prepotovali enako neravno pot. Z vidika meritev so vse tri lege robotov nerazpoznavne.

Slika 6.41: Analiza spoznavnosti sistema iz primera 6.21

Primer 6.22

Nadgradimo primer 6.21 tako, da lahko robot hkrati opazuje dve prostorsko ločeni znački, ki sta postavljeni na $x_{M1} = 0$, $y_{M1} = 0$ in $x_{M2} = 5$, $y_{M2} = 5$. Vsi ostali podatki so enaki kot v primeru 6.21.



Slika 6.42: Postavitev iz primera 6.22. Mobilni robot ima senzor za merjenje kota in oddaljenosti od značk $M1$ in $M2$.

Rešitev

Prilagoditi moramo le korekcijski del algoritma. Vektor meritve sedaj vsebuje štiri elemente

$$\hat{\boldsymbol{z}}_k = \begin{bmatrix} \sqrt{(x_{M1} - x_k)^2 + (y_{M1} - y_k)^2} \\ \operatorname{atan2}\left(y_{M1} - y_k, x_{M1} - x_k\right) - \varphi_k \\ \sqrt{(x_{M2} - x_k)^2 + (y_{M2} - y_k)^2} \\ \operatorname{atan2}\left(y_{M2} - y_k, x_{M2} - x_k\right) - \varphi_k \end{bmatrix}$$

razdaljo in kot do prve značke ter razdaljo in kot do druge značke. Določimo izhodno matriko $\boldsymbol{C}$ (glejte (6.31)) z linearizacijo okoli trenutne predikcijske ocene stanja $(x_k,\ y_k)$

$$\boldsymbol{C} = \begin{bmatrix} \frac{x_k}{d_1} & \frac{y_k}{d_1} & 0 \\ -\frac{y_k}{d_1^2} & \frac{x_k}{d_1^2} & -1 \\ \frac{x_k}{d_2} & \frac{y_k}{d_2} & 0 \\ -\frac{y_k}{d_2^2} & \frac{x_k}{d_2^2} & -1 \end{bmatrix}$$

kjer je $d_1 = \sqrt{(x_{M1} - x_k)^2 + (y_{M1} - y_k)^2}$ in $d_2 = \sqrt{(x_{M2} - x_k)^2 + (y_{M2} - y_k)^2}$. Določimo tudi kovariančno matriko šuma meritve kot

$$\boldsymbol{R} = \begin{bmatrix} 0{,}5 & 0 & 0 & 0 \\ 0 & 0{,}3 & 0 & 0 \\ 0 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0{,}3 \end{bmatrix}$$

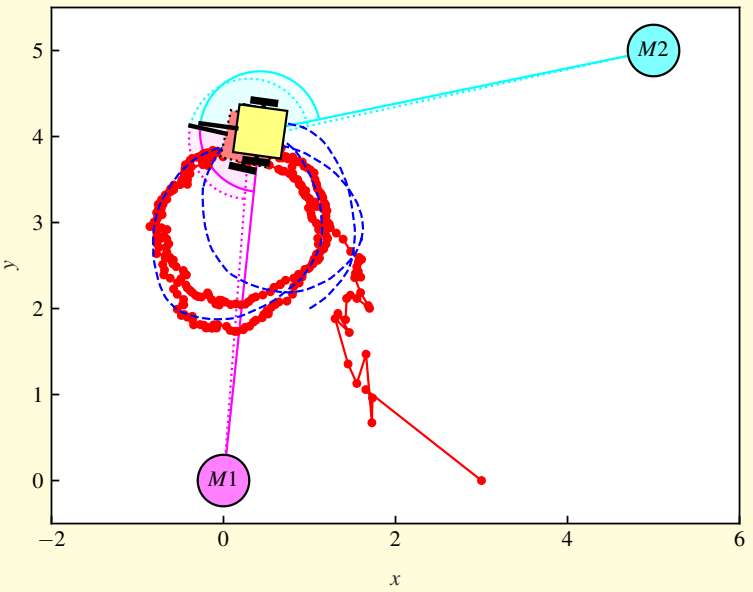
Matlab koda rešitve je podana v programu 6.13. Rezultati simulacije, predstavljeni na slikah 6.43 – 6.47, potrjujejo, da ocenjena stanja konvergirajo k pravi legi robota.

Program 6.13: Rešitev primera 6.22

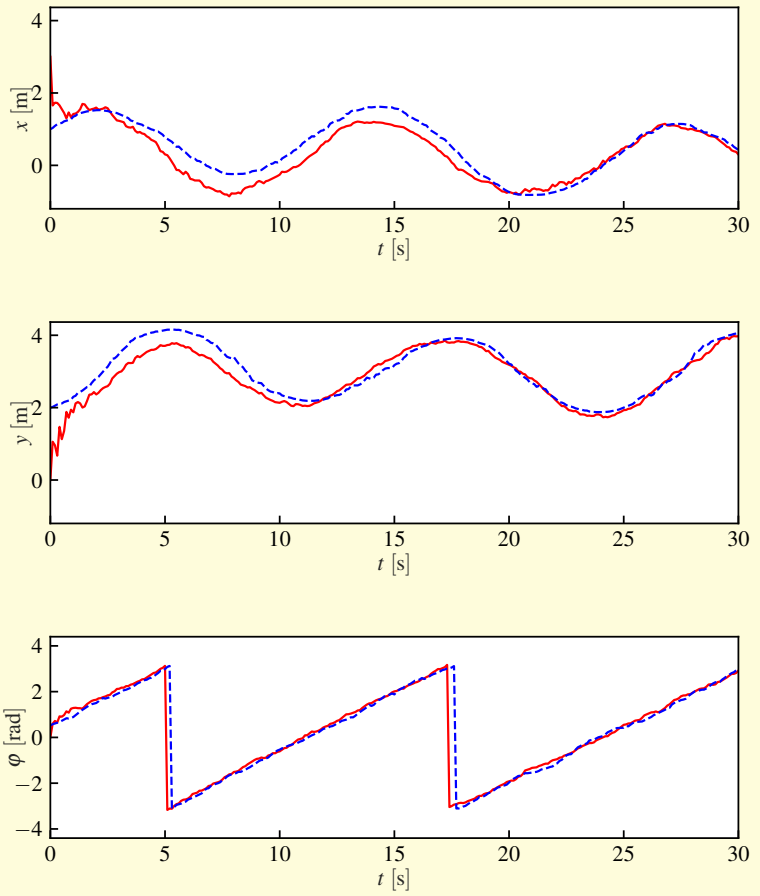
./src/prb/example_ekf3default.m

```
Ts = 0.1; % Računski korak
xTrue = [1; 2; pi/6]; % Prava začetna lega
x = [3; 0; 0]; % Ocena začetne lege
P = diag([9 9 0.6]); % Začetna kovariančna matrika ocene lege
Q = diag([0.1 0.1]); % Kovariančna matrika šuma modela gibanja
R = diag([0.5 0.3]); % Kovariančna matrika šuma merjenja razdalje in
                     % kota
enableNoise = 1; % Omogoči šum: 0 ali 1
N = 300; % Število simulacijskih korakov
marker = [0 0; 5 5]; % Položaji značk

% Zanka
for k = 1:N
    u = [0.5; 0.5]; % Ukazi (translatorna in kotna hitrost)
    uNoisy = u + sqrt(Q)*randn(2, 1)*enableNoise;

    % Simulacija pravih stanj (lege) robota
    xTrue = xTrue + Ts*[uNoisy(1)*cos(xTrue(3)); ...
                        uNoisy(1)*sin(xTrue(3)); ...
                        uNoisy(2)];
    xTrue(3) = wrapToPi(xTrue(3));

    % Simulacija meritev s šumom (razdalja in kot)
    zTrue = [];
    for m = 1:size(marker, 1)
        dist = sqrt((marker(m,1)-xTrue(1))^2 + (marker(m,2)-xTrue(2))^2);
        alpha = atan2(marker(m,2)-xTrue(2), marker(m,1)-xTrue(1))-xTrue(3);
        zz = [dist; alpha] + sqrt(R)*randn(2, 1)*enableNoise;
        zz(1) = abs(zz(1));
        zz(2) = wrapToPi(zz(2));
        zTrue = [zTrue; zz];
    end

    %%% Predikcija (ocena lege in hitrosti glede na znane vhode)
    xPred = x + Ts*[u(1)*cos(x(3)); ...
                    u(1)*sin(x(3)); ...
                    u(2)];
    xPred(3) = wrapToPi(xPred(3));

    % Jacobijeve matrike
    A = [1 0 -Ts*u(1)*sin(x(3)); ...
         0 1  Ts*u(1)*cos(x(3)); ...
```

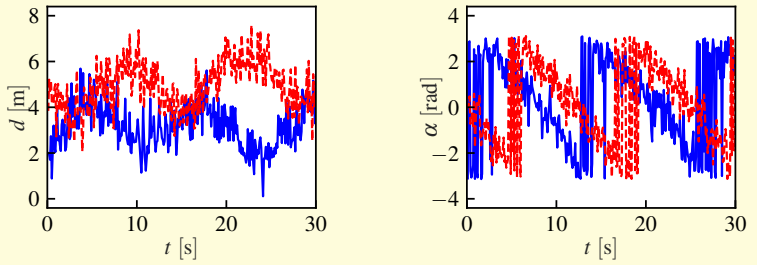
```
43          0 0  1];
44      F = [Ts*cos(x(3)) 0; ...
45           Ts*sin(x(3)) 0; ...
46           0            Ts];
47      PPred = A*P*A.' + F*Q*F.';
48
49      %%% Korekcija
50      z = [];
51      C = [];
52      for m = 1:size(marker,1)
53          dist = sqrt((marker(m,1)-xPred(1))^2 + (marker(m,2)-xPred(2))^2);
54          alpha = atan2(marker(m,2)-xPred(2), marker(m,1)-xPred(1))-xPred(3);
55          zz = [dist; alpha];
56          zz(2) = wrapToPi(zz(2));
57          z = [z; zz];
58
59          % Matrika C za korekcijo
60          c = [xPred(1)/dist   xPred(2)/dist    0; ...
61              -xPred(2)/dist^2 xPred(1)/dist^2 -1];
62          C = [C; c];
63      end
64
65      % Kovariančna matrika meritev
66      RR = diag(repmat([R(1,1) R(2,2)], 1, size(marker, 1)));
67      K = PPred*C.'/(C*PPred*C.' + RR);
68
69      inov = zTrue - z;
70      % Izbira primerne inovacije, zaradi šuma in cikličnosti kota
71      for m = 1:size(marker, 1)
72          inov(2*m) = wrapToPi(inov(2*m));
73      end
74
75      x = xPred + K*(inov);
76      P = PPred - K*C*PPred;
77  end
```



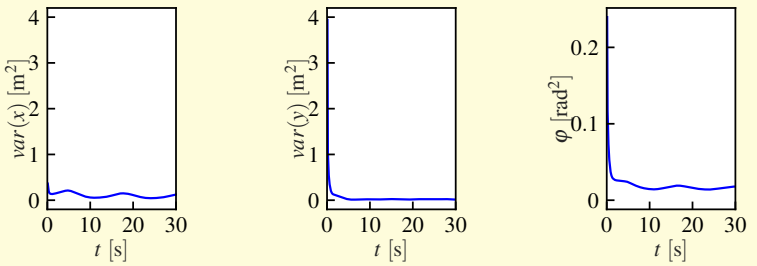
Slika 6.43: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija iz primera 6.22



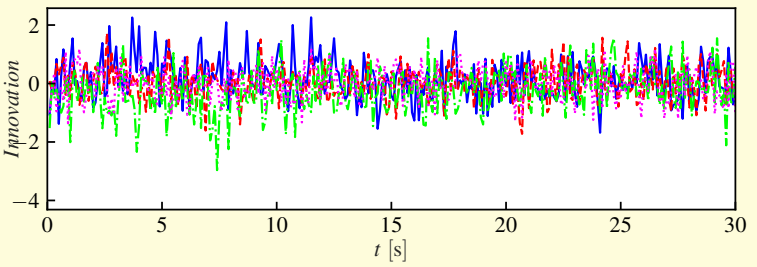
Slika 6.44: Ocena lege (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.22



Slika 6.45: Meritve razdalje in kota iz primera 6.22

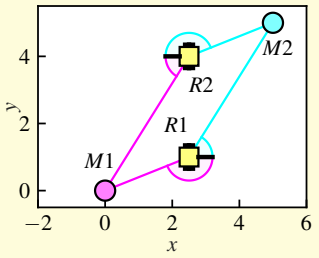


Slika 6.46: Variance ocene lege robota iz primera 6.22

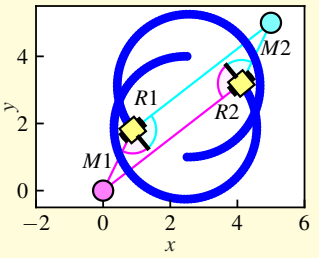


Slika 6.47: Časovni potek inovacije iz primera 6.22

Ponovno lahko grafično preverimo spoznavnost sistema. Tudi v tem primeru lahko najdemo posebno situacijo, kjer so vsa stanja nerazpoznavna, kar prikazuje slika 6.48. Na prvi pogled so obravnavana stanja z vidika meritev nerazpoznavna, ker pa predpostavljamo, da merilni podatki vsebujejo tudi oznako (ID) značk, ima ta posebna situacija razpoznavna stanja in sistem je spoznaven.



(a) Dve posebni legi, ki sta z vidika meritev simetrični



(b) Posodobljena situacija s slike (a) po enakih prepotovanih relativnih poteh glede na obe začetni legi. Z vidika meritev sta obe legi razpoznavni, če merilni podatki vsebujejo oznako (ID) značk.

Slika 6.48: Analiza spoznavnosti sistema iz primera 6.22

6.5.3 Druge različice Kalmanovega filtra

Poleg Kalmanovega filtra za linearne sisteme in razširjenega Kalmanovega filtra za nelinearne sisteme [10] obstaja vrsta izpeljank.

Nepristranski Kalmanov filter (UKF, angl. *unscented Kalman filter*) se običajno uporablja v sistemih z izrazito nelinearnostjo, kjer razširjeni Kalmanov filter morda ne zagotavlja zadovoljivih rezultatov. V tem primeru se kovariančne matrike statistično ocenijo na podlagi manjše množice vhodnih točk, ki se preslikajo preko nelinearne funkcije ter se nato uporabijo za oceno srednje vrednosti in kovariančne matrike. Te točke, znane kot *sigma točke*, so razpršene okoli ocenjene vrednosti po nekem algoritmu (običajno je $2n + 1$ točk za n dimenzij).

Informacijski filter uporablja informacijsko matriko in informacijski vektor namesto kovariančne matrike in ocene stanj. Informacijska matrika predstavlja inverz kovariančne matrike, informacijski vektor pa je produkt informacijske matrike in ocene vektorja stanj. Informacijski filter je dualen Kalmanovemu filtru, kjer je korekcijski korak računsko bistveno enostavnejši (le matrična vsota), vendar je predikcijski korak računsko bolj zahteven.

Kalman-Bucyjev filter je oblika Kalmanovega filtra za zvezne sisteme.

6.6 Filter delcev

Do zdaj smo uporabljali Bayesov filter za sisteme z diskretnim prostorom stanj s končnim številom vrednosti, ki jih spremenljivke stanj lahko zavzamejo. Če želimo uporabiti Bayesov filter za zvezne spremenljivke, lahko te spremenljivke kvantiziramo na končno število vrednosti. Tovrstna izvedba Bayesovega filtra za zvezne spremenljivke stanj je splošno znana kot *histogramski filter*.

Za zvezne spremenljivke stanj in eksplicitno rešitev Bayesovega filtra (6.21) moramo rešiti enačbo

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{0:k-1}) = \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k)}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{0:k-1})} \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{0:k-2}) \,\mathrm{d}\boldsymbol{x}_{k-1} \tag{6.32}$$

kjer smo predpostavili, da so stanja vsebovana in da obravnavamo Markovov proces (glejte poglavje 6.2). Trenutna porazdelitev stanj (6.32) je potrebna za izračun najverjetnejše ocene stanj (matematično upanje)

$$\mathrm{E}\{\hat{\boldsymbol{x}}_{k|k}\} = \int \boldsymbol{x}_{k|k} \cdot p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{0:k-1}) \,\mathrm{d}\boldsymbol{x}_{k|k}$$

Eksplicitna rešitev (6.32) je možna le za omejen nabor primerov, kjer predpostavimo Gaussovo porazdelitev in linearnost sistema, kar vodi v Kalmanov filter. V primeru nelinearnih sistemov lahko nelinearnost sistema (v modelu

gibanja, aktuatorja in/ali senzorja) lineariziramo, kar nas privede do razširjenega Kalmanovega filtra.

Filter delcev (angl. *particle filter*) je bolj splošen pristop, kjer nista potrebna Gaussova porazdelitev in linearnost sistema. Osnovna ideja je, da trenutno oceno porazdelitve stanj (6.21) po opravljeni meritvi aproksimiramo z množico N delcev. Vsak delec v množici predstavlja vrednost ocenjenega stanja $\boldsymbol{x}_k^i$, ki je naključno vzorčena iz porazdelitve (simulacija Monte Carlo). Vsak delec podaja svojo hipotezo o dejanskem stanju sistema. Porazdelitev je opisana s pomočjo množice naključno generiranih delcev, torej gre za neparametrični opis porazdelitve, ki ni omejen samo na Gaussove porazdelitve. Opis porazdelitve z delci omogoča modeliranje nelinearnih transformacij šuma (model aktuatorja in/ali senzorja). Torej lahko z delci opišemo porazdelitev šuma, ki se iz vhodov ali izhodov sistema prenaša preko nelinearnih funkcij na stanja sistema.

Algoritem 5 predstavlja osnoven princip filtra delcev.

Algorithm 5 Algoritem za filter delcev. Funkcijo `Filter_delcev` kličemo v vsakem časovnem trenutku s trenutnimi vhodi in meritvami ter predhodno oceno stanja.

---

function FILTER_DELCEV($\hat{x}_{k-1|k-1}$, u_{k-1}, z_k)

Inicializacija:

if $k > 0$ **then**

Inicializacija množice N delcev $\boldsymbol{x}_k^i$ na osnovi naključnega vzorčenja porazdelitve $p(\boldsymbol{x}_0)$.

end if

Predikcija:

Transformiraj (premakni) vsak delec $\hat{\boldsymbol{x}}_{k-1|k-1}^i$ na osnovi modela premika in znanega vhoda $\boldsymbol{u}_{k-1}$, kateremu dodaj naključno vrednost glede na lastnosti šuma, ki je del modela premika. Model premika podaja $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1})$. Dobljena predikcija je množica delcev $\hat{\boldsymbol{x}}_{k|k-1}^i$.

Korekcija:

Za vsak delec $\hat{\boldsymbol{x}}_{k|k-1}^i$ **oceni vrednost meritve**, ki bi jo sistem izmeril, če bi njegovo stanje ustrezalo stanju delca.

Glede na opravljeno meritev in primerjavo z ocenjenimi meritvami delcev **oceni pomembnost delcev**.

Nato **določi nov nabor delcev** glede na njihovo pomembnost z naključnim izborom delcev z verjetnostjo, ki je proporcionalna njihovi pomembnosti, torej $p(\boldsymbol{z}_k|\hat{\boldsymbol{x}}_{k|k-1}^i)$. Bolj verjetni delci so izbrani večkrat, manj verjetni delci pa manjkrat.

Ocena filtriranega stanja $\hat{\boldsymbol{x}}_{k|k}$ je enaka povprečni vrednosti vseh delcev.

end function

V začetnem koraku moramo določiti začetno populacijo delcev, $\hat{\boldsymbol{x}}_0^i$ za $i \in 1, \ldots, N$,

katere raztros je odvisen od zaupanja (porazdelitve) $p(\boldsymbol{x}_0)$ v začetno stanje sistema. V kolikor začetno stanje ni znano, je porazdelitev uniformna in so delci enakomerno (enako verjetno) razporejeni po celotnem prostoru stanj.

V predikcijskem koraku izračunamo novo stanje za vsak delec glede na podan vhod sistema. Dobljenim ocenam stanj za vsak delec dodamo naključno vrednost šuma, ki ga pričakujemo na vhodu sistema. Tako dobimo predikcijo stanja za vsak delec $\hat{\boldsymbol{x}}^i_{k|k-1}$. Dodani šum zagotavlja, da se delci razpršijo, saj to omogoča oceno prave vrednosti ob prisotnosti različnih motenj.

V korekcijskem koraku ovrednotimo pomembnost delcev tako, da za vsak delec (iz njegovih ocenjenih stanj) izračunamo odstopanje dejanske meritve $\boldsymbol{z}_k$ od ocenjene meritve delca $\hat{\boldsymbol{z}}^i_k$. Razlika med dejansko in ocenjeno meritvijo je splošno znana kot *inovacija* in se lahko oceni za vsak delec kot

$$\boldsymbol{innov}^i_k = \boldsymbol{z}_k - \hat{\boldsymbol{z}}^i_k$$

katere vrednost je manjša za bolj verjetne delce.

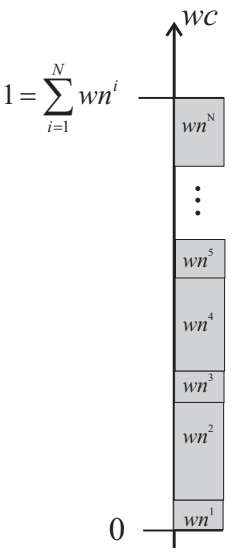
Na podlagi inovacije lahko določimo pomembnost vsakega delca oz. verjetnost $p(\boldsymbol{z}_k|\hat{\boldsymbol{x}}^i_{k|k-1})$, ki predstavlja utež w^i_k od i-tega delca. Utež lahko določimo z Gaussovo porazdelitvijo kot

$$w^i_k = \det(2\pi\boldsymbol{R})^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{innov}^i_k)^T \boldsymbol{R}^{-1}(\boldsymbol{innov}^i_k)}$$

kjer je $\boldsymbol{R}$ kovariančna matrika meritve.

Zelo pomemben korak v procesu uporabe filtra delcev je **vzorčenje po pomembnosti** (angl. *importance sampling*) glede na uteži w^i_k. Množica N delcev je naključno vzorčena tako, da je verjetnost izbire določenega delca iz množice sorazmerna njegovi uteži w^i_k. Torej so delci z večjo utežjo izbrani večkrat kot delci z manjšo utežjo, oz. delci z najmanjšo utežjo ne smejo biti nikoli izbrani. Pristop izbora nove množice delcev lahko izvedemo na več načinov. Enega od njih bomo predstavili v nadaljevanju:

- Uteži delcev w^i_k normiramo z vsoto vseh uteži $\sum_{i=1}^N w^i_k$ in tako dobimo nove uteži $wn^i_k = \frac{w^i_k}{\sum_{i=1}^N w^i_k}$.
- Kumulativno seštejemo normirane uteži, da dobimo kumulativne uteži $wc^i_k = \sum_{j=1}^i wn^i_k$, kot prikazuje slika 6.49.
- Naključno izberemo N števil med 0 in 1 (iz enakomerne porazdelitve) in preverimo, katere uteži pripadajo izbranim številom. Torej primerjamo kumulativne uteži wc^i_k in naključno generirana števila. Glede na sliko 6.49 so delci z večjimi utežmi bolj verjetno izbrani (te uteži na sliki 6.49 zavzemajo več prostora).
- Izbrane delce uporabimo v korekcijskem koraku za oceno trenutne vrednosti stanja $\hat{\boldsymbol{x}}_{k|k} = \sum_{i=1}^N w^i_k \hat{\boldsymbol{x}}^i_{k|k-1}$.



Slika 6.49: Vzorčenje po pomembnosti v korekcijskem koraku filtra delcev. Uteži delcev nanizamo eno za drugo in nato normiramo tako, da je vsota vseh delcev enaka 1. Bolj verjetni delci zavzamejo več prostora na enotskem intervalu in obratno. Novo populacijo delcev določimo tako, da naključno izberemo (z enakomerno porazdelitvijo) N števil iz enotskega intervala in pogledamo, katerim delcem pripadajo.

V kolikor sistem miruje (trenutno stanje je enako preteklemu), je priporočeno, da ne vzorčimo po pomembnosti, ampak uporabimo kar stari nabor delcev in jim le prilagodimo nove uteži, kot je predstavljeno v [11].

V primeru 6.23 je prikazan primer uporabe filtra delcev.

Primer 6.23

Uporabite filter delcev za ocenjevanje najverjetnejšega stanja v primeru 6.22. Pri implementaciji uporabimo $N = 300$ delcev. Vsi drugi podatki so enaki kot v primeru 6.22.

Rešitev

Matlab koda rešitve je podana v programu 6.14, rezultati simulacije pa so prikazani na slikah 6.50 in 6.51.

Program 6.14: Implementacija rešitve primera 6.23

./src/prb/example_pf1default.m

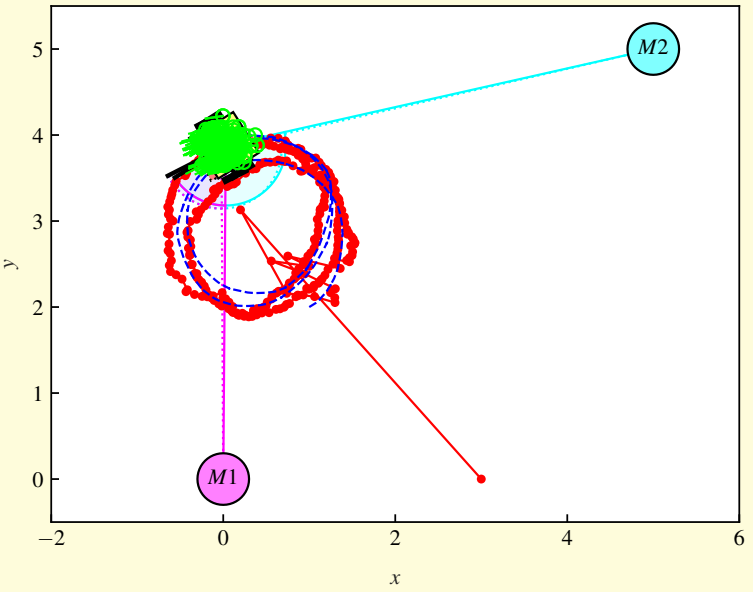
```
1 Ts = 0.1; % Računski korak
2 xTrue = [1; 2; pi/6]; % Prava začetna lega
```

```
x = [3; 0; 0]; % Ocena začetne lege
P = diag([9 9 0.6]); % Začetna kovariančna matrika ocene lege
Q = diag([0.1 0.1]); % Kovariančna matrika šuma modela gibanja
R = diag([0.5 0.3]); % Kovariančna matrika šuma merjenja razdalje in
                     % kota
enableNoise = 1; % Omogoči šum: 0 ali 1
N = 300; % Število simulacijskih korakov
marker = [0 0; 5 5]; % Položaji značk

% Inicializacija delcev
nParticles = 300;
xP = repmat(xTrue, 1, nParticles) + diag([4 4 1])*randn(3, nParticles);
W = ones(nParticles, 1)/nParticles; % Vsi delci so enako verjetni

% Zanka
for k = 1:N
    u = [0.5; 0.5]; % Ukazi (translatorna in kotna hitrost)
    u_sum = u + sqrt(Q)*randn(2, 1)*enableNoise;

    % Simulacija pravih stanj (lege) robota
    xTrue = xTrue + Ts*[u_sum(1)*cos(xTrue(3)); ...
                        u_sum(1)*sin(xTrue(3)); ...
                        u_sum(2)];
    xTrue(3) = wrapToPi(xTrue(3));

    % Simulacija meritev s šumom (razdalja in kot)
    zTrue = [];
    for m = 1:size(marker, 1)
        dist = sqrt((marker(m,1)-xTrue(1))^2 + (marker(m,2)-xTrue(2))^2);
        alpha = atan2(marker(m,2)-xTrue(2), marker(m,1)-xTrue(1))-xTrue(3);
        zz = [dist; alpha] + sqrt(R)*randn(2, 1)*enableNoise;
        zz(1) = abs(zz(1));
        zz(2) = wrapToPi(zz(2));
        zTrue = [zTrue; zz];
    end

    % Predikcija
    for p = 1:nParticles
        % Delci se premikajo glede na model šuma
        un = u + sqrt(Q)*randn(2, 1)*1;
        xP(:,p) = xP(:,p) + Ts*[un(1)*cos(xP(3,p)); ...
                                un(1)*sin(xP(3,p)); ...
                                un(2)];
        xP(3,p) = wrapToPi(xP(3,p));
    end

    % Korekcija
    for p = 1:nParticles
        % Ocenjena meritev za vsak delec
        z = [];
        for m = 1:size(marker, 1)
            dist = sqrt((marker(m,1)-xP(1,p))^2 + (marker(m,2)-xP(2,p))^2);
            alpha= atan2(marker(m,2)-xP(2,p), marker(m,1)-xP(1,p))-xP(3,p);
            zz = [dist; alpha];
            zz(1) = abs(zz(1));
            zz(2) = wrapToPi(zz(2));
            z = [z; zz];
        end

        Innov = zTrue - z; % Izračun inovacije

        % Izbira primerne inovacije,
        % zaradi šuma in cikličnosti kota
```

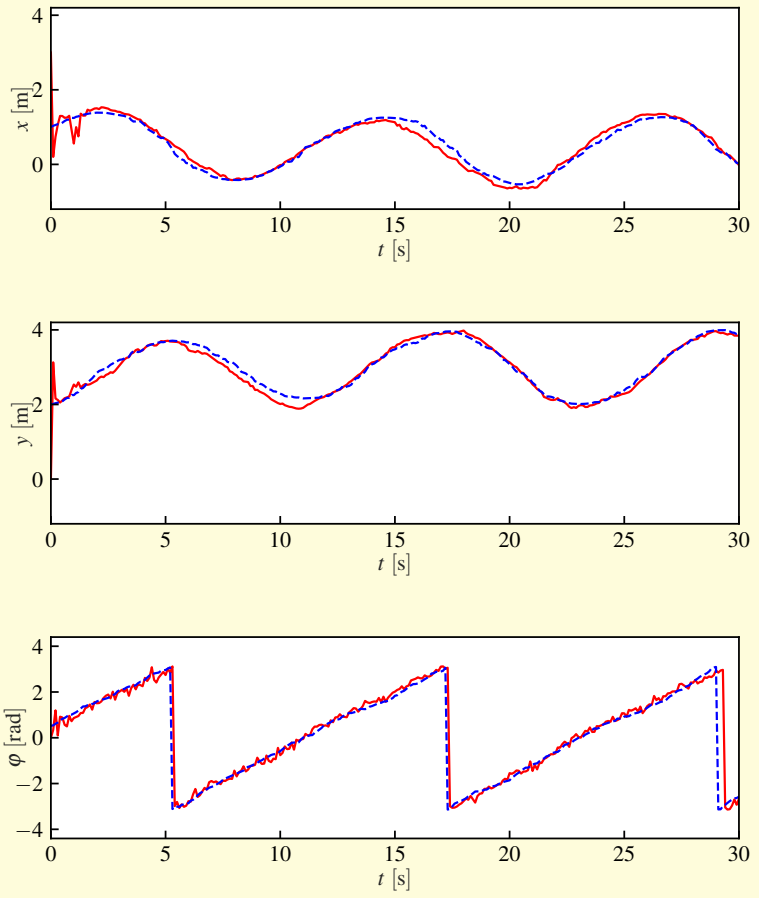
```
66          for m = 1:size(marker, 1)
67              iii = zTrue(2*m) - (z(2*m) + [0; 2*pi; -2*pi]);
68              [tmp, index] = min(abs(iii));
69              Innov(2*m) = iii(index);
70          end
71
72          % Uteži delcev (verjetnosti delcev)
73          % Kovariančna matrika meritev
74          RR = diag(repmat(diag(R), size(marker, 1), 1));
75          W(p) = exp(-0.5*Innov.'*inv(RR)*Innov) + 0.0001;
76      end
77
78      iNextGeneration = obtainNextGenerationOfParticles(W, nParticles);
79      xP = xP(:,iNextGeneration);
80
81      % Nova ocena stanj je povprečje vseh delcev
82      x = mean(xP, 2);
83      x(3) = wrapToPi(x(3));
84      % Usmeritev robota je določena z najbolj verjetnim delcem
85      % namesto s povprečnim kotom vseh delcev.
86      [gg, ggi] = max(W);
87      x(3) = xP(3,ggi);
88  end
```

Program 6.15: Funkcija, uporabljena v programih 6.14 in 6.16
./src/prb/obtainNextGenerationOfParticles.m

```
1  function iNextGeneration = obtainNextGenerationOfParticles(W, nParticles)
2      % Izbira glede na uteži delcev
3      CDF = cumsum(W)/sum(W);
4      iSelect  = rand(nParticles, 1); % Naključne številke
5      % Indeksi novih delcev
6      CDFg = [0; CDF];
7      indg = [1; (1:nParticles).'];
8      iNextGeneration_float = interp1(CDFg, indg, iSelect, 'linear');
9      iNextGeneration=round(iNextGeneration_float + 0.5); % Zaokroževanje indeksov
10 end
```



Slika 6.50: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija ter generirani delci v zadnjem koraku simulacije primera 6.23



Slika 6.51: Ocena lege (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.23

V primeru 6.23 merimo kote in razdalje do značk, pri čemer moramo upoštevati omejitev kota. Primer lokalizacije, kjer merimo le razdalje do značk za oceno lege mobilnega robota, je prikazan v primeru 6.24.

Primer 6.24

Uporabite filter delcev za oceno najverjetnejšega stanja v primeru 6.23, vendar upoštevajte le merjenje razdalj do značk. V implementaciji uporabite $N = 500$ delcev. Vsi ostali podatki so enaki kot v primeru 6.23.

Rešitev

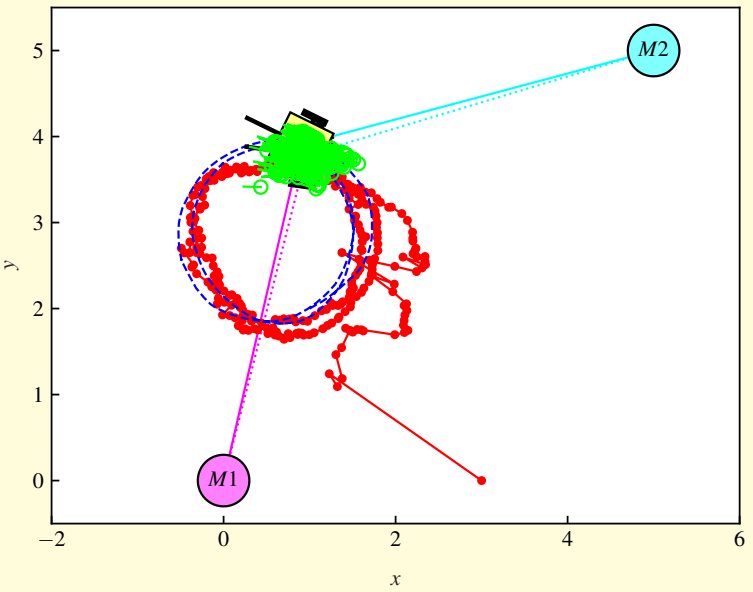
Matlab koda rešitve je prikazana v programu 6.16. Rezultati simulacije so prikazani na slikah 6.52 in 6.53, od koder je razvidno, da ocenjena vrednost konvergira k pravi vrednosti podobno kot v primeru 6.23.

Program 6.16: Implementacija rešitve primera 6.24

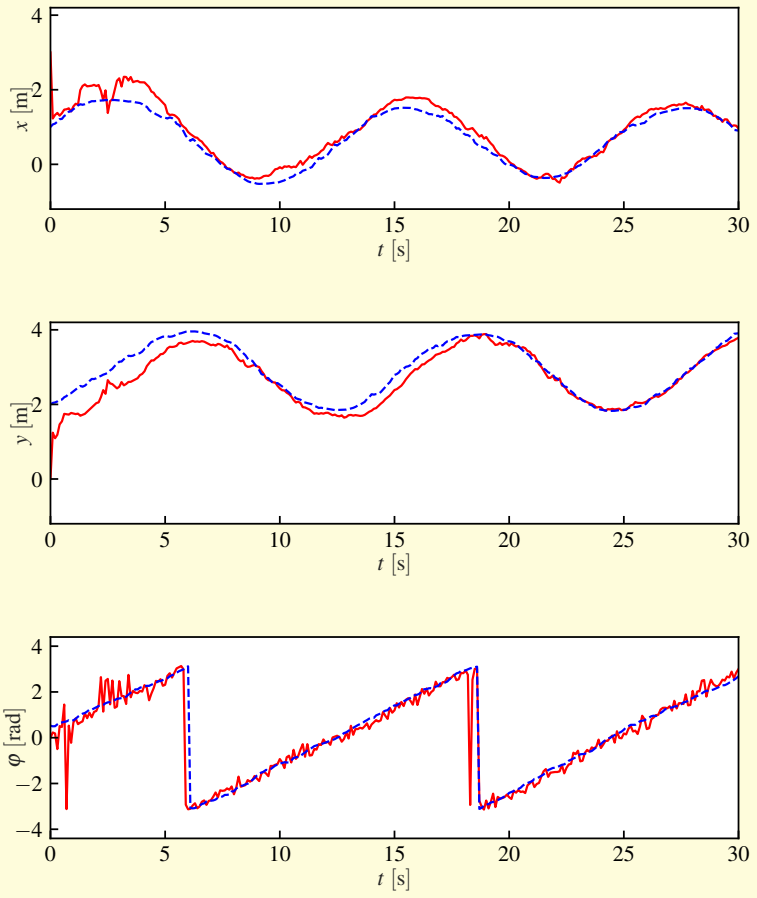
./src/prb/example_pf2default.m

```
Ts = 0.1; % Računski korak
xTrue = [1; 2; pi/6]; % Prava začetna lega
x = [3; 0; 0]; % Ocena začetne lege
P = diag([9 9 0.6]); % Začetna kovariančna matrika ocene lege
Q = diag([0.1 0.1]); % Kovariančna matrika šuma modela gibanja
R = diag([0.5 0.3]); % Kovariančna matrika šuma merjenja razdalje in
                     % kota
enableNoise = 1; % Omogoči šum: 0 ali 1
N = 300; % Število simulacijskih korakov
marker = [0 0; 5 5]; % Položaji značk
R = R(1,1); % Le merjenje razdalje

% Inicializacija delcev
nParticles = 500;
xP = repmat(xTrue, 1, nParticles) + diag([4 4 1])*randn(3, nParticles);
W = ones(nParticles, 1)/nParticles; % Vsi delci so enako verjetni

% Zanka
for k = 1:N
    u = [0.5; 0.5]; % Ukazi (translatorna in kotna hitrost)
    u_sum = u + sqrt(Q)*randn(2, 1)*enableNoise;

    % Simulacija pravih stanj (lege) robota
    xTrue = xTrue + Ts*[u_sum(1)*cos(xTrue(3)); ...
                        u_sum(1)*sin(xTrue(3)); ...
                        u_sum(2)];
    xTrue(3) = wrapToPi(xTrue(3));

    % Simulacija meritev s šumom (razdalja)
    zTrue = [];
    for m = 1:size(marker, 1)
        dist = sqrt((marker(m,1)-xTrue(1))^2 + (marker(m,2)-xTrue(2))^2);
        zz = [dist] + sqrt(R)*randn(1, 1)*enableNoise;
        zz(1) = abs(zz(1));
        zTrue = [zTrue; zz];
    end

```

```
38      % Predikcija
39      for p = 1:nParticles
40          % Delci se premikajo glede na model šuma
41          un = u + sqrt(Q)*randn(2, 1)*1;
42          xP(:,p) = xP(:,p) + Ts*[un(1)*cos(xP(3,p)); ...
43                                  un(1)*sin(xP(3,p)); ...
44                                  un(2)];
45          xP(3,p) = wrapToPi(xP(3,p));
46      end
47
48      % Korekcija
49      for p = 1:nParticles
50          % Ocenjena meritev za vsak delec
51          z = [];
52          for m = 1:size(marker, 1)
53              dist = sqrt((marker(m,1)-xP(1,p))^2 + (marker(m,2)-xP(2,p))^2);
54              zz = [dist];
55              zz(1) = abs(zz(1));
56              z = [z; zz];
57          end
58
59          Innov = zTrue - z; % Izračun inovacije
60
61          % Uteži delcev (verjetnosti delcev)
62          % Kovariančna matrika meritev
63          RR = diag(repmat(diag(R), size(marker, 1), 1));
64          W(p) = exp(-0.5*Innov.'*inv(RR)*Innov) + 0.0001;
65      end
66
67      iNextGeneration = obtainNextGenerationOfParticles(W, nParticles);
68      xP = xP(:,iNextGeneration);
69
70      % Nova ocena stanj je povprečje vseh delcev
71      x = mean(xP, 2);
72      x(3) = wrapToPi(x(3));
73      % Usmeritev robota je določena z najbolj verjetnim delcem
74      % namesto s povprečnim kotom vseh delcev.
75      [gg, ggi] = max(W);
76      x(3) = xP(3,ggi);
77  end
```



Slika 6.52: Dejanska (črtkana krivulja) in ocenjena (polna krivulja) trajektorija ter generirani delci v zadnjem koraku simulacije primera 6.24



Slika 6.53: Ocena lege (polna krivulja) in pravo stanje (črtkana krivulja) mobilnega robota z začetnim neničelnim pogreškom ocene iz primera 6.24

Filter delcev je implementacija Bayesovega filtra za zvezne sisteme (zvezni prostor stanj), ki omogoča opis nelinearnih sistemov in lahko upošteva poljubno porazdelitev šuma. V primeru sistemov večjih dimenzij postane računsko precej zahteven, saj je za ustrezno konvergenco filtra potrebno veliko število delcev. Število potrebnih delcev narašča z dimenzijo prostora stanj.

Dobra lastnost filtra delcev je robustnost ter zmožnost rešitve problema globalne lokalizacije in ugrabitve robota. Pri problemu globalne lokalizacije je začetna lega (vrednost stanj) neznana, zato se lahko mobilni robot nahaja kjerkoli v prostoru. Pri problemu ugrabitve pa je robot premaknjen (ugrabljen) na poljubno novo lokacijo. Robustni lokalizacijski algoritmi so sposobni rešiti te težave.

Literatura

[1] D. C. Montgomery in G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Englewood Cliffs, New Jersey, 2003, 720 str.

[2] R. Hermann in A. J. Krener. Nonlinear Controllability and Observability. *IEEE Transaction on Automatic Control*, zv. 22, št. 5, str. 728–740, 1977.

[3] W. Respondek. *Introduction to geometric nonlinear control; linearization, observability, decoupling*, *ICTP Lecture Notes Series*, zv. 8, str. 173–222. ICTP, 2002.

[4] A. Martinelli. Nonlinear unknown input observability: Analytical expression of the observable codistribution in the case of a single unknown input. V *2015 Proceedings of the Conference on Control and its Applications*, str. 9–15. 2015.

[5] D. Fox, J. Hightower in sod. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, zv. 2, št. 3, str. 24–33, 2003.

[6] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, zv. 82, št. Series D, str. 35–45, 1960.

[7] L. Teslić, I. Škrjanc in G. Klančar. Using a LRF sensor in the Kalman-filtering-based localization of a mobile robot. *ISA transactions*, zv. 49, št. 1, str. 145–153, 2010.

[8] L. Teslić, I. Škrjanc in G. Klančar. EKF-based localization of a wheeled mobile robot in structured environments. *Journal of Intelligent & Robotic Systems*, zv. 62, št. 2, str. 187–203, 2010.

[9] G. Klančar, L. Teslić in I. Škrjanc. Mobile-robot pose estimation and environment mapping using an extended Kalman filter. *International Journal of Systems Science*, zv. 45, št. 12, str. 2603–2618, 2014.

[10] B. D. O. Anderson in J. B. Moore. *Optimal Filtering*. Prentice-Hall, Englewood Cliffs, New Jersey, 1979, 367 str.

[11] S. Thrun, W. Burgard in D. Fox. *Probabilistic robotics*. MIT Press, 4. izd., 2005, 647 str.

7

Agenti in večagentni sistemi

7.1 Uvod

Eden od načinov reševanja določenih nalog je vpeljava agenta ali entitete, tj. smiselno zaključena celota, ki je zmožna sama bolj ali manj uspešno reševati določen problem. Agenti so lahko fizični (robot) in vplivajo na stvarni svet, ali pa virtualni (simulacije, programske komponente) in vplivajo na virtualno okolje. Agenti, ki delujejo v nekem okolju, sestavljajo **večagentni sistem**. Večagentni sistemi podajajo principe za gradnjo kompleksnih sistemov s pomočjo agentov in mehanizmov za koordinacijo delovanja neodvisnih agentov.

Osnovno vodenje ali delovanje agenta je potrebno, ne pa tudi zadostno, za usklajeno delovanje skupine agentov pri doseganju skupnega cilja. Vodenje večagentnega sistema je tako vedno kombinacija učinkovitega delovanja na nivoju osnovnih agentov in ustreznega sodelovanja med njimi.

V nadaljevanju je podanih nekaj definicij ter klasifikacij agentov in večagentnih sistemov.

7.2 Večagentni sistemi

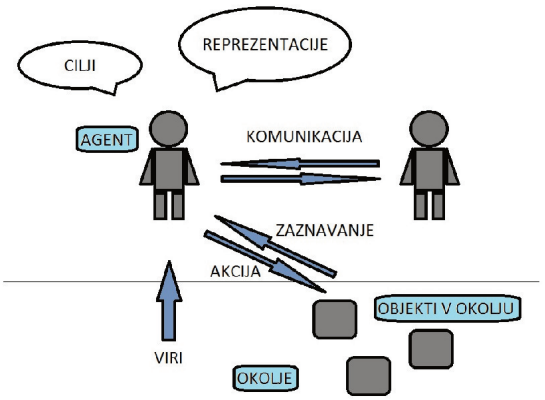
Večagentni sistemi so razmeroma mlada veda na področju umetne inteligence. Pristopi večagentnih sistemov posegajo na področje porazdeljene umetne inte-

ligence in umetnega življenja. Namen prvega je razvoj organizacije sistemov, ki so zmožni reševati probleme z razmišljanjem, drugi pa skuša modelirati žive organizme, zato preučuje tudi zmožnost preživetja in prilagajanja v običajno sovražnem okolju. Porazdeljeni sistemi so v računalništvu zelo dobro uveljavljeni (večprocesorski sistemi), medtem ko metode za koordinacijo več agentov v robotiki pridobivajo na priljubljenosti.

Večagentni sistem je sestavljen iz več popolnoma ali delno avtonomnih agentov in podaja mehanizme za koordinacijo njihovega delovanja. Agenti izkazujejo neko obnašanje, ki ga pogosto določajo enostavna pravila in na katerega vpliva komunikacija z drugimi agenti ter interakcije z okoljem in objekti v okolju. Izziv večagentnih sistemov je predvsem sodelovanje več agentov, kar dosežemo z izvedbo nekaj korakov za zagotovitev njihove sinhronizacije, komunikacije (neposredno: sistem sporočil, skupne tabele ipd.; posredno: opazovanje ostalih, sklepanje) in pogajanj o delitvi dela.

Večagentni sistem, prikazan na sliki 7.1, lahko torej opredelimo kot sistem, ki v splošnem vsebuje

- okolje,
- množico pasivnih objektov,
- množico agentov (aktivni objekti v okolici) ter
- množico odnosov in metod interakcije agentov z objekti okolice.



Slika 7.1: Prikaz večagentnega sistema, kjer agent zaznava in vpliva na okolje ter ostale agente v okolju

Večagentne sisteme, kjer so edini objekti agenti ter okolje ni definirano, imenujemo *komunikacijski večagentni sistemi*. V tem primeru odnosi med agenti predstavljajo omrežje, v katerem je vsak agent povezan z ostalimi. Taki sistemi so pogosti na področju porazdeljene umetne inteligence (DAI), kjer so agenti tipično programski moduli.

V kolikor pa so agenti situirani v okolju ter komunikacija poteka le posredno preko zaznavanja in delovanja na okolje, imamo *izključno situiran večagentni sistem*. Splošni večagentni sistemi imajo lastnosti obeh omenjenih skrajnih oblik.

7.3 Agenti

Čeprav stroga, uveljavljena definicija agenta ne obstaja, lahko rečemo, da agenta predstavlja entiteta v nekem okolju, ki lahko okolje tudi zazna in v njem deluje, ima cilje, znanje iz določenega področja ter zmožnost odločanja. Agent ima senzorje, s katerimi zaznava okolje (npr. senzor bližine zazna oviro), aktuatorje, s katerimi vpliva na okolje (npr. kolesni pogon premakne robota in/ali odrine oviro), ter znanje o okolju, v katerem deluje in mu omogoča, da s pomočjo informacije iz senzorjev upravlja svoje aktuatorje za doseg cilja (npr. doseg želene lokacije).

Naštejmo nekaj lastnosti, ki opisujejo fizičnega ali virtualnega agenta:

- zmožnost delovanja v okolju,
- zmožnost komunikacije z ostalimi agenti,
- ima nabor svojih teženj in ciljev,
- ima dostop do virov (napajanje, CPU, spomin, informacije),
- ima zmožnost zaznavanja svoje okolice (do določene mere),
- ima svojo (delno) predstavitev okolice, ali pa je sploh nima,
- lahko se reproducira,
- njegovo delovanje stremi k dosegu ciljev, kjer uporablja vire, svoja znanja, zaznave senzorjev, svojo predstavitev okolice (oz. znanje o njej) in komunikacijo.

Pomembna lastnost agenta je *avtonomnost*, kar pomeni, da agent ni upravljan preko operaterja ali drugega agenta, ampak je sposoben samostojnega delovanja glede na lastne cilje in situacije, v katerih se znajde. Avtonomni agent ima tudi dostop do lastnih virov, kot so napajanje, pomnilnik, informacije, itd. Agentovo zaznavanje je omejeno z lastnostmi lastnih senzorjev, zato ima le delno predstavitev okolja, saj ne more zaznavati vsega dogajanja v okolju. Na voljo so mu samo lokalne informacije, torej tiste v dosegu njegovih senzorjev, zato so večagentni sistemi večinoma decentralizirani (obnašanje agentov ni centralno nadzorovano). Agent ima svoje trenutno stanje v okolju predstavljeno s spremenljivkami, njegovo delovanje pa je odvisno od stanja, v katerem se nahaja. Več ko ima možnih stanj, na več različnih načinov lahko deluje. Agenti v večagentnem sistemu se lahko

razlikujejo po lastnostih, obnašanju, virih, zmožnosti predstavitev, sposobnosti pomnjenja dogodkov in interpretaciji razpoložljivih informacij.

Agent mora imeti sposobnost prilagajanja. Hkrati mora biti njegovo vgrajeno znanje fleksibilno, da ga lahko dopolnjuje s spreminjanjem določenih parametrov. Možni so tudi določeni algoritmi, ki slonijo na evoluciji živih bitij, ter ostali algoritmi strojnega učenja (genetski algoritmi, nevronske mreže, učenje z nagrajevanjem). Uspeh teh metod pogojuje dejstvo, da je problem umetne inteligence pogosto kombinacijsko preveč kompleksen, da bi bil rešljiv v realnem času. Zato se inteligenca agenta skoraj vedno sestoji le iz dveh virov: znanja, pridobljenega na osnovi lastnih izkušenj (učenje, adaptiranje), ter vgrajenega znanja.

Agent ima za razliko od ostalih programov in objektno orientiranega programiranja naslednje lastnosti:

- zaznava okolje, v katerem se nahaja,
- ima sposobnost interakcije z ostalimi agenti in (najpomembnejše)
- na poti k izpolnjevanju lastnih ciljev se odloča in deluje samoiniciativno.

Objekti so pasivni elementi, ki nimajo možnosti izbire svojega delovanja, temveč delujejo le na zunanjo iniciativo.

7.4 Arhitektura in delovanje agentov

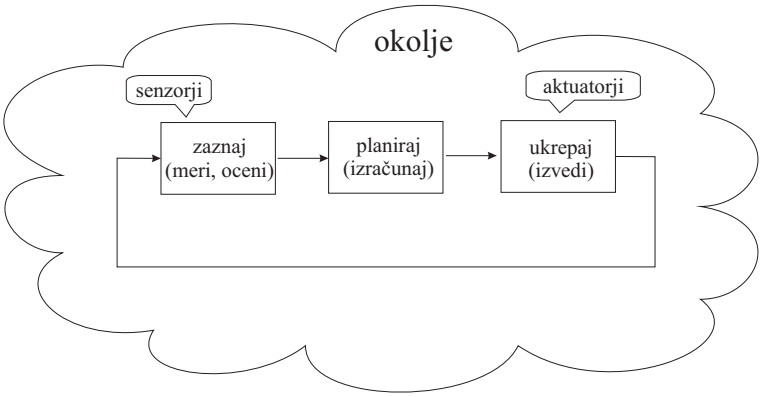
Klasičen in uveljavljen način vodenja v mobilni robotiki in avtomatiki (od leta 1985 dalje) temelji na načelu ***zaznaj-planiraj-ukrepaj*** (SPA, angl. *sense-plan-act*). Sistem najprej s senzorji pridobi informacijo iz okolja, nato pa zgradi model z uporabo pridobljene informacije in načrta, oz. izračuna naslednji korak. Agent mora torej ugotoviti, kako naj se z vgrajeno strategijo odzove na zaznane podatke. Na koncu agent ukrepa in izvede akcijo. SPA poteka v iteracijah – po zaznavanju, planiranju in ukrepanju se celoten cikel ponovi.

SPA je osnova avtomatskega vodenja, kjer se poskuša postopno zmanjševati pogrešek med želenim in dejanskim stanjem mobilnega (ali kateregakoli drugega) sistema (slika 7.2).

7.4.1 Kognitivni agenti

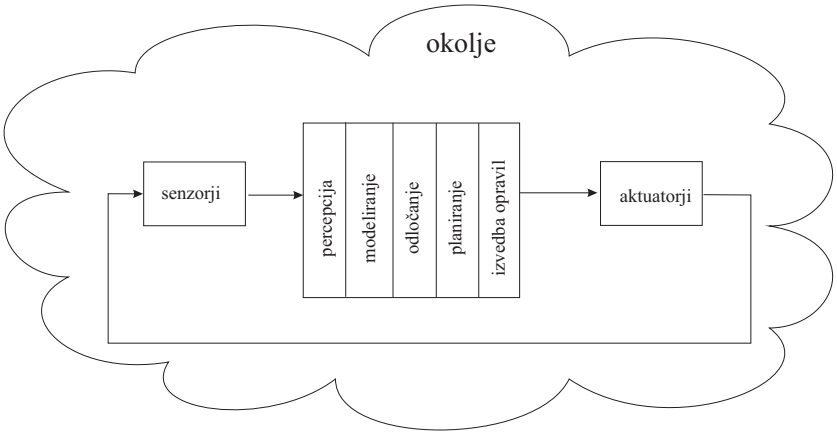
Kognitivni agenti (angl. *deliberative agents*) delujejo po principu SPA. Ko agent zazna okolico s pomočjo modela sveta (simbolična predstavitev okolice), naredi načrt za izvedbo akcije.

Agent določi načrt reševanja problema (slika 7.3) korak za korakom (interpretacija zaznav senzorjev, modeliranje, odločanje, planiranje, izvedba opravil, upravljanje



Slika 7.2: Osnovni način upravljanja agenta (SPA)

aktuatorjev) na osnovi svojega zaznavanja okolja. Vsak tak agent ima običajno bazo podatkov in znanje, potrebno za reševanje problemov. V nepredvidljivem dinamičnem okolju agent z izključno kognitivnimi sposobnostmi ni učinkovit, saj njegov načrt reševanja problemov ne more predvideti sprememb okolja in bi ga moral nenehno spreminjati.



Slika 7.3: Kognitivni agent naredi načrt reševanja problema na osnovi zaznav

Kognitivni agenti imajo nedvomno prednost v statičnih in poznanih okoljih. V primeru nepričakovanih dogodkov, glede na njihov model sveta, pa lahko odpovejo. Potrebujejo precej natančen model sveta (npr. zemljevid), ki ga je pogosto težko dobiti in vzdrževati. Za svoje delovanje potrebujejo veliko procesno moč, kar se lahko odraža v počasnem odzivu na spremembe v oklici.

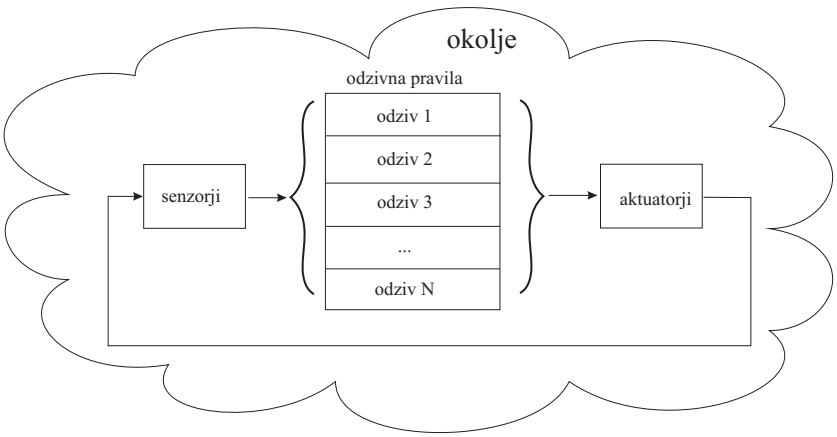
7.4.2 Odzivni agenti

Odzivni agenti (slika 7.4) so zmožni povezati svoje zaznavanje okolja z akcijami vodenja (različno vnaprej definirano obnašanje) in s tem kar najbolje izvršiti

naloge brez gradnje internega modela okolja (kar je sorodno obnašanju manjših živali, npr. mravelj).

Delujejo torej po principu *zaznaj-deluj* brez uporabe simbolične predstavitve okolja (modela sveta) in planiranja. Zanašajo se le na eno ali več enostavnih pravil, ki neposredno povežejo zaznave senzorjev z akcijami.

V osnovi odzivni agenti nimajo stanj (ne shranjujejo nekaterih preteklih podatkov), so brez spomina in internega modela okolice, nimajo možnosti planiranja akcij vnaprej ter niso zmožni učenja. Njihova prednost je ravno v njihovi preprostosti, kar jim omogoča hiter (trenuten) odziv.



Slika 7.4: Odzivni agent reagira na zaznave brez načrtovanja

7.4.3 Hibridni agenti

V nepredvidljivem dinamičnem okolju so primernejši hibridni agenti, ki združujejo dobre lastnosti odzivnih in kognitivnih agentov. Obstajajo agenti, ki nimajo celotne ali obsežne simbolične predstavitve sveta okoli njih, ampak si zapomnijo le nekaj pomembnejših parametrov, kar jim lahko pomaga pri boljši asociaciji zaznav z akcijami ali izvedbi bolj dovršene akcije (npr. agent si lahko zapomni, da je v bližini stene).

Nadalje ima lahko agent zmožnost adaptacije. To pomeni, da spreminja vzorce delovanja (obnašanje) in se prilagaja spreminjajočim se razmeram glede na svoje prejšnje izkušnje. Z drugimi besedami lahko temu rečemo tudi učenje. Za učenje na individualni ravni mora imeti agent spomin – torej pri popolnoma odzivnih agentih to ni mogoče.

Obstaja tudi prilagodljivost na ravni sistema, ki je mogoča tudi pri večagentnih sistemih, sestavljenih iz odzivnih agentov. Če se agenti v sistemu lahko reproducirajo, se lahko poveča število tistih agentov, ki so bolj primerni za novonastale razmere.

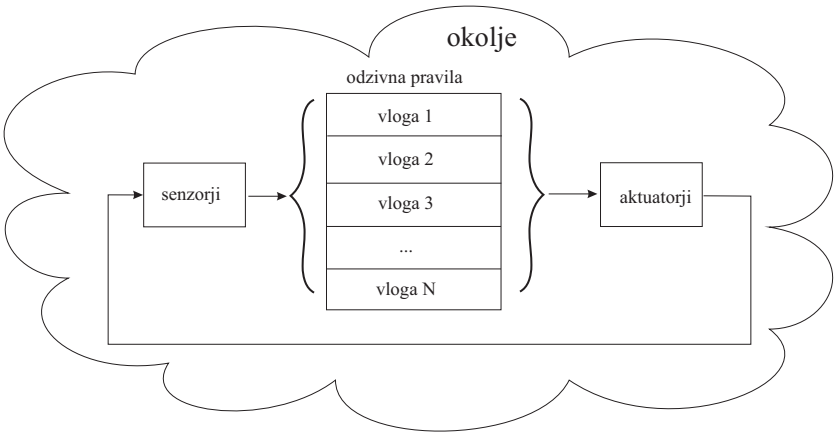
7.4.4 Odzivno vedenjski agenti

Njihovo delovanje je enako delovanju odzivnih agentov, le da namesto enostavnih pravil uporabljajo *vloge*. Vloge predstavljajo module, ki se izvajajo paralelno (slika 7.5), torej ima vsak modul dostop do senzorjev in lahko neposredno upravlja z aktuatorji. Vsaka vloga vsebuje neko znanje v obliki algoritmov vodenja, ki agentu omogočajo primerno delovanje v določeni situaciji (sledenje steni, iskanje predmeta, izogibanje oviri, prihod v začetni položaj itd.).

Podajmo nekaj lastnosti vedenjskih agentov:

- vsebujejo različne vloge za doseg ali sledenje ciljem,
- vhodne informacije prejmejo vloge od senzorjev in drugih vlog ter posredujejo ukaze aktuatorjem,
- vloge so lahko kompleksne in sestavljene iz različnih akcij (akcije: stop, naprej, levo itd.; vloge: sledenje cilju, izogibanje oviri).

Ker se vloge izvajajo hkrati in neodvisno, so taki agenti primerni za aplikacije v realnem času. Vloge imajo lahko stanja (si zapomnijo zgodovino), model okolice in zmožnost planiranja vnaprej, kar omogoča izvedbo učinkovitih vlog.



Slika 7.5: Vedenjski agent se odziva na zaznave z izvajanjem vlog

Primer 7.1

Poglejmo si primer izvedbe enostavnega kognitivnega agenta in odzivnega agenta. Agent je mobilni robot, ki želi iti skozi zaklenjena vrata.

Rešitev

Kognitivni agent lahko planira svoje delovanje, torej bo zgradil načrt v več zaporednih korakih v obliki:

```
Načrt odpiranja vrat:
 Grem do mesta, kjer je spravljen ključ,
 vzamem ključ,
 grem do vrat,
 odprem vrata s ključem.
```

Odzivni agent pa se brez načrtovanja ali razmišljanja odzove na situacije iz okolja. Njegovo obnašanje omogoča skupek enostavnih pravil *Pi* oz. vlog:

```
 P1: Če sem pred vrati in imam ključ, potem odprem vrata.
 P2: Če sem pred vrati in nimam ključa, potem poskusim
     odpreti vrata.
 P3: Če se vrata ne odprejo in nimam ključa,
     potem grem iskat ključ.
 P4: Če iščem ključ in vidim ključ pred sabo, potem
     vzamem ključ in grem proti vratom.
```

Vidimo, da kognitivni agent zgradi načrt, medtem ko ima odzivni agent že predhodno vgrajena pravila. Kognitivni agent bo gotovo odprl vrata hitreje, z manj akcijami, saj lahko predvidi zaporedje potrebnih akcij. Odzivni agent pa bo najprej šel do vrat in nato ugotovil, da nima ključa in da ga mora iti iskat. Vendar je odzivni agent bolj robusten: če so vrata odprta, jih bo odprl takoj, ne da bi šel po ključ. Kognitivni agent pa bo šel najprej po ključ, saj njegov model ne predvideva možnosti, da so vrata mogoče že odprta.

7.4.5 Osnovne vedenjske arhitekture

Možnih je več načinov izvedbe arhitekture odzivno vedenjskih agentov. Osnovni arhitekturi sta tekmovalna in vsebovana shema.

Tekmovalna shema

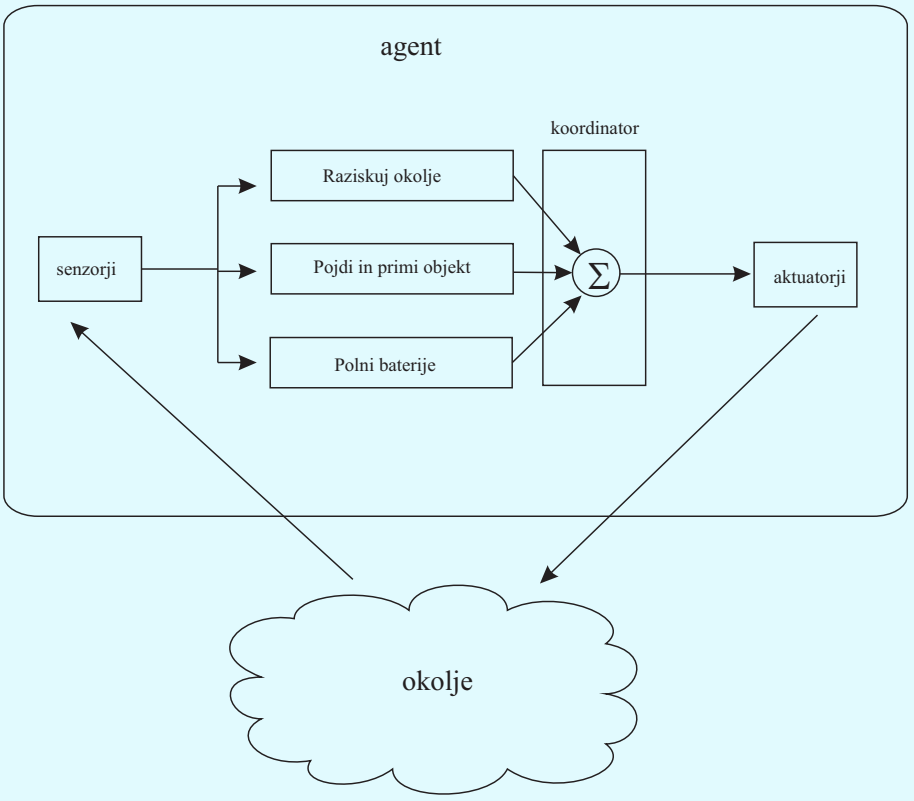
Tekmovalna shema (angl. *competitive architecture, motor schema architecture*) je princip, ki ga je vpeljal Arkin [1]. Vloga sestoji iz sheme percepcije, ki procesira vhode iz senzorjev in jih posreduje motornim shemam. Tekmovalne ali motorne sheme generirajo izhode za vodenje, ki določajo premikanje robota, da doseže cilj. Gre za to, da več konkurenčnih vlog (shem) posreduje svoje ukaze (hitrost, smer gibanja itd.) agentu, ukazi posameznih vlog pa so z uporabo potencialnega

polja predstavljeni kot vektorji, ki so normirani glede na percepcijo in motorno shemo vloge. Prispevki posameznih vlog se nato združijo v končen ukaz, ki je posredovan aktuatorju agenta. Druga možnost pa je, da se izmed vseh vlog izbere le najbolj uspešno (uspešnost se oceni na podlagi določenih parametrov).

V osnovi gre za privlačna in odbojna polja. Predstavljajmo si primer, kjer agenta privlači cilj (vektor smeri vožnje je v smeri cilja), hkrati pa ga odbija od ovire (vektor želene smeri vožnje kaže stran od ovire). Bliže je agent oviri, bolj prevladuje odbojni vektor smeri in privlačni vektor proti cilju se zmanjšuje. Končna usmeritev je vektorska vsota teh dveh normiranih vektorskih polj.

Primer 7.2

Poglejmo si primer vedenjskega agenta, katerega vloge so organizirane v tekmovalno oz. motorno shemo. Imamo preprostega raziskovalnega robota, ki raziskuje okolje in ob zaznavi predmeta gre ponj. Ko mu zmanjka energije, gre napolnit akumulatorje. Nabor vlog za izvedbo delovanja agenta povežemo v strukturo, kot nakazuje slika 7.6.



Slika 7.6: Tekmovalna shema

Vsebovana shema

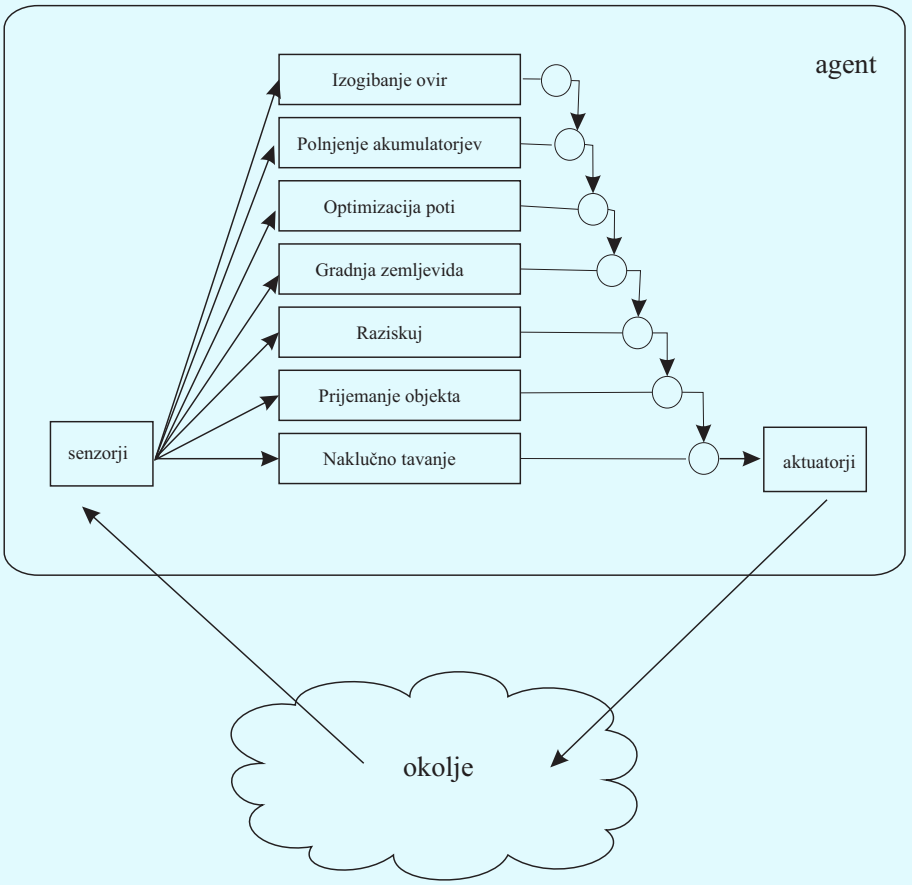
Vsebovana shema (angl. *subsumption architecture*) je način dekompozicije inteligentnega obnašanja agenta v več preprostih vlog, ki so organizirane po nivojih glede na prioriteto. Posamezne nivoje lahko zgradimo z uporabo končnih avtomatov. Pojem je vpeljal Rodney Brooks [2]. Vse vloge se izvajajo hkrati in prejemajo informacijo od senzorjev. Vloge z višjo prioriteto posredujejo ukaze aktuatorjem. Tu velja opomniti, da določene vloge (opravila) lahko onemogočijo ali spremenijo percepcijo, pa tudi povozijo akcije nižjih prioritetnih vlog.

Vloge lahko onemogočijo svoje delovanje (onemogočijo vhode ali izhode), če na podlagi senzorjev ni izpolnjen pogoj za njihovo izvajanje. Vloga z višjo prioriteto lahko onesposobi vloge z nižjo prioriteto. Vloga z najvišjo prioriteto, ki ostane aktivna, določa novo akcijo.

Vloge z višjimi prioritetami (v višjih slojih) so bolj abstraktne in lahko popolnoma dosežejo cilj. Višje vloge vključujejo funkcije nižjih vlog. Vloge z nižjimi prioritetami (nižji sloji) pa so preprostejše in hitro odzivne (refleksi).

Primer 7.3

Poglejmo si primer vedenjskega agenta, katerega vloge so organizirane v vsebovano shemo. Raziskovalnega robota z nekoliko nadgrajeno funkcionalnostjo primera 7.6 podaja slika 7.7. Nabor vlog za izvedbo delovanja agenta povežemo v vsebovano strukturo, kjer so vloge razdeljene v nivoje. Vloge v višjih nivojih lahko onemogočijo vloge v nižjih nivojih, kar nakazujejo krogci.



Slika 7.7: Vsebovana shema

7.4.6 Ostale delitve agentov in večagentnih sistemov

Večagentne sisteme lahko obravnavamo glede na štiri lastnosti agentov:

- granulacija agentov (fina ali groba),
- raznolikost znanja agentov (splošno ali specializirano),
- znanje agenta (konstruktivno ali tekmovalno, ekipno ali hierarhično, statično ali dinamično spreminjanje vloge) in
- komunikacija agentov (oglasna deska ali sporočila, malo ali veliko komunikacije, neposredna ali posredna vsebina).

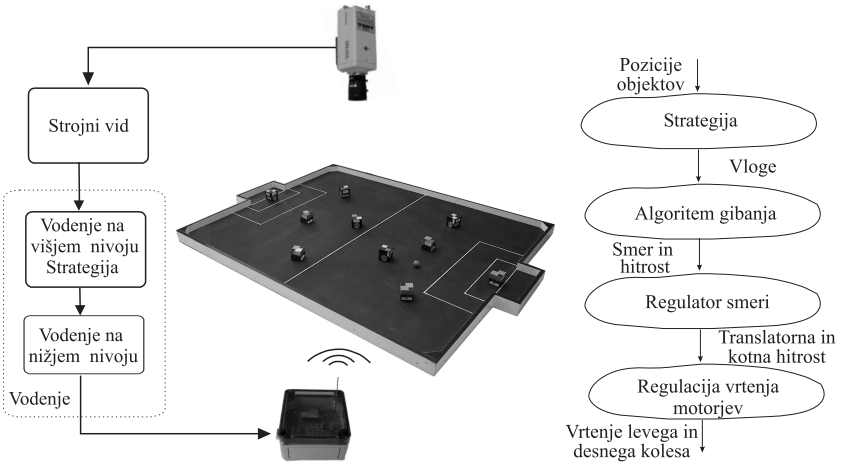
Ponavadi imajo agenti grobo granulacijo in visoko stopnjo komunikacije, v ostalih lastnostih pa se razlikujejo. Skupina sodelujočih agentov je pri reševanju kompleksnega problema lahko bolj prilagodljiva in ekonomična kot en sam zmogljivejši agent, če le uspemo učinkovito rešiti oz. zagotoviti njihovo koordinacijo. Dejstvo pa je, da ni smiselno pretiravati z večanjem števila agentov pri opravljanju nekega opravila, ker je v tem primeru vloženo preveč energije v njihovo koordinacijo, komunikacijo in pogajanja. Isto opravilo lahko enako ali bolj učinkovito opravi tudi manj agentov.

7.5 Primeri uporabe večagentnih sistemov

Področje uporabe večagentnih sistemov je zelo široko. Tako imamo aplikacije v avtomatizaciji proizvodnje (avtomobilska proizvodnja, avtonomni vozički v skladiščih) in robote skavte (nevarna območja). Nekatere aplikacije pošiljajo robote skavte v izvidnico, kjer le-ti med seboj sodelujejo, raziskujejo in kartirajo teren. To so lahko simulacije ali pa resnične aplikacije (vojska, vesolje, nevarni tereni: globina, vulkani, minska polja itd.). Modele večagentnih sistemov lahko uporabljamo za simulacijo transporta in prometa, za raziskovanje potrošniških in finančnih trgov, preučevanje razširjanja epidemij, optimizacijo proizvodnje in logistike, simulacijo premikov bojnih enot na bojišču in simulacijo socialnih mrež. Večagentni sistemi se uporabljajo tudi v filmski industriji za simulacijo velikih množic ljudi. V filmih, kot so Avatar, Gospodar prstanov, King Kong ipd., je bil uporabljen programski paket *MASSIVE* (angl. *Multiple agent simulation system in virtual environment*). Nekateri modeli so enostavnejši in zajemajo le bistvene lastnosti sistemov, drugi pa so kompleksnejši in preverjeni tudi za uporabo podatkov iz realnega sveta. S pomočjo razvoja specialne programske opreme za modeliranje večagentnih sistemov in povečevanjem računske moči računalnikov je možno ustvarjati vedno bolj napredne večagentne aplikacije, s katerimi pridemo do točnejših rezultatov in ugotovitev na raznovrstnih strokovnih področjih.

7.5.1 Robotski nogomet — avtonomna igra kolesnih robotov

Prikazan je primer, kako napisati program za kolesne robote, da ti lahko avtonomno igrajo nogomet. Vsak robot je predstavljen kot agent, ki lahko okolje zaznava in v njem deluje v skladu s svojimi cilji in znanjem, ki ga o okolici ima. Delovanje agentov je izvedeno s pomočjo predstavljenih arhitektur delovanja (odzivne, kognitivne, hibridne in vedenjske). Vsak agent ima zakodirano znanje, potrebno za izvedbo osnovnih opravil, kot so vožnja v želeno lego, streljanje žoge



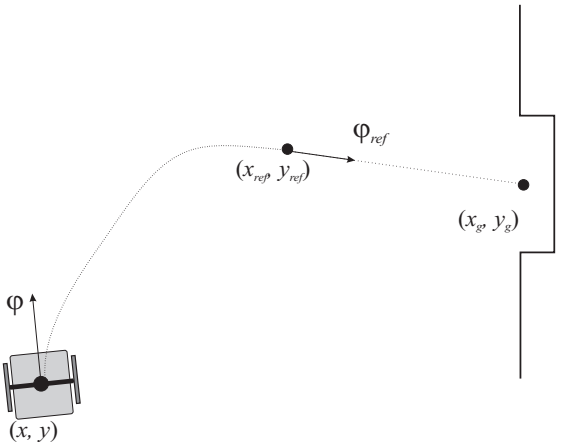
Slika 7.8: Poligon za robotski nogomet (levo) in diagram izvedbe vodenja (desno)

v želeno smer, branjenje gola in podobno. Ker pri igri nogometa sodeluje več agentov je potrebno zagotoviti ustrezno koordinacijo njihovega delovanja. Slednje je zagotovljeno z vsebovano shemo in s predpisanimi prioritetami izvajanja vlog glede na želeno strategijo igre.

V nadaljevanju so prikazane osnovne izvedbe vlog (npr. vratar, napadalec, obrambni igralec itd.), ki jih posamezni agenti lahko izvajajo. Pri tem so ključni algoritmi, za izvedbo gibanja dvokolesnih robotskih vozil po trajektoriji, v želeno lego in za izogibanja ovir. Te algoritme lahko nato uporabimo za izvedbo vlog agentov (npr. vratar, napadalec). Vloge lahko na podlagi zaznanih informacij iz senzorjev določijo ustrezne akcije. Primer je lahko vratar, ki se mora premakniti na ustrezno pozicijo, da brani strele žoge ali napadalec, ki mora znati priti do žoge in jo ustreliti (potisniti) v smeri nasprotnikovega gola. Vloge pa se morajo znati tudi izogibati oviram, ki so lahko drugi igralci ali ograja igrišča. Nekatere vloge so povsem odzivne druge ali pa vsebujejo tudi elemente planiranja (kognitivne vloge). Primer slednjega je prediktivno delovanje za bolj učinkovito in hitrejše prestrezanje gibajoče žoge.

Slika 7.8 prikazuje testni poligon, ki sestoji iz dvokolesnih robotov (deset robotov za dve moštvi) kockaste oblike s stranico 7.5 cm, igrišča velikosti 2.2×1.8 m, barvne kamere in osebnega računalnika. Kamera je nameščena nad igriščem in omogoča sledenje objektov (žoge in igralcev) na podlagi barvne informacije [3].

Osnovno delovanje posameznega agenta (robota) prikazuje levi del slike 7.8. Na podlagi trenutnih lokacij objektov in izbrane strategije delovanja program agentom dodeli ustrezne vloge. Vloga med drugim vsebuje algoritem za določitev želene smeri in hitrosti gibanja agenta. Algoritem vodenja nato določi referenčne ukaze za translatorno in kotno hitrost, ki se robotu pošljejo preko brezžične povezave. Robot nato s pomočjo internega regulatorja PID doseže želene hitrosti vrtenja koles. Omenjeni cikel se izvaja s frekvenco 30 ali 60 Hz.



Slika 7.9: Algoritem vodenja napadalnega robota, ki ustreli žogo proti golu

Podrobnejše delovanje sistema je opisano v treh podpoglavjih. V prvem so opisani algoritmi za izvedbo gibanja. V drugem delu predstavimo vloge za izvedbo odzivno vedenjske arhitekture delovanja posameznega agenta. V zadnjem podpoglavju pa je predstavljen večagentni sistem s kooperativnim delovanjem upoštevajoč strategijo igre.

Izvedba algoritmov za različne vloge

Za izvedbo želenega delovanja agentov je potrebno napisati ustrezne algoritme vodenja. Ti algoritmi bodo npr. napadalcu omogočili, da se žogi približa s prave strani in jo potisne v smeri gola. Podobno se mora vratar voziti v liniji pred golom in prestreči strele na gol. V nadaljevanju predstavimo nekaj osnovnih vlog uporabljenih v igri robotskega nogometa.

Vloga napadalec

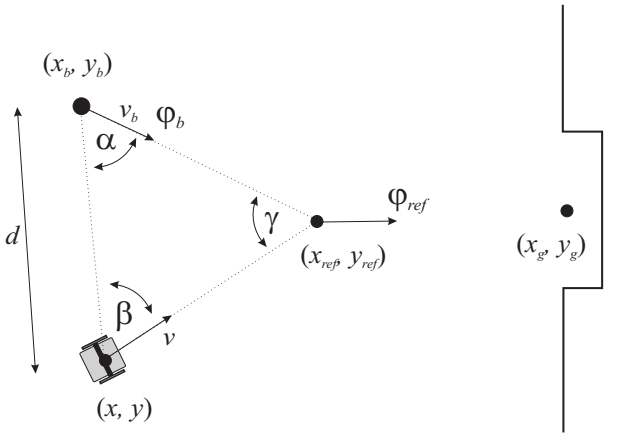
Osnovni način delovanja napadalca (oz. agenta z vlogo napadalec) je, da pride do žoge in jo potisne v smeri gola. Torej referenčna lokacija vsebuje pozicijo žoge (x_{ref},y_{ref}) in referenčno orientacijo (φ_{ref}), ki je v smeri želene smeri gibanja žoge po trku (glejte sliko 7.9).

Za strel proti golu (x_g, y_g) definiramo referenčno orientacijo kot

$$\varphi_{ref} = \text{atan2}\,(y_g - y_{ref}, x_g - x_{ref})$$

kjer je $\text{atan2}\,(y, x)$ štirikvadrantna razširitev funkcije arctan $\frac{y}{x}$. Za izvedbo vodenja lahko uporabimo enega od prikazanih algortimov v poglavju 3.2.3.

V primeru, ko se žoga premika, je vodenje proti trenutni poziciji žoge (x_b, y_b), kot ga prikazuje slika 7.9, manj učinkovito. Boljše delovanje lahko dosežemo, če



Slika 7.10: Predikcija gibanja žoge in ocena točke, kjer robot žogo lahko prestreže. Predpostavimo premočrtno gibanje robota in konstantno hitrost v.

izračunamo prihodnjo pozicijo žoge, kjer jo prestreže. V splošnem je izračun predikcije žoge za poljubno gibanje robota zahteven problem. Zato tu predpostavimo, da se robot lahko giblje premočrtno, kot prikazuje slika 7.10.

Relativni kot α med smerjo kotaljenja žoge in povezavo z robotom lahko ocenimo iz skalarnega produkta

$$\frac{[v_b \cos\varphi_b,\ v_b \sin\varphi_b]^T \cdot [x - x_b,\ y - y_b]^T}{v_b\sqrt{(x-x_b)^2 + (y-y_b)^2}} = \cos\alpha$$

Nadalje z uporabo sinusnega izreka $\frac{v_b}{\sin\beta} = \frac{v}{\sin\alpha}$ določimo še kot β in kot $\gamma = \pi - \alpha - \beta$. Končno lahko z uporabo sinusnega izreka $\frac{d}{\sin\gamma} = \frac{v_b t}{\sin\beta}$ izračunamo potreben čas, ki ga robot rabi, da doseže predvideno pozicijo žoge (x_{ref}, y_{ref}).

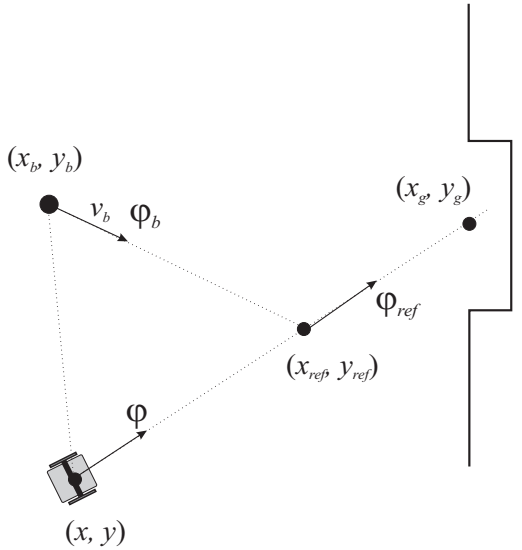
$$t = \frac{d \sin\beta}{v_b \sin\gamma}$$

Predikcija pozicija žoge oziroma referenčna pozicija je

$$x_{ref} = x_b + v_b t \cos\varphi_b$$
$$y_{ref} = y_b + v_b t \sin\varphi_b$$

Izračunana referenčna točka bo dosegljiva le, če bo robot na začetku usmerjen proti žogi in bo hkrati želena smer strela žoge φ_{ref} podobna začetni orientaciji robota φ. Slednje je redko izpolnjeno saj je dejanska pot robota večinoma daljša od direktne linije, ki jo pri izračunu predpostavimo. Robot v splošnem tudi ni obrnjen proti predvideni referenčni točki in končna smer strela mora biti v smeri gola. Upoštevajoč približno oceno dejanske razdelje vožnje l do referenčne lege x_{ref}, y_{ref}, φ_{ref} lahko prilagodimo hitrost vožnje robota $v_n = \frac{l}{t}$, da ta doseže izračunano predikcijo žoge v predvidenem času t.

Vloga napadalec s premočrtnim enakomerno pospešenim strelom



Slika 7.11: Premočrtni, enakomerno pospešeni strel na gol. Robot na začetku miruje in je usmerjen proti golu. Potrebno pospeševanje in predikcijo pozicije žoge je potrebno izračunati.

Gibajočo žogo lahko natančno ustrelimo na gol, če je začetna orientacija robota v smeri gola x_g, y_g in je gibanje robota enakomerno pospešeno. Razmere so prikazane na sliki 7.11. Neznano referenčno pozicijo (x_{ref}, y_{ref}) lahko izračunamo iz trenutne znane pozicije žoge in njenega gibanja (hitrost v_b in smer φ_b)

$$\begin{aligned} x_{ref} &= x_b + v_b t \cos\varphi_b \\ y_{ref} &= y_b + v_b t \sin\varphi_b \end{aligned} \tag{7.1}$$

oziroma jo lahko določimo tudi iz pozicije robota

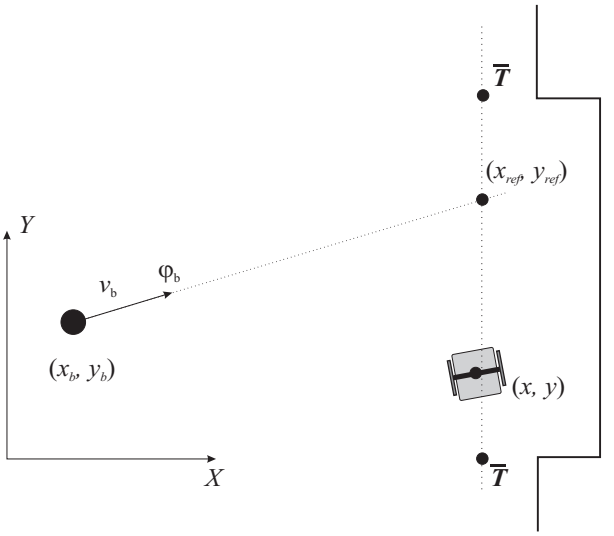
$$\begin{aligned} x_{ref} &= x + (x_g - x)p \\ y_{ref} &= y + (y_g - y)p \end{aligned} \tag{7.2}$$

kjer je t čas, ko robot lahko doseže žogo s predpisanim premočrtnim gibanjem in p je ustrezen parameter. Z upoševanjem enačb (7.1) in (7.2) lahko napišemo sledečo matrično relacijo

$$\begin{bmatrix} x_g - x & -v_b \cos\varphi_b \\ y_g - y & -v_b \sin\varphi_b \end{bmatrix} \begin{bmatrix} p \\ t \end{bmatrix} = \begin{bmatrix} x_b - x \\ y_b - y \end{bmatrix}$$

ki je krajše predstavljena kot $\boldsymbol{Au} = \boldsymbol{b}$. Določimo njeno rešitev $\boldsymbol{u} = \boldsymbol{A}^{-1}\boldsymbol{b}$. Rešitev je veljavna, če je $0 < p < 1$, kar pomeni, da je točka prestrezanja žoge med robotom in golom. Referenčno točko določimo kot

$$\begin{aligned} x_{ref} &= x_b + v_b t \cos\varphi_b \\ y_{ref} &= y_b + v_b t \sin\varphi_b \end{aligned}$$



Slika 7.12: Gibanje robota pri vlogi vratar

in potreben pospešek vožnje $a(t)$ upoštevajoč trenutno hitrost vožnje $v(t)$ določimo iz

$$a(t) = \frac{2\left(\sqrt{(x_{ref}-x)^2+(y_{ref}-y)^2} - v(t)\right)}{t^2}.$$

V kolikor je vloga uspešna ($0 < p < 1$) robotu ukažemo novo translatorno hitrost gibanja $v\left((k+1)\,T_s\right) = v(kT_s) + a(kT_s)T_s$, kjer je $v((k+1)Ts)$ hitrost v naslednjem trenutku, $v(kT_s)$ trenutna hitrost, T_s je računski korak in k je naravno število. Kotno hitrost robota vodimo, da se ta pelje v želeni smeri $\varphi_{ref} = \arctan\frac{y_g - y}{x_g - x}$.

Vloga vratar

Vratar se se giblje v ravni liniji pred golom med točkama $\overline{\boldsymbol{T}}$ in $\underline{\boldsymbol{T}}$ kot je prikazano na sliki 7.12. Trenutno referenčno pozicijo x_{ref}, y_{ref} je potrebno določiti glede na pozicijo žoge x_b, y_b, njeno hitrostjo v_b in smerjo kotaljenja φ_b. Robot se mora prediktivno premakniti na pozicijo kjer bo žoga prečkala črtkano črto pred golom. Iz znanih podatkov določimo čas, v katerem bo žoga prečkala linijo gola in referenčno pozicijo

$$\begin{aligned} t &= \frac{x_g - x_b}{v_b \cos\varphi_b} \\ x_{ref} &= x_g \\ y_{ref} &= y_b + v_b t \sin\varphi_b \\ \varphi_{ref} &= \frac{\pi}{2} \end{aligned}$$

kjer je x_g koordinata linije gola.

Za vožnjo po liniji gola lahko uporabimo linearen algoritem vodenja za sledenje trajektoriji predstavljen v poglavju 3.3.5. Kjer nastavimo referenčne hitrosti (v_{ref}, ω_{ref}) za referenčno pozicijo na nič. Regulacijski zakon se tako poenostavi v

$$\begin{aligned} v &= k_x e_x \\ \omega &= k_\varphi e_\varphi + \texttt{sign}(v) k_y e_y \end{aligned}$$

kjer so ojačenja k_x, k_y in k_φ lahko konstantna in določena eksperimentalno. Lokalne pogreške e_x, e_y in e_φ izračunamo z izrazom (3.28), kjer poskrbimo, da je e_φ v območju $-\pi < e_\varphi \leq \pi$.

Delovanje vratarja lahko nadalje izboljšamo, če zagotovimo, da se vratar nikoli ne obrača za več kot $|\frac{\pi}{2}|$. V primeru, ko je $|e_\varphi| > \frac{\pi}{2}$, to dosežemo z vzvratno vožnjo in popravljenim pogreškom orientacije

$$e_\varphi = \begin{cases} e_\varphi - \pi & ; \quad e_\varphi > \frac{\pi}{2} \\ e_\varphi + \pi & ; \quad e_\varphi < -\frac{\pi}{2} \end{cases}$$

Izogibanje oviram

Igrišče je obkroženo z ograjo in v igri sodeluje več igralcev, zato morajo opisane vloge (predvsem napadalec) vsebovati tudi algoritem izogibanja oviram. Tako obnašanje lahko učinkovito dosežemo z uporabo metode potencialnega polja, ki je predstavljena v poglavju 4.2.4.

Arhitektura delovanja posameznega agenta

Za namen lažje razlage predpostavimo, da moštvo sestoji iz treh robotov oziroma agentov. Vsak agent lahko v danem trenutku izbira med naborom različnih vlog (npr. vratar, napadalec, sredinski igralec, itd.). Izbira vlog in s tem delovanje agenta je izvedena z vsebovano shemo, kjer so vloge organizirane po prioritetah, kot je opisano v poglavju 7.4.5. Z izbiro prioritet lahko določamo strategijo igre. Bolj obrambno strategijo dosežemo z večjimi prioritetami obrambnih vlog in obratno, bolj napadalno strategijo dobimo, če imajo višjo prioriteto napadalne vloge.

V danem trenutku glede na trenutno situacijo igre (lega agenta, pozicija žoge, pozicije soigralcev in nasprotnih igralcev) se agent odloči katero vlogo bo izvajal. Agent najprej preverja pogoje za izvedbo bolj prioritetnih vlog, če pogoji niso izpolnjeni, potem preverja manj prioritetne vloge. Primer pogoja za vlogo vratar je razdalja agenta do gola, v kolikor je najbližje golu (med soigralci) je pogoj za izvajanje vloge vratar izpolnjen. Podobno lahko definiramo pogoj za napadalne vloge upoštevajoč razdaljo do žoge.

Posamezne vloge lahko imajo več možnih načinov delovanja. Pri izvajanju vloge vratar se mora agent nahajati pred golom. Če temu ni tako, se mora agent najprej pripeljati pred gol. Če se nahaja pred golom, potem lahko brani gol, izračuna pot žoge in se prediktivno premakne na pozicijo, kjer bo žoga prečkala linijo gola (kot je opisano v podpoglavju 7.5.1).

Podobno lahko agent izvaja napadalne vloge, če je najbližje žogi oziroma jo lahko najhitreje doseže (glede na ostale soigralce). Agent lahko izvaja osnovno vlogo napadalec brez predikcije gibanja žoge, če je hitrost kotaljenja žoge nizka, drugače pa izbere prediktivno delovanje. Če je izpolnjen pogoj za premočrtno pospešen strel, potem izvaja to vlogo, saj je njena uspešnost večja in ima zato nastavljeno višjo prioriteto. Dodatno napadalec preverja možnost trka z ovirami in se jim izogiba. Izogibanje pa ni vselej zaželeno, npr. če ima napadalec žogo v posesti, izogibanje soigralcem ali nasprotnikom ni vselej smiselno.

Tretji agent je sredinski igralec, ki se poskuša pozicionirati na strateške lokacije (preddoločene) in se orientirati proti nasprotnikovem golu. Najprej se mora pripeljati v želeno pozicijo, nato pa še zavrteti v želeno smer proti nasprotnemu golu. Ta vloga je pomembna za potek igre, saj agenti lahko dinamično spreminjajo vloge med igro (vloge niso statično dodeljene). Igralec s to vlogo lahko v naslednjem trenutku, če je izpolnjen pogoj, prevzame vlogo prediktivnega napadalca in izvede pospešen strel na gol.

Večagentna igra, koordinacija in strategija igre

Izvajanje vseh predhodno definiranih vlog z razpoložljivimi agenti lahko rezultira v avtonomno igro robotskega nogometa. Za boljšo učinkovitost moštva pa je potrebno poskrbeti še za koordinacijo izvajanja vlog med agenti. V danem trenutku je namreč lahko vratar le en igralec in tudi ni smiselno, da več agentov hkrati želi prevzeti žogo, saj bi se pri tem ovirali.

Opisane vloge imajo dodeljene prioritete, kar definira želeno strategijo igre (bolj obrambno ali napadalno). Dodatno ima vsaka vloga tudi funkcijo kriterij, s katero lahko posamezen agent preveri njeno učinkovitost v dani situaciji. Kriteriji so lahko enostavni kot na primer: razdalja do žoge, razdalja do gola in podobno. Vsi agenti tako najprej preverijo njihovo učinkovitost izvajanja za najbolj prioritetno vloge (npr. vratar) in se pogajajo za njeno izvajanje. Najbolj uspešen agent lahko prevzame vlogo v naslednjem trenutku igre, ostali pa se potegujejo za manj prioritetne preostale vloge (napadalec, prediktivni napadalec, sredinski igralec in podobno) iz seznama vlog.

Za izvedbo pogajanja, kateri agent lahko v danem trenutku izvaja določeno vlogo, je potrebno zagotoviti medagentno komunikacijo. Agenti si lahko sporočijo kriterije učinkovitosti za posamezne vloge in se koordinirano odločijo za njihovo izvajanje.

Vloge se med igro dinamično dodeljujejo agentom, kot je podrobneje opisano v

[4] in prikazano tudi na dveh izsekih igre s slikama 7.13 and 7.14. V tem primeru igra pet robotov proti petim nasprotnikom in strategija domačega moštva sestoji iz enajstih vlog. Nekatere od njih so zelo podobne predhodno opisanim.

Sliki 7.13 prikazuje daljši prodor agenta 1. Agent se začne približevati žogi z vlogo napadalec (indeks 0) in se nato približuje bodoči točki srečanja z žogo (vloga prediktivni napadalec z indeksom 8). Napad se zaključi tako, da agent preklopi v vlogo kreatorja (indeks 4), kjer robot v kontaktu z žogo pospešeno vozi proti golu. Omeniti je potrebno, da nasprotnikove pozicije na sliki 7.13 ustrezajo času posnetka 12.08 sekund. Zdi se, da bo nasprotnikov igralec blokiral agenta 1 in smer kotaljenja žoge, vendar se ta kasneje premakne proč in tako je imel agent 1 prosto pot v gol. Iz diagrama potekov indeksov vlog na sliki 7.13 lahko pri prvem agentu opazimo kratek skok iz vloge z indeksom 4 (kreator) na vlogo z indeksom 3 (prediktivni napadalec) in nato nazaj. To se je zgodilo zato, ker se je žoga nekoliko bolj oddaljila od agenta 1 in kriterijska funkcija za vlogo 4 ni bila več izpolnjena. V tem primeru je postala aktivna neka druga aktivna vloga (prediktivni napadalec). Nadalje lahko opazimo, da so ostali agenti zavzeli vloge vratarja (agent 2 ,vloga 1) in sredinskih igralcev (agenti 3 do 5, vloga 5).

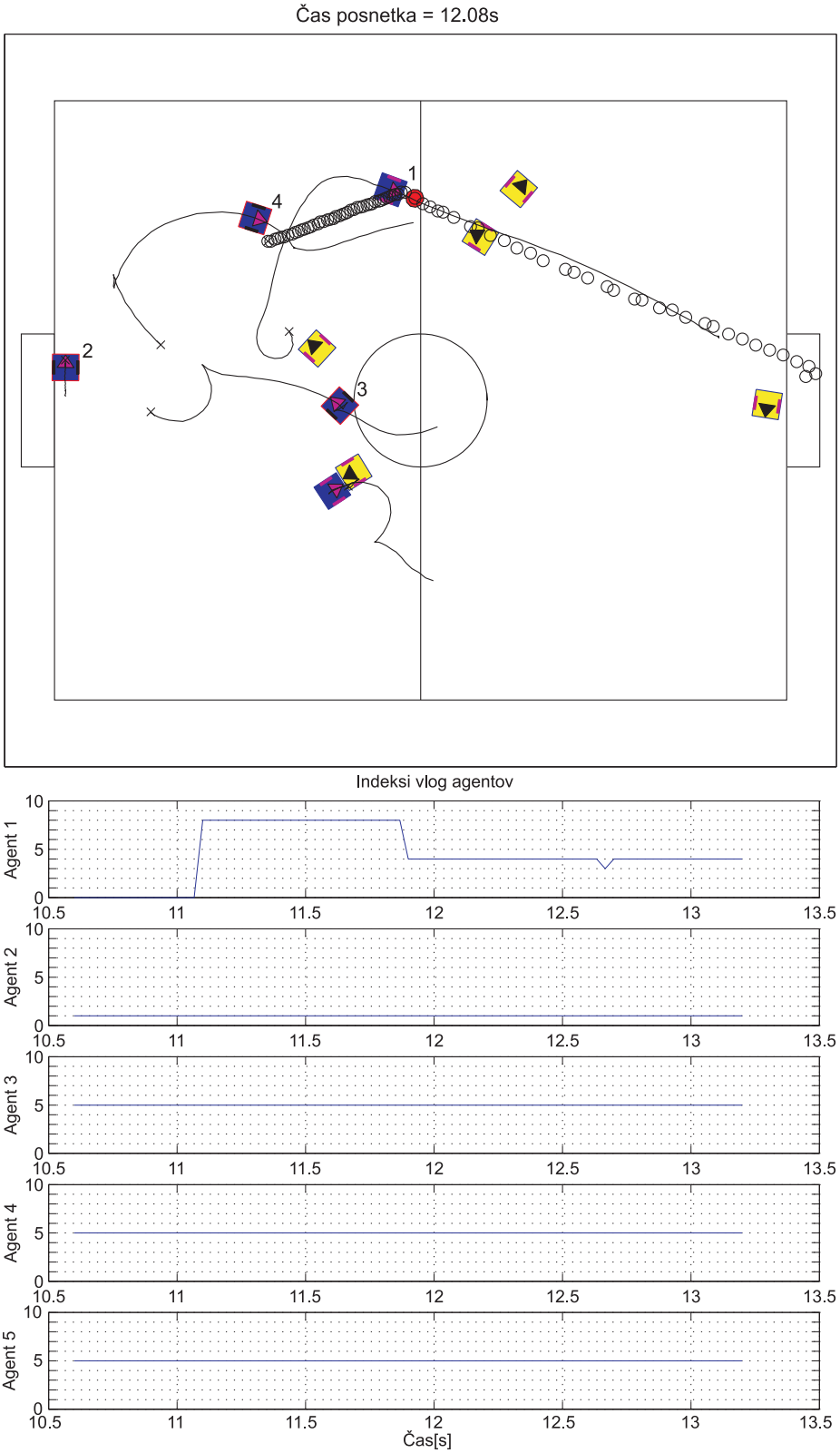
Primer dobre podaje agenta 3 z vlogo napadalec (indeks 0) preko odboja žoge od ograje prikazuje slika 7.14. Po odboju žogo prestreže agent 5 z vlogo napadalca s premočrtnim pospešenim strelom (indeks 6) in akcijo zaključi z vlogo kreatorja (vloga z indeksom 4, kjer robot v kontaktu z žogo pospešeno vozi proti golu).

7.5.2 Vožnja vozil v formaciji

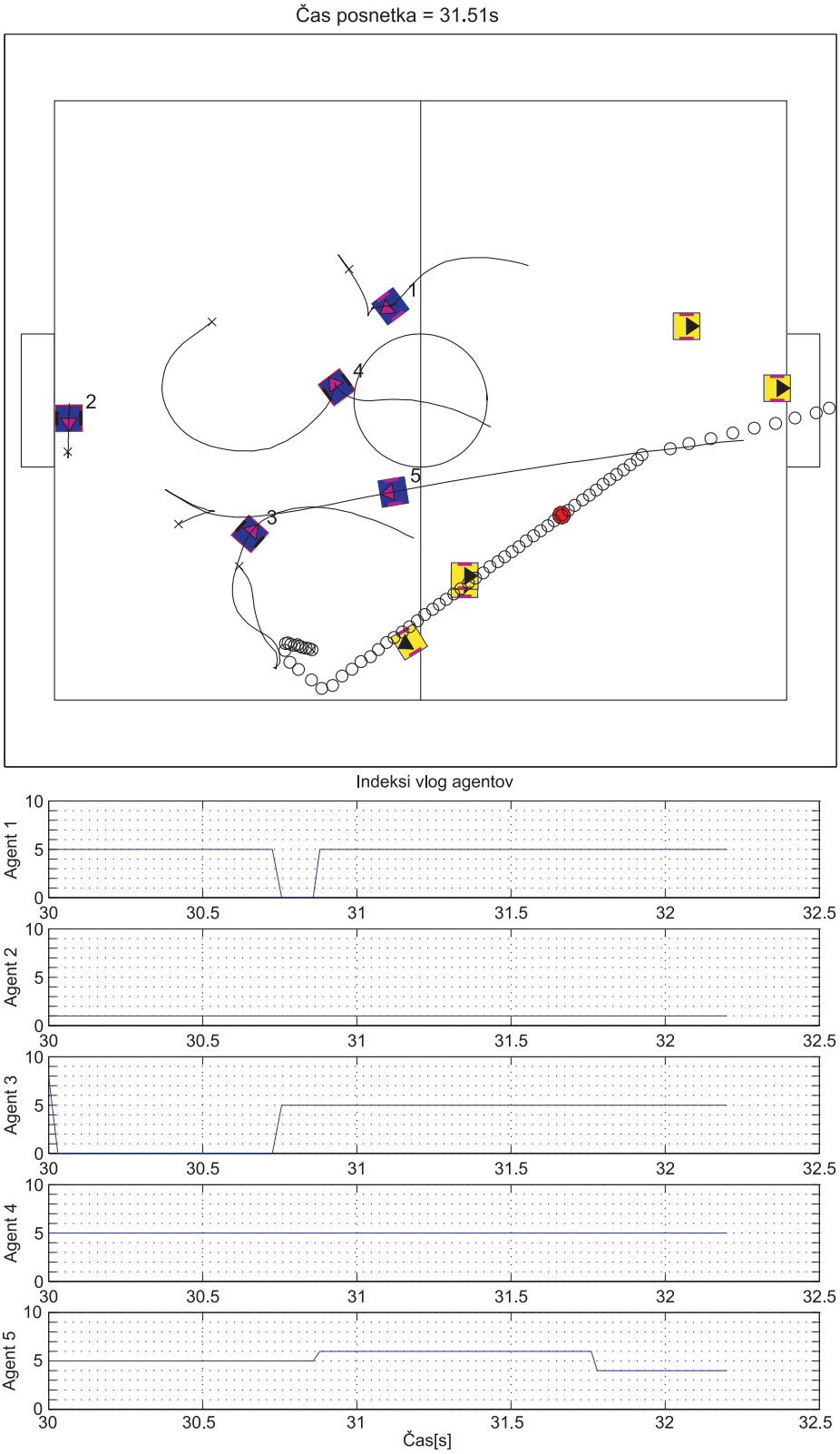
Bodoče inteligentne transportne sisteme si težko predstavljamo brez avtonomnih kolesnih vozil. Pomembna tovrstna aplikacija je avtomatiziran vod (formacija) vozil na avtocestah, ki lahko vozijo avtonomno en za drugim z minimalno varnostno razdaljo kot virtualni vlak. Gre za primer večagentnega sistema. Vozila v formaciji morajo natančno in varno sledi svojemu predhodnemu vozilu z upoštevanjem minimalne varnostne razdalje. Tak pristop bi povečal gostoto transportnih vozil na avtocestah, izboljšal prometne zastoje, pretočnost in varnost.

Prikazan je primer izvedbe algoritma vodenja za mobilna vozila v linearni formaciji. Za avtomatizirano vožnjo potrebujemo natančen senzorski sistem, ki meri globalne informacije vozil (npr. GPS senzorji) ali relativne informacije kot je razdalja in smer med vozili (npr. laserski pregledovalnih razdalj, LRF) oziroma oboje.

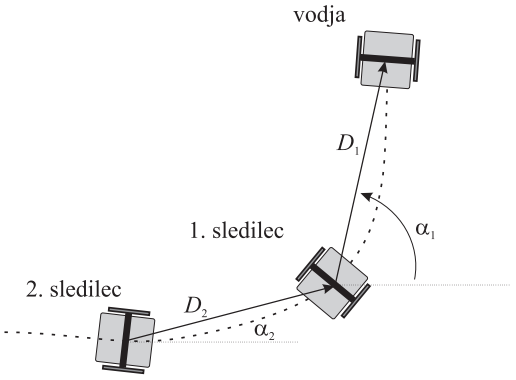
V prikazanem primeru se bomo omejili le na relativne senzorje. Vodenje vozil bo izvedeno decentralizirano, kjer vozila (agenti) upoštevajo le lokalno informacije, ki jih lahko izmerijo s pomočjo laserskega pregledovalnika razdalj (LRF), kot je prikazano v [5]. Predpostavimo, da ima vsako vozilo LRF za merjenje razdalje in azimuta svojega predhodnega vozila. Pot vodilnega vozila se zabeleži v lokalnih koordinatah sledilnega vozila z uporabo odometrije in LRF meritev (razdalje D



Slika 7.13: Primer usklajenega izvajanja vlog v igri robotskega nogometa. Slika prikazuje trajektorije robotov in diagram dinamičnega dodeljevanja vlog.



Slika 7.14: Primer usklajenega izvajanja vlog v igri robotskega nogometa. Slika prikazuje trajektorije robotov in diagram dinamičnega dodeljevanja vlog.



Slika 7.15: Vožnja vozil v linearni formaciji. Vsako sledilno vozilo ocenjuje pot svojega predhodnika in ji sledi s pomočjo algoritma sledenje po trajektoriji.

in smeri α) kot je prikazano na sliki 7.15. Sledilna vozila sledijo ocenjeni poti svojih predhodnikov z uporabo algoritma sledenja trajektoriji, predstavljenega v poglavju 3.3.

Znano je, da je lokalizacija vozila z uporabo odometrije podvržena akumulaciji različnih pogreškov skozi čas (zdrs koles, šum senzorjev in aktuatorjev ter podobno). Torej je uporabnost odometrije omejena le na krajša časovna obdobja, ko je napaka zaradi akumulacije še zanemarljiva. V nadaljevanju je pokazano, da absolutna napaka lege zaradi odometrije ni ključna pri vodenju v linearni formaciji saj je tu pomembna le relativna informacija med vozili (D in α). Slednja je izmerjena z natančnim senzorjem LRF, medtem ko je odometrija uporabljena le za kratko obdobje, da ocenimo odsek poti predhodnega vozila, ki mu sledilno vozilo mora slediti v bližnji prihodnosti.

V nadaljevanju najprej predstavimo tri podsisteme, ki jih kasneje integriramo v končno aplikacijo linearne formacije vozil. Prvi sklop opisuje izvedbo lokalizacije z odometrijo. Drugi sklop opiše oceno trajektorije predhodnega vozila. Tretji sklop pa predstavi algoritem vodenja za sledenje ocenjeni trajektoriji za formacijo. Vsako vozilo predstavlja neodvisnega agenta, ki izvaja omenjene algoritme. Vsi agenti (razen vodilnega) imajo enako obnašanje, s senzorji zbirajo informacije o predhodniku, ocenjujejo njegovo pot in sledijo ocenjeno pot predhodnika na predpisani varnostni razdalji.

Lokalizacija z uporabo odometrije

Odometrija je najpreprostejša metoda za lokalizacijo, kjer lego ocenjujemo z integracijo kinematičnega modela pri znanih hitrosti vozila. Hitrosti vozila z diferencialnim pogonom so znane v diskretnih časovnih trenutkih $t = kT_S$, $k = 0, 1, 2, \ldots$ kjer je T_s čas vzorčenja. Naslednjo lego vozila (pri $(k+1)$) ocenimo

iz trenutne lege (k) in trenutnih hitrosti (glejte poglavje 2.2.1)

$$x(k+1) = x(k) + v(k)T_s\cos(\varphi(k))$$
$$y(k+1) = y(k) + v(k)T_s\sin(\varphi(k))$$
$$\varphi(k+1) = \varphi(k) + \omega(k)T_s$$

Ker za namen sledenja v linearni formaciji začetna absolutna lega vozila ni pomembna jo lahko postavimo kar v začetno (pri $k = 0$) koordinatno izhodišče sledilnega vozila.

Ocena referenčne trajektorije

Vsako sledilno vozilo pozna svojo lego v globalnih koordinatah ocenjeno z odometrijo. Vozilo lahko tako tudi določi lokacijo svojega predhodnika iz poznanih relativnih pozicij med njima. Ta relativna pozicija je izmerjena s pomočjo meritev senzorja LRF in vsebuje razdaljo $D(k)$ in kot do predhodnega vozila $\alpha(k)$. Osnovna ideja je sledenje pozicije predhodnega vozila, ocena njegove trajektorije vožnje in nato lahko sledilno vozilo uporabi to trajektorijo kot referenco in ji sledi.

Lega predhodnega vozila ($x_L(k)$, $y_L(k)$, $\varphi_L(k)$) je ocenjena kot

$$x_L(k) = x(k) + D(k)\cos(\varphi(k) + \alpha(k))$$
$$y_L(k) = y(k) + D(k)\sin(\varphi(k) + \alpha(k))$$

Če potrebujemo pozicijo predhodnega vozila ob določenem času $t \neq kT_s$ jo lahko ocenimo s pomočjo interpolacije

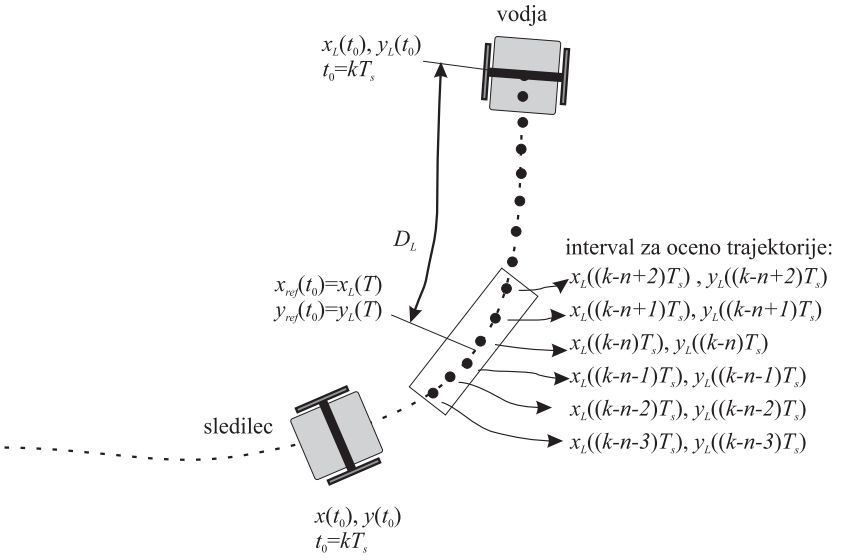
$$x_L(t) = x_L(kT_s) + \frac{t - kT_s}{Ts}\left(x_L\left(k+1\right)T_s - x_L\left(kT_s\right)\right)$$
$$y_L(t) = y_L(kT_s) + \frac{t - kT_s}{Ts}\left(y_L\left(k+1\right)T_s - y_L\left(kT_s\right)\right)$$

Vsako sledilno vozilo mora slediti opravljeni poti svojega predhodnika (vodilnega vozila) na razdalji D_L merjeno po opravljeni trajektoriji vodilnega vozila. Torej moramo ob trenutnem času t_0 oceniti pozicijo vodilnega pri času $t = T$ ($x_L(T)$, $y_L(T)$) tako, da je razdalja med trenutno pozicijo vodilnega robota pri t_0 in prejšnjo pozicija vodilnega robota pri $t = T$ enaka D_L. Ta pozicija predstavlja referenco za sledilno vozilo (glejte sliko 7.16).

Da sledimo opravljeni trajektoriji vodilnega vozila mora sledilno vozilo poznati referenčno trajektorijo in ne zgolj trenutno referenčno pozicijo. Trajektorija vodilnega vozila je ocenjena v parametrični polinomski obliki

$$\hat{x}_L(t) = a_2^x t^2 + a_1^x t + a_0^x$$
$$\hat{y}_L(t) = a_2^y t^2 + a_1^y t + a_0^y$$

upoštevajoč šest pozicij vodilnega vozila, tri pred in tri za referenčno pozicijo, kot je prikazano na sliki 7.16. Koeficienti polinoma a_i^x in a_i^y $(i = 0, 1, 2)$ so ocenjeni z metodo najmanjših kvadratov.



Slika 7.16: Sledilno vozilo mora slediti predhodno vozilo na razdalji D_L vzdolž trajektorije. Oblika referenčne trajektorije v okolici trenutne referenčne lege $x_{ref}(t_0)$, $y_{ref}(t_0)$ je ocenjeno s pomočjo šestih okoliških točk okrog referenčne točke.

Referenčna pozicija za sledilno vozilo ob trenutnem času t_0 je ocenjena z

$$\begin{bmatrix} x_{ref}(t_0) \\ y_{ref}(t_0) \\ \varphi_{ref}(t_0) \end{bmatrix} = \begin{bmatrix} \hat{x}_L(T) \\ \hat{y}_L(T) \\ \hat{\varphi}_L(T) \end{bmatrix} = \begin{bmatrix} a_2^x T^2 + a_1^x T + a_0^x \\ a_2^y T^2 + a_1^y T + a_0^y \\ \text{atan2}\,(2a_2^y T + a_1^y, 2a_2^x T + a_1^x) \end{bmatrix} \tag{7.3}$$

kjer je $\text{atan2}\,(y, x)$ štirikvadrantna razširitev funkcije $\arctan \frac{y}{x}$.

Vodenje vozil v linearni formaciji

Vsako sledilno vozilo mora oceniti in slediti referenčno trajektorijo (7.3) z uporabo nelinearnega regulatorja predstavljenega v poglavju 3.3.6 kot sledi

$$\begin{aligned} v_{fb} &= v_{ref} \cos e_\varphi + k_x e_x \\ \omega_{fb} &= \omega_{ref} + k_y v_{ref} \frac{\sin e_\varphi}{e_\varphi} e_y + k_\varphi e_\varphi \end{aligned}$$

kjer so predkrmilni signali (glejte poglavje 3.3.2) določeni iz ocenjene referenčne trajektorije (7.3) kot

$$v_{ref}(t_0) = \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} = \sqrt{(2a_2^x T + a_1^x)^2 + (2a_2^y T + a_1^y)^2}$$

in

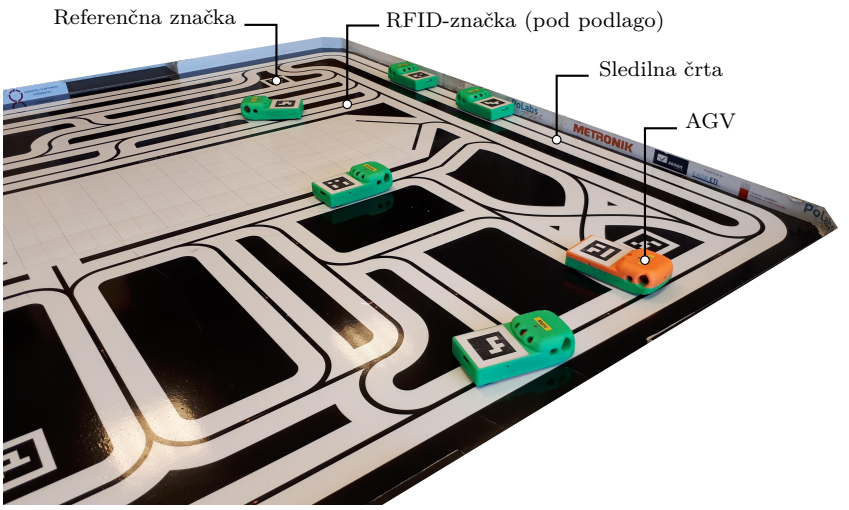
$$\omega_{ref}(t_0) = \frac{\dot{x}_{ref}(t)\ddot{y}_{ref}(t) - \dot{y}_{ref}(t)\ddot{x}_{ref}(t)}{\dot{x}_{ref}^2(t) + \dot{y}_{ref}^2(t)} = \frac{(2a_2^x T + a_1^x)2a_2^y - (2a_2^y T + a_1^y)2a_2^x}{(2a_2^x T + a_1^x)^2 + (2a_2^y T + a_1^y)^2}.$$

Sledilni pogrešek pa je izračunan upoštevajoč dejansko lego sledilnega vozila $(x(t_0),\ y(t_0),\ \varphi(t_0))$ in referenčne lege $(x_{ref}(t_0),\ y_{ref}(t_0),\ \varphi_{ref}(t_0))$ z enačbo (3.28).

7.5.3 Avtomatsko vodeni vozički

Večagentni sistem so tudi avtomatsko vodeni vozički (AGV, angl. *automated guided vehicle*), ki jih dandanes srečamo v mnogih modernih industrijskih halah. Gre za floto mobilnih vozil, ki se avtonomno gibljejo in opravljajo naloge, ki so jim dodeljene. Pogosto se uporabljajo za razvoz materiala in/ali izdelkov. V ta namen imajo AGV-ji lahko vgrajen tovorni prostor, pogosteje pa imajo le posebne mehanizme (običajno preproste, lahko pa tudi dvižne vilice ali celo robotske roke), ki omogočajo prijemanje in odlaganje tovora. To lahko storijo tako, da se pripeljejo pred/pod pasivni mobilni voziček, ga zapnejo in nato vlečejo. Ali tako, da pridejo do standardiziranega zabojnika (npr. paleta), ki ga dvignejo, prepeljejo in nato odložijo. Lahko pa so mehanizmi ali robotske roke za natovarjanje/raztovarjanje kar na postajah, med katerimi tovor razvažajo AGV-ji. Gibanje AGV-jev med postajami običajno ni povsem prosto, temveč so postaje med seboj povezane z omrežjem označenih prog, po katerih se AGV-ji lahko gibljejo. Okolje je torej prirejeno za avtomatsko delovanje AGV-jev tako, da so v okolju na primeren način označene proge. Pogosto se v ta namen uporabljajo magnetni trakovi in RFID-značke, ki so vgrajeni v podlago. Lahko se uporabljajo tudi vidne oznake (npr. kontrastne/barvne črte, QR-kode). AGV lahko tako s primernimi senzorji sledi talnim oznakam in tudi določa svoj položaj v omrežju prog na podlagi unikatnih značk, ki se najajajo ob progah. Lahko pa so proge definirane tudi virtualno, če imamo na voljo globalni ali lokalni sistem, ki omogoča lokalizacijo AGV-ja v okolju. AGV-ji imajo vgrajene še dodatne senzorje bližine, ki jim omogočajo varno delovanje in zaustavitev v primeru ovir na poti. Če je v okolju prisoten tudi človek, potem se pogosto zahteva, da je AGV opremljen z varnostnim laserskih merilnikom razdalj.

V nadaljevanju je predstavljen fizični model pomanjšane industrijske hale (slika 7.17) z omrežjem poti po katerih se vozijo miniaturni avtomatsko vodeni vozički. Dimenzije poligona so $2{,}2\,\mathrm{m} \times 1{,}8\,\mathrm{m}$, dimenzije miniaturnega AGV-ja pa $0{,}1\,\mathrm{m} \times 0{,}2\,\mathrm{m} \times 0{,}06\,\mathrm{m}$. Sistem je bil izdelan za raziskave, razvoj in preizkušanje algoritmov, ki omogočajo avtonomno delovanje AGV-jev, ter za pedagoške namene. Pri izdelavi miniaturnih AGV-jev nismo zahtevali natančne preslikave dejanske situacije iz industrijskega okolja, saj vseh sistemov (npr. laserskega merilnika razdalj, pogonskega mehanizma) ni moč primerno pomanjšati. S fizičnim modelom tako posnemamo le tiste lastnosti avtomatsko vodenih vozičkov, ki so potrebne za učenje in razvoj algoritmov za avtonomno vožnjo. Celoten sistem smo zasnovali tako, da je možna izvedba in študija različnih algoritmov za avtonomno delovanje mobilnih vozičkov: odometrija, vodenje po poti, načrtovanje poti med poljubnimi cilji, iskanje alternativnih poti in obvozov v primeru zastojev, lokalizacija v znanem zemljevidu okolja, večagentno vodenje in podobno. Zaradi



Slika 7.17: Fizični model industrijske hale z AGV-ji

boljše povezljivost med različnimi sistemi in modularnosti smo se odločili, da bomo uporabili okolje ROS (angl. *Robot operating system*).

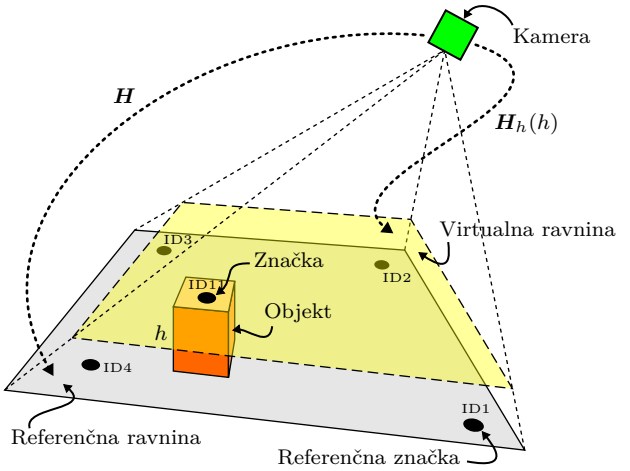
Globalni sistem za merjenje lege s strojnim vidom

Nad poligonom z miniaturnimi AGV-ji se nahaja kamera, ki omogoča sledenje vseh objektov, ki se gibljejo v ravnini poligona. Lega kamere glede na poligon je poljubna, dokler so vsi objekti vidni v njenem vidnem polju. Predpostavimo, da je lega kamere glede na globalni koordinatni sistem (glede na poligon) podana z rotacijsko matriko $\boldsymbol{R} = [\boldsymbol{r}_1,\ \boldsymbol{r}_2,\ \boldsymbol{r}_3]$ in translacijskih vektorjem $\boldsymbol{t}$. Če uporabimo model kamere z luknjico, je povezava med homogeno točko $\boldsymbol{p}_W^T = [x_W,\ y_W,\ 1]$ v poljubni globalni ravnini in homogeno točko na sliki $\boldsymbol{p}_P^T = [x_P,\ y_P,\ 1]$ podana z

$$\boldsymbol{p}_P \propto \boldsymbol{S} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix} \boldsymbol{p}_W = \boldsymbol{H}\boldsymbol{p}_W \tag{7.4}$$

Matrika $\boldsymbol{S}$ v (7.4) vsebuje notranje parametre kamere (glejte poglavje 5.2.4). Matriko $\boldsymbol{S}$ lahko določimo s postopkom kalibracije kamere, npr. z uporabo dobro znanega pristopa s šahovnico [6]. Preslikava $\boldsymbol{H}$ v (7.4) je znana kot *homografija* — predstavlja preslikavo med ravninama.

Homografijo $\boldsymbol{H}$ lahko ocenimo na podlagi vsaj štirih parov točk v slikovni in globalni ravnini. Zato vsebuje poligon štiri referenčne značke (glejte sliko 7.17) — lokacija teh značk glede na globalni koordinatni sistem je znana. Vsak AGV (ali drug objekt, katerega lego želimo meriti) mora tudi biti opremljen z unikatno značko, ki ni nujno, da se nahaja v ravnini tal. Med gibanjem AGV-ja po ravni podlagi se značka na njem giblje po virtualni ravnini, ki je vzporedna z ravnino tal. Situacija je prikazana na sliki 7.18. zatorej lege AGV-ja ne moremo oceniti neposredno iz znane homografije $\boldsymbol{H}$. Ker predpostavljamo, da je višina



Slika 7.18: Sistem za merjenje lege s kamero

h, na kateri je nameščena značka glede na podlago, znana, je možno določiti homografijo $\boldsymbol{H}_h(h)$ za to vzporedno ravnino iz znane homografije $\boldsymbol{H}$.

Slika položaja $\boldsymbol{p}_P$ značke, ki se nahaja v vzporedni ravnini glede na referenčno ravnino (tla) na višini h, lahko preslikamo (z ortogonalno projekcijo) v točko $\boldsymbol{p}_W$ v ravnini tal:

$$\boldsymbol{p}_W \propto \boldsymbol{H}_h^{-1}(h)\boldsymbol{p}_P$$

Da lahko ocenimo homografijo $\boldsymbol{H}_h(h)$, moramo poznati notranje parametre kamere $\boldsymbol{S}$

$$\boldsymbol{H}_h(h) = \boldsymbol{S}\begin{bmatrix}\boldsymbol{q}_1 & \boldsymbol{q}_2 & (\boldsymbol{q}_1 \times \boldsymbol{q}_2)\frac{h}{n} + \boldsymbol{q}_3\end{bmatrix} \tag{7.5}$$

kjer lahko koeficient normiranja n določimo kot

$$n = (\|\boldsymbol{q}_1\| + \|\boldsymbol{q}_2\|)/2 \tag{7.6}$$

Vektorji $\boldsymbol{q}_1$ do $\boldsymbol{q}_3$ v enačbah (7.5) in (7.6) so

$$\boldsymbol{S}^{-1}\boldsymbol{H} = \begin{bmatrix}\boldsymbol{q}_1 & \boldsymbol{q}_2 & \boldsymbol{q}_3\end{bmatrix}$$

Če uporabljene značke niso invariantne na rotacijo, lahko določimo tudi usmerjenost značke in torej celotno lego (položaj in orientacijo) označenega objekta v globalni ravnini. Predstavljen pristop merjenja lege objektov na ravnini je mogoč za poljubno postavitev kamere, dokler so vsi objekti vidni in v vidnem polju kamere, a točnost in natančnost meritev lahko variira. Sledenje objektov je mogoče izvesti tudi pri premikajoči se kameri, če pri tem ne pride do okluzij nobenih značk. Sicer pa lahko homografijo $\boldsymbol{H}$ ocenimo le kadar so vidne vse oz. vsaj štiri referenčne značke. V kolikor se AGV-ji gibljejo le po ravni podlagi poligona, omogoča predstavljen pristop ocenjevanja leg zelo točne meritve, saj je omejitev gibanja na ravnino upoštevana implicitno. Če je kamera kalibrirana, lahko globino značk ocenimo tudi na podlagi znane velikosti značk. V našem

primeru je globina ocenjena implicitno glede na štiri referenčne značke na ravnini tal. Te značke niso kolinearne in so relativno daleč narazen, kar omogoča, da je ocena homografije $\boldsymbol{H}$ zelo točna.

V primeru močne popačitve slike, zaradi distorzij leč, lahko le-te opišemo z nelinearnim modelom

$$\boldsymbol{p}_P = \boldsymbol{f}(\boldsymbol{p}_P^*) \tag{7.7}$$

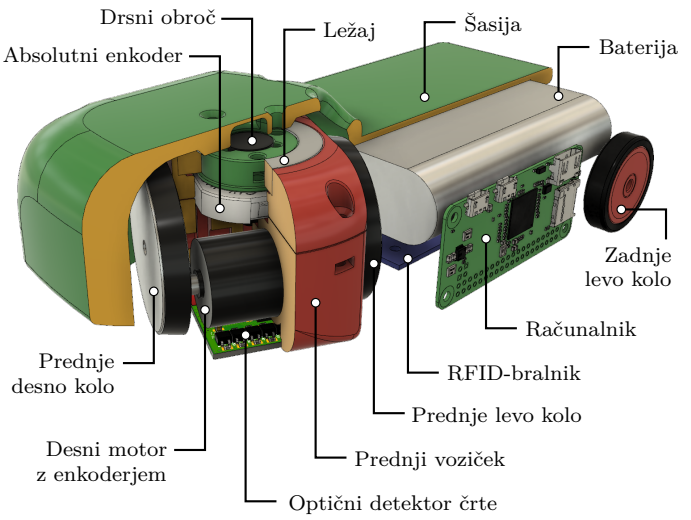
kjer $\boldsymbol{p}_P^*$ predstavlja točko na popačeni sliki in $\boldsymbol{p}_P$ je točka na popravljeni sliki, kjer je popačitev odpravljena. Model distorzije (7.7) lahko ocenimo tekom postopka kalibracije kamere (glejte npr. [7]).

Za detekcijo in sledenje unikatnih značk (s posebnimi črno-belimi vzorci) na sliki smo uporabili knjižnico *ArUco* [8, 9], ki je računsko učinkovita in robustna, torej je primerna za delovanje v realnem času, tudi ob spremenljivi osvetlitvi. Celoten sistem za sledenje več objektov je računsko precej učinkovit in ga je mogoče izvesti na nizkocenovni strojni opremi. V našem primeru smo izvedli celoten sistem na vgradnem računalniku *Raspberry Pi 3 B+* s kamero *Raspberry Pi Camera Board v2*, ki se nahaja približno 2,5 m nad poligonom. V naši izvedbi smo dosegli merjenje lege več objektov v globalnem koordinatnem sistemu s frekvenco 15 Hz. Pri mirujočih objektih smo dosegli standardno deviacijo 0,0001 m pri merjenju položaja in 0,003 rad pri merjenju orientacije (širina značk je 0,1 m). Vse meritve se sproti objavljajo v omrežje ROS in so tako na voljo vsem AGV-jem in ostalim sistemom. Sistem je modularen in precej enostaven za uporabo. Ko imamo na voljo notranje parametre kamere (le-te lahko določimo s postopkom kalibracije, npr. s šahovnico), moramo vnesti le še lokacije štirih referenčnih značk na ravnini tal in izmeriti višine vseh značk na objektih nad ravnino tal.

Miniaturni avtomatsko vodeni vozički

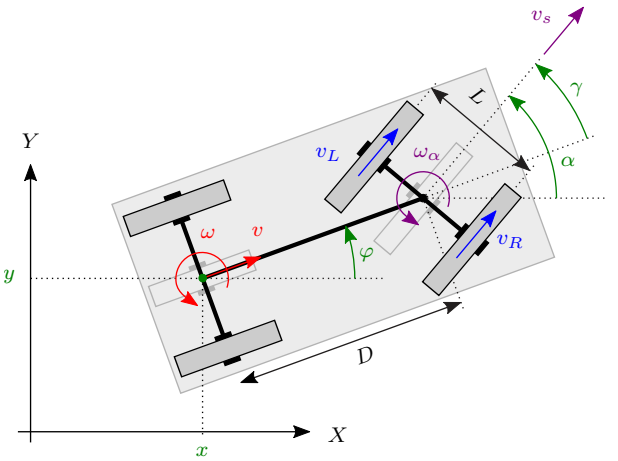
Na sliki 7.19 je prikazana zgradba miniaturnega avtomatsko vodenega vozička z glavnimi sestavnimi deli. Prednji pogonski del predstavlja voziček z diferencialnim pogonom, ki je na šasijo pritrjen pasivno preko ležaja. Takšna oblika mehanskega mehanizma omogoča enostavno izvedbo, pri tem pa še vedno dosežemo želeni kinematični model trikolesnika s prednjim pogonom.

Miniaturni AGV je opremiljen z več senzorji, ki omogočajo implementacijo algoritmov za avtonomno vožnjo. Motorja na prednjem vozičku, ki ženeta obe kolesi, sta opremljena z inkrementalnim enkoderjem. Poleg tega imamo še absolutni enkoder, ki omogoča merjenje kota prednjega vozička glede na šasijo. Na dnu prednjega vozička se nahaja namensko tiskano vezje z mikrokrmilnikom, ki skrbi za nizkonivojsko obdelavo signalov v realnem času in regulacijo hitrosti vrtenja motorjev. Na tiskanem vezju se nahaja tudi sedem segmentni linijski optični detektor črte. Zadnji kolesi sta pasivni in v trenutni izvedbi nista opremljeni z enkoderji. Na spodnjem delu šasije se nahaja še RFID-bralnik. Za obdelavo informacij s senzorjev, komunikacijo z zunanjimi sistemi in izvedbo regulacijskih algoritmov za avtonomno vožnjo se znotraj šasije nahaja računalnik



Slika 7.19: Zgradba miniaturnega AGV-ja v prerezu

Raspberry Pi Zero W. V šasiji se nahaja še baterija. Na vrhu šasije se nahaja posebna unikatna značka s črnobelim vzorcem, ki omogoča, da določimo lego AGV-ja s sistemom za globalno merjenje lege s strojnim vidom (glejte poglavje 7.5.3).



Slika 7.20: Štirikolesni robot s kinematičnim modelom bicikla

Štirikolesni robot na sliki 7.19 ima enake kinematične omejitve kot bicikel (slika 7.20). Omejitve so posledica dejstva, da se kolesa (brez spodrsavanja) ne morejo gibati v smeri osi rotacije

$$\begin{aligned} \dot{x}\sin\varphi - \dot{y}\cos\varphi &= 0 \\ \dot{x}\sin\alpha - \dot{y}\cos\alpha - D\dot{\varphi}\cos\gamma &= 0 \end{aligned} \tag{7.8}$$

V enačbi (7.8) je D razdalja med osjo zadnjih koles in centrom rotacije vozička, na katerega sta pritrjeni prednji kolesi. Vpeljimo posplošeni vektor stanj $\boldsymbol{q}^T = [x,\ y,\ \varphi,\ \gamma]$. Kinematične omejitve lahko tako zapišemo v vrstice t. i. omejitvene matrike $\boldsymbol{A}(\boldsymbol{q})$:

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \begin{bmatrix} \sin\varphi & -\cos\varphi & 0 & 0 \\ \sin\alpha & -\cos\alpha & -D\cos\gamma & 0 \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{0} \tag{7.9}$$

Kinematični model s hitrostnim vhodom $\boldsymbol{u}$ je

$$\dot{\boldsymbol{q}} = \boldsymbol{S}\boldsymbol{u} \tag{7.10}$$

kjer je matrika $\boldsymbol{S}$ jedro (ničelni prostor) omejitvene matrike $\boldsymbol{A}(\boldsymbol{q})$, torej zadošča enačbi $\boldsymbol{A}(\boldsymbol{q})\boldsymbol{S} = \boldsymbol{0}$.

Za hitrosti vhodnega vektorja $\boldsymbol{u}^T = [u_1,\ u_2]$ lahko izberemo različne veličine: pri prednjem pogonu je $u_1 = v_s$, pri zadnjem pogonu pa je $u_1 = v_s$; kotna hitrost prednjega vozička je lahko podana glede na globalni ($u_2 = \omega_\alpha = \dot{\alpha}$) ali lokalni ($u_2 = \omega_\gamma = \dot{\gamma}$) koordinatni sistem. Različni zapisi kinematičnega modela, ki jih pri tem dobimo, so zbrani v tabeli 7.1. Vsi ti modeli zadostujejo omejitvam, ki so podane v (7.9). Ker so si kinematičnimi modeli med seboj precej podobni, moramo biti pri uporabi pozorni, da ne pride do zmede in napačne uporabe.

Tabela 7.1: Kinematični modeli bicikla $\dot{\boldsymbol{q}} = \boldsymbol{S}\boldsymbol{u}$ glede na hitrostne vhode ($\boldsymbol{q}^T = [x,\ y,\ \varphi,\ \gamma]$)

	Globalna kotna hitrost ω_α	Lokalna kotna hitrost ω_γ
Prednji pogon v_s	$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos\gamma\cos\varphi & 0 \\ \cos\gamma\sin\varphi & 0 \\ -\frac{\sin\gamma}{D} & 0 \\ \frac{\sin\gamma}{D} & 1 \end{bmatrix} \begin{bmatrix} v_s \\ \omega_\alpha \end{bmatrix}$	$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos\gamma\cos\varphi & 0 \\ \cos\gamma\sin\varphi & 0 \\ -\frac{\sin\gamma}{D} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_s \\ \omega_\gamma \end{bmatrix}$
Zadnji pogon v	$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ -\frac{\tan\gamma}{D} & 0 \\ \frac{\tan\gamma}{D} & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega_\alpha \end{bmatrix}$	$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ -\frac{\tan\gamma}{D} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega_\gamma \end{bmatrix}$

V našem primeru je vhodni vektor $\boldsymbol{u}^T = [v_s,\ \omega_\alpha]$ in je kinematični model (7.10) torej (model v prvi vrstici in prvem stolpcu v tabeli 7.1)

$$\dot{\boldsymbol{q}} = \boldsymbol{S}\boldsymbol{u} = \begin{bmatrix} \cos\gamma\cos\varphi & 0 \\ \cos\gamma\sin\varphi & 0 \\ \frac{\sin\gamma}{D} & 0 \\ -\frac{\sin\gamma}{D} & 1 \end{bmatrix} \begin{bmatrix} v_s \\ \omega_\alpha \end{bmatrix}$$

Linearna hitrost v_s in kotna hitrost $\omega_\alpha = \dot{\alpha}$ sta neposredno povezani s hitrostjo

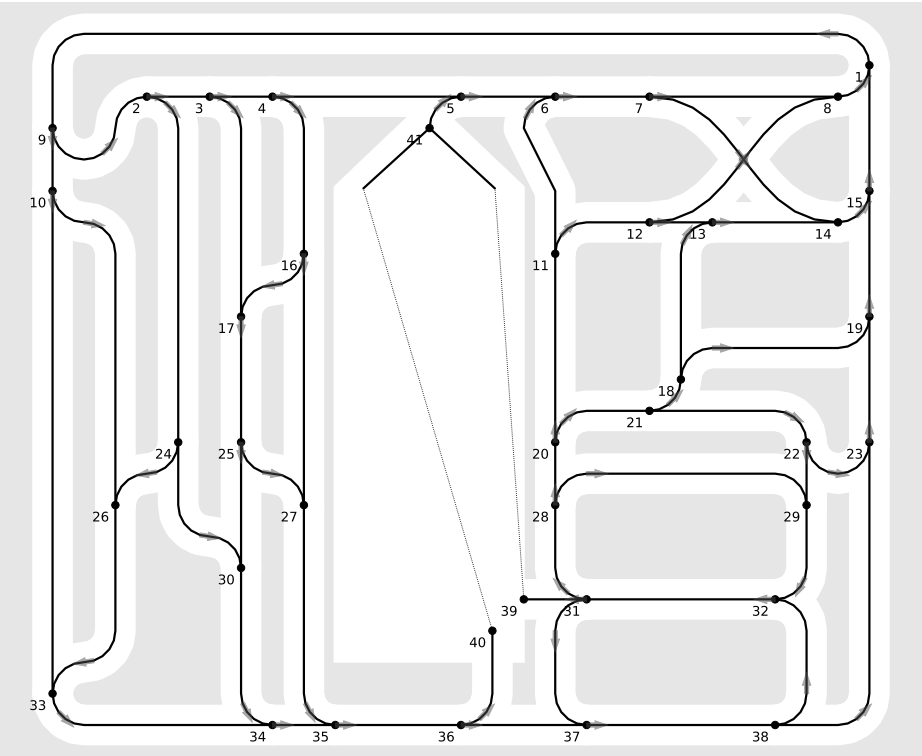
levega kolesa v_L in hitrostjo desnega kolesa v_R

$$v_L = v_s - \frac{L\omega_\alpha}{2} \qquad v_R = v_s + \frac{L\omega_\alpha}{2} \tag{7.11}$$

kjer je L razdalja med prednjima kolesoma.

Omrežje križišč in prog

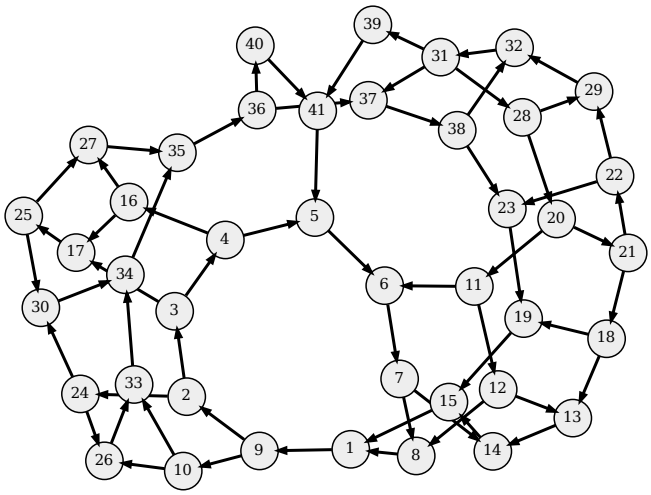
Oglejmo si primer poligona, ki je prikazan na sliki 7.21. Poligon je sestavljen iz omrežja usmerjenih prog, ki se v križiščih razdružijo v več prog ali pa se združijo v eno progo. AGV se lahko vozi le po progah in le v označenih smereh. Izjema je osrednji del poligona, kjer imamo področje brez prog. V tem področju je dovoljeno poljubno gibanje, za kar mora biti robot opremljen s primernimi sistemi za lokalizacijo. V nadaljevanju se bomo posvetili predvsem obravnavi navigacije v delu prostora s progami.



Slika 7.21: Omrežje oštevilčenih križišč in prog

Na sliki 7.21 so vsa križišča oštevilčena. Celoten zemljevid lahko predstavimo z grafom, kjer vozlišča predstavljajo križišča (stik dveh ali več prog) in konce prog. Povezave med vozlišči pa predstavljajo proge, ki povezujejo vozlišča (križišča ali konci prog). Celoten zemljevid prog lahko tako predstavimo v obliki grafa, ki je prikazan na sliki 7.22. Iz grafa so razvidne le povezave med sosednjimi vozlišči, ni pa več vidna oblika poti, ki vodi med križišči. Predstavitev zemljevida z grafom

se izkaže za koristno, saj nam omogoča uporabo različnih algoritmov iz teorije grafov. Uporabimo lahko npr. Dijkstrov algoritem za iskanje optimalne poti — najkrajšo pot dobimo, če povezave med vozlišči utežimo z razdaljami prog.

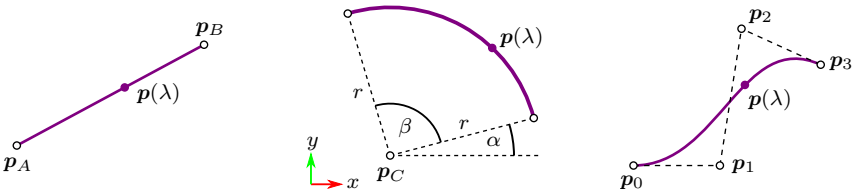


Slika 7.22: Graf križišč in prog

Za prostorsko predstavitev zemljevida moramo poznati položaje vseh križišč in koncev prog (vozlišč) ter obliko vseh prog, ki so v grafu predstavljene s povezavami med vozlišči. Eden izmed možnih načinov zapisa posamezne proge je s pomočjo parametričnih krivulj oz. zlepkov parametričnih krivulj. Na sliki 7.23 so prikazane tri bazične krivulje, ki jih lahko uporabimo za gradnjo zlepka: daljica (slika 7.23a), krožni lok (slika 7.23b) in Bézierjeva krivulja (slika 7.23c). Enačbe in parametri teh krivulj so podani v tabeli 7.2. Uporabimo lahko tudi kakšne druge parametrične krivulje, a že z Bézierjevimi krivuljami lahko aproksimiramo skoraj poljubno krivuljo. Ker pri Bézierjevih krivuljah visokega reda v praksi naletimo na numerične težave, se raje poslužujemo uporabe zlepkov Bézierjevih krivulj nižjega reda. Bézierjeva krivulja prvega reda ($n = 1$) je enaka daljici, kar je razvidno tudi iz enačb v tabeli 7.2. Nikakor pa z Bézierjevo krivuljo (neglede na red) ne moremo natančno opisati krožnega loka — z Bézierjevo krivuljo ali zlepkom Bézierjevih krivulj lahko krožni lok le aproksimiramo s poljubno natančnostjo. Torej bi celoten zemljevid prog lahko opisali tudi le z zlepki Bézierjevih krivulj.

Pomembna lastnost krivulj je tudi njihova dolžina. Za ravninsko krivuljo, ki je podana s $\boldsymbol{p}(\lambda) = [x(\lambda),\ y(\lambda)]$ in parametrom $\lambda \in [0, 1]$, izračunamo njeno dolžino z integralom

$$D = \int_0^1 \sqrt{\left(\frac{\mathrm{d}x(\lambda)}{\mathrm{d}\lambda}\right)^2 + \left(\frac{\mathrm{d}y(\lambda)}{\mathrm{d}\lambda}\right)^2}\,\mathrm{d}\lambda$$



(a) (b) (c)

Slika 7.23: Množica osnovnih krivulj za opis poti: (a) daljica, (b) krožni lok in (c) Bézierjeva krivulja tretjega reda

Tabela 7.2: Množica osnovnih krivulj podanih v parametrični obliki za opis poti

Krivulja	*Enačba in parametri*	*Omejitve*
Daljica	$\boldsymbol{p}(\lambda) = (1-\lambda)\boldsymbol{p}_A + \lambda\boldsymbol{p}_B$ začetna točka $\boldsymbol{p}_A = [x_A,\ y_A]$ končna točka $\boldsymbol{p}_B = [x_B,\ y_B]$	$\lambda \in [0, 1]$
Krožni lok	$\boldsymbol{p}(\lambda) = \boldsymbol{p}_C + r[\cos(\alpha + \lambda\beta),\ \sin(\alpha + \lambda\beta)]$ center $\boldsymbol{p}_C = [x_C,\ y_C]$ radij r začetni kot α ločni kot β	$\lambda \in [0, 1]$ $r \geq 0$ $\lvert\beta\rvert < 2\pi$
Bézierjeva krivulja	$\boldsymbol{p}(\lambda) = \sum_{i=0}^{n} \binom{n}{i} (1-\lambda)^{n-i} \lambda^i \boldsymbol{p}_i$ kontrolna točka $\boldsymbol{p}_i = [x_i,\ y_i]$	$\lambda \in [0, 1]$ $i = 0, 1, \ldots, n$

Definiramo lahko tudi dolžino po krivulji od začetne točke krivulje do točke $\boldsymbol{p}(\lambda)$

$$d(\lambda) = \int_0^{\lambda} \sqrt{\left(\frac{\mathrm{d}x(\xi)}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}y(\xi)}{\mathrm{d}\xi}\right)^2}\,\mathrm{d}\xi \tag{7.12}$$

Torej je celotna dolžina krivulje $D = d(1)$.

Dolžina daljice je enaka evklidski razdalji $d(\lambda) = \lambda\|\boldsymbol{p}_B - \boldsymbol{p}_A\|_2$, dolžina krožnega loka pa je $d(\lambda) = \lambda\beta r$. Medtem ko sta dolžini daljice in krožnega loka lahko določljivi, v splošnem ne obstaja analitična rešitev integrala (7.12) za izračun dolžine splošne Bézierjeve krivulje, razen za krivulje do tretjega reda. Dolžino krivulje v tem primeru zato poiščemo s pomočjo primerne numerične metode. Velja omeniti, da je parameter λ v primeru daljice in krožnega loka proporcionalen dolžini krivulje od začetne točke do točke $\boldsymbol{p}(\lambda)$. To pa v splošnem ne velja za Bézierjeve krivulje, kjer se še vedno z monotonim večanjem parametra λ monotono veča oddaljenost (po krivulji) točke $\boldsymbol{p}(\lambda)$ od začetne točke, a ne linearno.

Z zlepkom, ki je sestavljen iz bazičnih krivulj, lahko aproksimiramo poljubno krivuljo. Glede na zahteve v stičnih točkah poznamo različne načine tvorjenja zlepka, ki je v stičnih točkah lahko le zvezen, lahko pa ima tudi zvezne prve in/ali višje odvode. Tako lahko dosežemo želeno gladkost zlepka v stičnih točkah. Začetna oblika vsake (razen prve) krivulje v zlepku je torej pogojena z obliko predhodne krivulje v zlepku. Zveznost zlepka, ki je sestavljen iz krivulj $\boldsymbol{p}_{i-1}(\lambda)$ in $\boldsymbol{p}_i(\lambda)$, v točki $\boldsymbol{p}_i(0)$ dosežemo z

$$\boldsymbol{p}_i(0) = \boldsymbol{p}_{i-1}(1)$$

Zahtevamo lahko tudi zveznost prvega odvoda koordinat obeh krivulj v stični točki

$$\frac{\mathrm{d}\boldsymbol{p}_i(0)}{\mathrm{d}\lambda} = \frac{\mathrm{d}\boldsymbol{p}_{i-1}(1)}{\mathrm{d}\lambda}$$

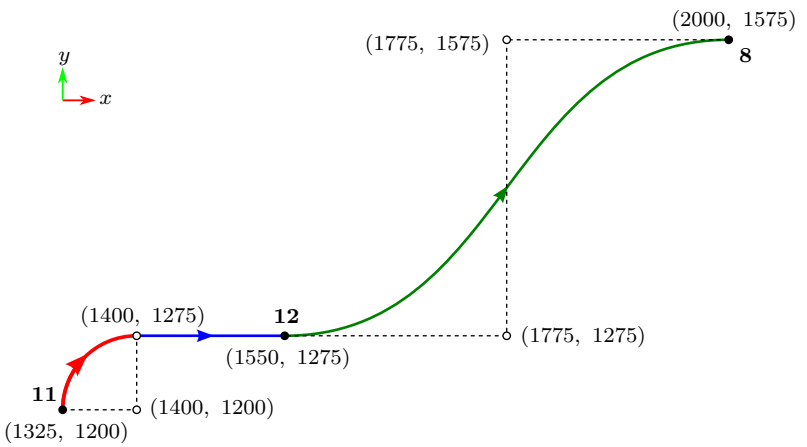
in/ali tudi n-tega odvoda

$$\frac{\mathrm{d}^n\boldsymbol{p}_i(0)}{\mathrm{d}\lambda^n} = \frac{\mathrm{d}^n\boldsymbol{p}_{i-1}(1)}{\mathrm{d}\lambda^n}$$

Zahtevamo lahko tudi zveznost katere druge veličine, kot je usmerjenost ali ukrivljenost krivulje v stični točki.

Na sliki 7.24 je narisana usmerjena pot, ki vodi od vozlišča 11 preko vozlišča 12 do vozlišča 8 (glejte sliko 7.21). Pot lahko opišemo z naslednjim zlepkom bazičnih krivulj (koordinate in razdalje so podane v milimetrih):

1. krožnim lokom s parametri $\boldsymbol{p}_C = [1400,\ 1200]$, $r = 75$, $\alpha = 180°$ in $\beta = -90°$;
2. daljico s parametroma $\boldsymbol{p}_A = [1400,\ 1275]$ in $\boldsymbol{p}_B = [1550,\ 1275]$ ter
3. Bézierjevo krivuljo s parametri $n = 3$, $\boldsymbol{p}_0 = [1550,\ 1275]$, $\boldsymbol{p}_1 = [1775,\ 1275]$, $\boldsymbol{p}_2 = [1775,\ 1575]$ in $\boldsymbol{p}_3 = [2000,\ 1575]$.



Slika 7.24: Pot od vozlišča 11 preko vozlišča 12 do vozlišča 8 (koordinate so v milimetrih)

Vse krivulje so podane tako, da parameter λ vedno narašča v smeri usmerjenosti poti. V danem primeru gre za zvezen zlepek krivulj, ki ima tudi zvezno ukrivljenost v stičnih točkah, kar je za namen vodenja robota po črti smiselno in zaželeno, saj ne želimo, da bi kjerkoli prišlo do prehitre ali celo hipne spremembre usmerjenosti poti. Krivulje lahko združimo tudi kako drugače — npr. tako da so krivulje v stičnih točkah le zvezne, brez zveznih odvodov, kot je to med vozliščema 11 in 6 na sliki 7.21.

Z omenjenimi bazičnimi krivuljami lahko opišemo celoten zemljevid na sliki 7.21. Proga med vozliščema 12 in 8 je podana z eno Bézierjevo krivuljo, medtem ko je proga med vozliščema 11 in 12 podana z zlepkom krožnega loka in daljice. Vsako progo lahko torej opišemo z urejeno množico krivulj

$$proga = (krivulja_1,\ krivulja_2,\ \ldots,\ krivulja_K)$$

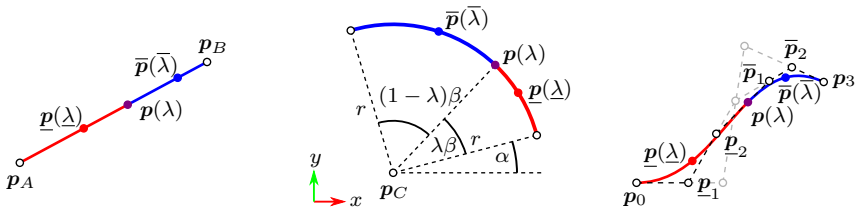
Posamezne proge, ki se stikajo, lahko združimo tako, da tvorimo pot, ki jo zopet lahko zapišemo v obliki urejenega seznama

$$pot = (proga_1,\ proga_2,\ \ldots,\ proga_L)$$

Tako lahko opišemo pot oz. poti med poljubnima vozliščema (križiščema) v zemljevidu.

Če želimo pot, ki je opisana z urejenim seznamom osnovnih parametričnih krivulj, razdeliti na dva ali več delov, moramo znati razdeliti vsako izmed osnovnih krivulj na poljubnem mestu. Razdelitev daljice na mestu λ na daljici $\underline{\boldsymbol{p}}(\underline{\lambda})$ (med točkama $\boldsymbol{p}_A$ in $\boldsymbol{p}(\lambda)$) in $\overline{\boldsymbol{p}}(\overline{\lambda})$ (med točkama $\boldsymbol{p}(\lambda)$ in $\boldsymbol{p}_B$) je enostavna (slika 7.25a):

$$\underline{\boldsymbol{p}}(\underline{\lambda}) = (1-\underline{\lambda})\boldsymbol{p}_A + \underline{\lambda}\boldsymbol{p}(\lambda)\ ; \quad \underline{\lambda} \in [0,\ 1]$$
$$\overline{\boldsymbol{p}}(\overline{\lambda}) = (1-\overline{\lambda})\boldsymbol{p}(\lambda) + \overline{\lambda}\boldsymbol{p}_B\ ; \quad \overline{\lambda} \in [0,\ 1]$$



(a) (b) (c)

Slika 7.25: Množica osnovnih krivulj za opis poti: (a) daljica, (b) krožni lok in (c) Bézierjeva krivulja tretjega reda

Podobno lahko enostavno razdelimo tudi krožni lok (slika 7.25b):

$$\underline{\boldsymbol{p}}(\underline{\lambda}) = \boldsymbol{p}_C + r \begin{bmatrix} \cos(\alpha + \underline{\lambda}\lambda\beta) \\ \sin(\alpha + \underline{\lambda}\lambda\beta) \end{bmatrix}^T \quad ; \quad \underline{\lambda} \in [0,\ 1]$$

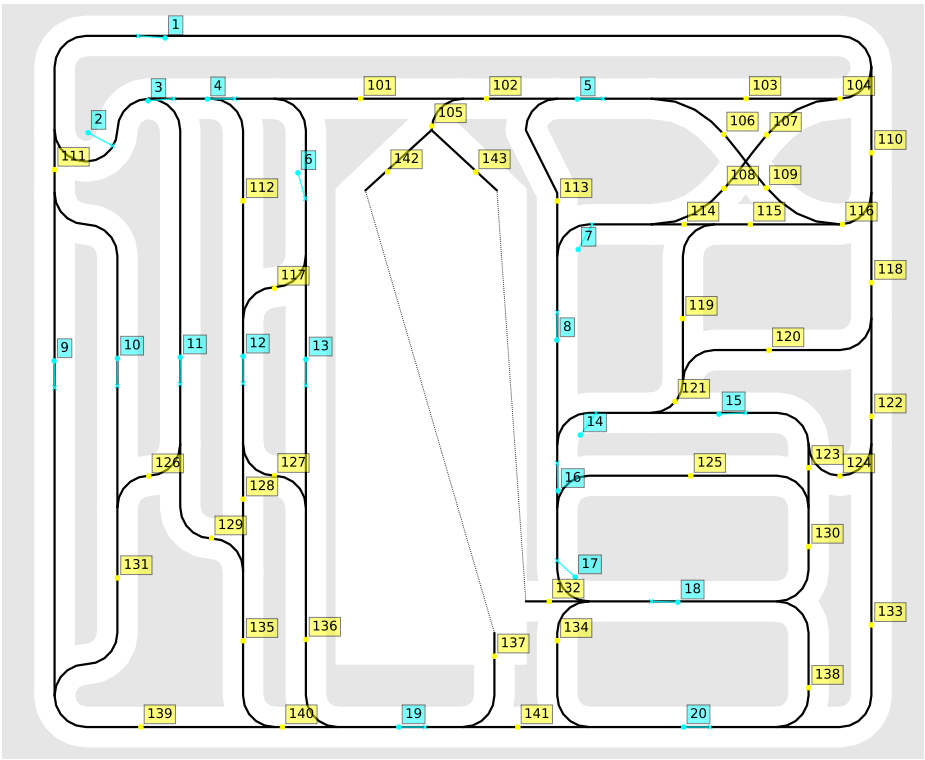
$$\overline{\boldsymbol{p}}(\overline{\lambda}) = \boldsymbol{p}_C + r \begin{bmatrix} \cos(\alpha + \lambda\beta + \underline{\lambda}(1-\lambda)\beta) \\ \sin(\alpha + \lambda\beta + \underline{\lambda}(1-\lambda)\beta) \end{bmatrix}^T \quad ; \quad \overline{\lambda} \in [0,\ 1]$$

Tudi za Bézierjeve krivulje velja, da lahko krivuljo na poljubnem mestu razdelimo na dve krivulji, ki sta zopet Bézierjevi krivulji enakega reda kot osnovna krivulja. Kontrolne točke, ki definirajo novi Bézierjevi krivulji, lahko določimo z de Casteljaujevim algoritmom [10] (slika 7.25c). V vseh primerih velja, da je delitvena točka $\boldsymbol{p}(\lambda)$ na krivulji (daljica/krožni lok/Bézierjeva krivulja) enaka končni točki prve krivulje in začetni točki druge krivulje: $\boldsymbol{p}(\lambda) = \underline{\boldsymbol{p}}(1) = \overline{\boldsymbol{p}}(0)$. To nam omogoča, da zemljevid razširimo z novimi križišči in progami, pri čemer moramo spremeniti le zapis prog, ki se stikajo v novih križiščih.

Omrežje postaj

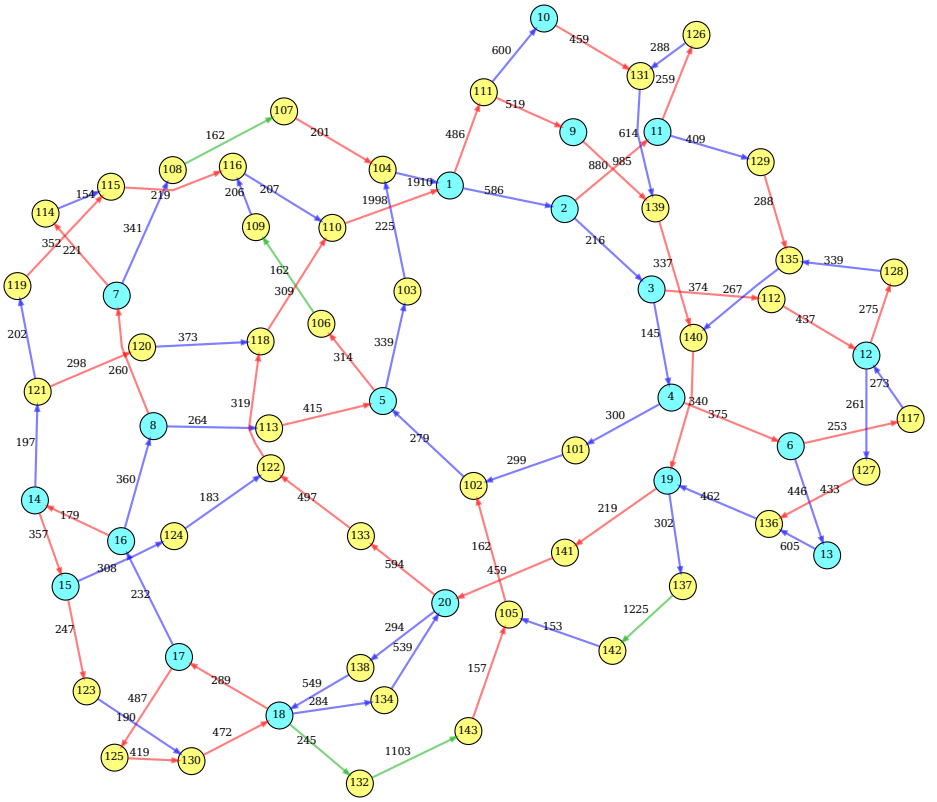
Pod progami se vgrajene RFID-značke z unikatnimi oznakami, ki jih AGV lahko zazna z RFID-bralnikom. Te RFID-značke omogočajo, da AGV ugotovi na kateri progi in kje na progi se nahaja. Če se RFID-značka nahaja pred križiščem, je to primeren trenutek, da se AGV odloči katero smer bo izbral v križišču. Ta izbira je lahko določena fiksno glede na ID značke — npr. v križišču, ki sledi znački z ID-jem 1, pojdi vedno levo. Lahko pa se izbira smeri v križišču določa tudi dinamično glede na pot, ki jo mora AGV opraviti — da gre AGV po želeni poti, mora v križiščih izbrati primerne smeri. Ker pred vsemi križišči ni RFID-značk, mora AGV svojo lego ocenjevati s postopkom lokalizacije (npr. z uporabo RFID-značk in na podlagi odometrije). Za namen planiranja poti in vodenja zato graf križišč in prog predstavimo v nekoliko spremenjeni obliki.

Mesta na progah, kjer se nahajajo RFID-značke, obravnavamo kot postaje, kjer se AGV lahko ustavi oz. se odloči o svoji naslednji akciji — običajno izbiramo le med sledenjem levemu ali desnemu robu črte. Proge, ki ne vsebujejo RFID-značk,



Slika 7.26: Zemljevid z označenimi progami in položaji RFID-značk (sinje modre oznake) ter virtualnih značk (rumene oznake)

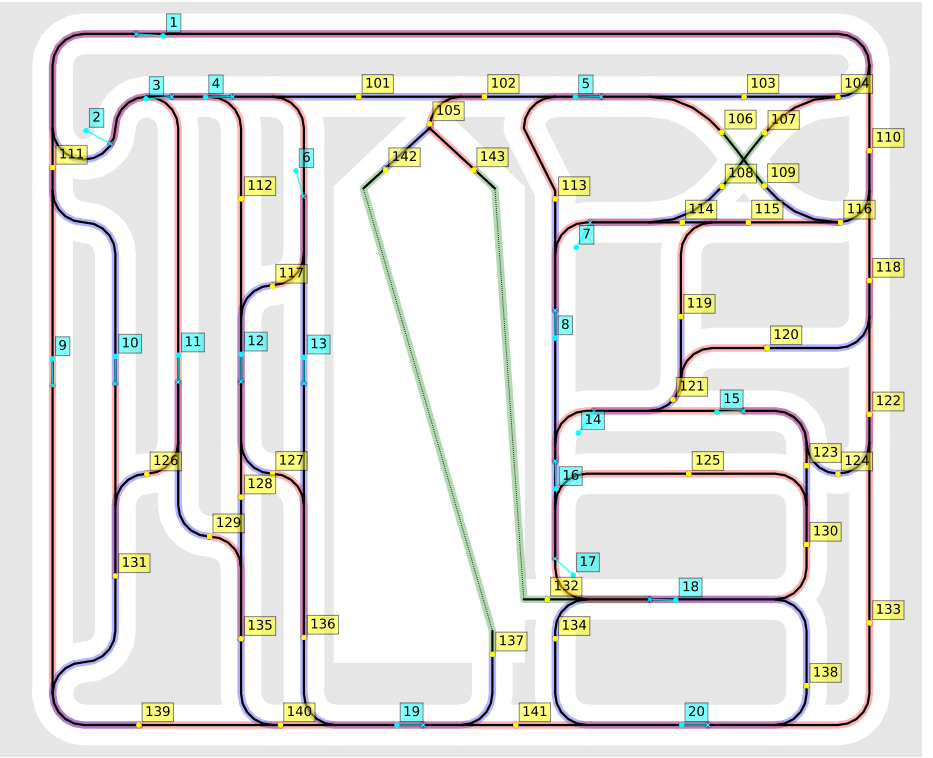
opremimo s t. i. virtualnimi značkami (njihov položaj lahko določimo le na podlagi odometrije/lokalizacije), ki jih tudi obravnavamo kot postaje. Vsaka proga tako vsebuje vsaj eno postajo, ki je označena bodisi z RFID-značko bodisi z virtualno značko. Na sliki 7.26 so na zemljevidu prog označene vse postaje. Postaje z ID-jem, ki je manjši ali enak 100, so označene z RFID-značkami, ostale postaje pa so označene z virtualnimi značkami. Vsaka RFID-značka je na sliki 7.26 označena z dvema točkama: krogec predstavlja dejanski položaj RFID-značke, križec pa lokacijo prednjega vozička AGV-ja, ko le-ta značko zazna. To je posledica dejstva, da je RFID-bralnik izmaknjen glede na točko na AGV-ju, ki potuje po črti.



Slika 7.27: Graf postaj in prog

Graf, ki predstavlja omrežje prog, lahko preslikamo v nov graf, ki predstavlja omrežje postaj. V novem grafu na sliki 7.27 vozlišča predstavljajo vse postaje, ki so povezane z usmerjenimi povezavami, ki predstavljajo poti med postajami. Uteži poleg povezav v grafu na sliki 7.27 predstavljajo razdalje (v milimetrih) med postajami vzdolž prog. Barva usmerjene povezave iz vozlišča določa, kako pridemo do sosednjega vozlišča: modra pomeni, da moramo v križišču izbrati levo smer; rdeča pomeni, da moramo izbrati desno smer; zelena predstavlja poseben način delovanja. Da v križišči izberemo levo ali desno smer, moramo sledili ali levemu ali desnemu robu črte, s katero je označena proga. V tej predstavitvi zemljevida se določeni deli prog med postajami torej prekrivajo. Z upoštevanjem

pravil za delitev krivulj, ki sestavljajo proge, lahko originalni graf z omrežjem križišč avtomatsko pretvorimo v nov graf z omrežjem postaj. Na sliki 7.28 so predstavljene proge, ki povezujejo postaje.



Slika 7.28: Zemljevid z označenimi potmi in položaji RFID-značk (sinje modre oznake) ter virtualnih značk (rumene oznake)

Vodenje po poti

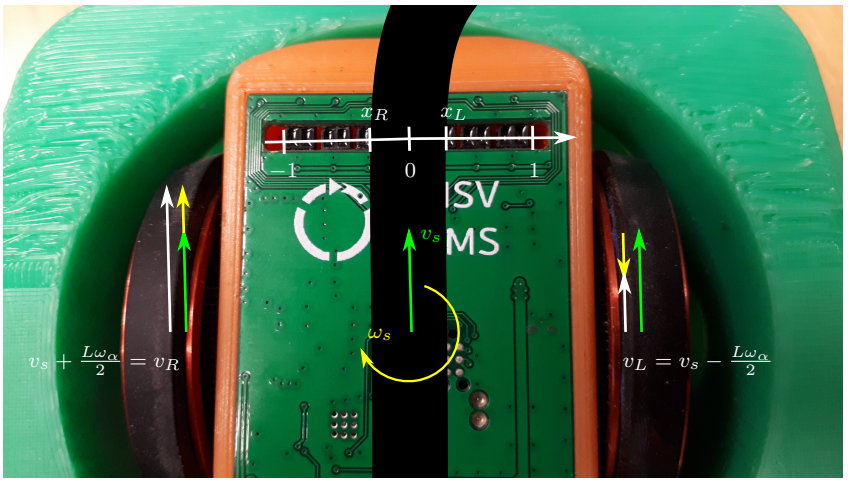
Na podlagi grafa postaj (slika 7.27) lahko torej s pomočjo algoritmov iskanja optimalne poti v grafu (poglavje 4.4) poiščemo pot med poljubnima postajama. Optimalno pot lahko opišemo kot urejen seznam postaj, ki jih moramo obiskati, če želimo priti od začetne do končne postaje. Ta urejeni seznam postaj pa lahko pretvorimo v urejeni seznam akcij, ki jih mora AGV izvesti, da se pelje po želeni poti. Vsako akcijo lahko opišemo s tremi parametri:

1. tip akcije: *levo*, če mora AGV slediti levemu robu črte, *desno*, če mora AGV slediti desnemu robu črte ali *posebno* za vse ostale primere;
2. ID naslednje postaje;
3. razdalja do naslednje postaje vzdolž proge.

Na podlagi tako zapisanih akcij lahko izvedemo vodenje AGV-ja. AGV se torej avtomatsko pelje po poti tako, da sledi črtam, ki označujejo proge, in izbira

primerne smeri v križiščih. Tip akcije vpliva na izbiro regulatorja, ki ga uporabimo za vodenje AGV-ja. V večini primerov imamo v križiščih le dve možnosti. AGV lahko tako gre v želeno smer, če skozi križišče sledi levemu ali desnemu robu črte. Če je (pri trenutni akciji) ID naslednje postaje manjši ali enak 100, to pomeni, da je naslednja postaja označena z RFID-značko. V tem primeru izvajamo trenutno akcijo vse dokler z RFID-bralnikom ne zaznamo ID-ja naslednje postaje. Če je ID postaje večji od 100, potem izvajamo trenutno akcijo toliko časa, dokler AGV vzdolž proge ne prepotuje razdalje do naslednje postaja — ta podatek je vsebovan v akciji. Prepotovano razdaljo lahko ocenjujemo s postopkom odometrije, saj je AGV opremljen z enkoderji, ali globalne lokalizacije.

AGV ima na prednjem vozičku nameščen linijski senzor za zaznavanje črte (slika 7.29). V našem primeru gre za sedem segmentni optični senzor, ki oddaja svetlobo v infrardečem spektru in zaznava količino odbite svetlobe od podlage. Vsak posamezni segment senzorja na beli podlagi vrne nizko vrednost in pri črni podlagi visoko vrednost. Če se senzor nahaja nad črto, lahko na podlagi zaporedja vrednosti senzorja določimo levi in desni rob črte — iščemo prehod iz nizkih k visokim vrednostim oz. obratno. Tako lahko v koordinatnem sistemu senzorja podamo položaj levega roba x_L in položaj desnega roba x_R. Koordinatni sistem smo v našem primeru definirali tako, da je koordinatno izhodišče na sredini senzorja, skrajna robova senzorja pa sta od izhodišča oddaljena za vrednost ena.



Slika 7.29: Sledenje črti na podlagi linijskega senzorja črte (pogled od spodaj)

Glede na želen položaj roba črte na senzorju x_0, lahko definiramo pogrešek: $e(t) = x_0 - x_L(t)$ za sledenje levemu robu in $e(t) = x_0 - x_R(t)$ za sledenje desnemu robu. Če bi želeli slediti sredini črte, bi lahko pogrešek definirali tudi kot $e(t) = x_L(t) + x_R(t)$. Za sledenje levemu ali desnemu robu črte lahko nato zasnujemo preprost regulator za kotno hitrost prednjega vozička

$$\omega_\alpha(t) = K_\omega e(t)$$

kjer je K_ω ojačenje regulatorja. Želen položaj roba črte x_0 nastavimo različno za

sledenje levemu in desnemu robu, tako da pri preklopu med načinom regulacije ne pride do udara regulirne veličine — razlika med želenima vrednostima levega in desnega roba bo tako ravno enaka širini črte. Maksimalna vrednost pogreška je v našem primeru enaka dva ($|e(t)| \leq 2$). Linearno hitrost lahko nastavimo kar na konstantno vrednost ($v_s(t) = v_0 = \text{konst.}$), lahko pa jo tudi moduliramo glede na pogrešek (npr. $v_s(t) = v_0 \cos \frac{\pi e(t)}{4}$), s čimer lahko dosežemo bolj robustno sledenje črti v ovinkih. V kolikor s senzorjem ne moremo zaznati roba črte, ki mu sledimo, regulacijo prekinemo in robota ustavimo.

Linearno hitrost $v_s(t)$ in kotno hitrost $\omega_\alpha(t)$ na podlagi enačbe (7.11) pretvorimo v hitrosti levega in desnega kolesa, $v_L(t)$ in $v_R(t)$. Pri izvedbi regulacije moramo upoštevati še, da sta hitrosti obeh koles omejeni

$$v_{MIN} \leq |v_L(t)| \leq v_{MAX} \qquad v_{MIN} \leq |v_R(t)| \leq v_{MAX} \tag{7.13}$$

sicer se lahko zgodi, da se prednji voziček bodisi ne bo obračal bodisi se bo le obračal na mestu. To lahko rešimo tako, da primerno omejimo tudi hitrosti $v_s(t)$ in $\omega_\alpha(t)$. Lahko pa poskusimo ohraniti razmerje $\frac{\omega_\alpha(t)}{v_s(t)}$, ki predstavlja ukrivljenost, pred in po upoštevanju omejitve (7.13). Kot že omenjeno, za regulacijo hitrosti vrtenja obeh motorjev oz. koles skrbi mikrokrmilnk na prednjem vozičku.

Dodatni virtualni senzorji

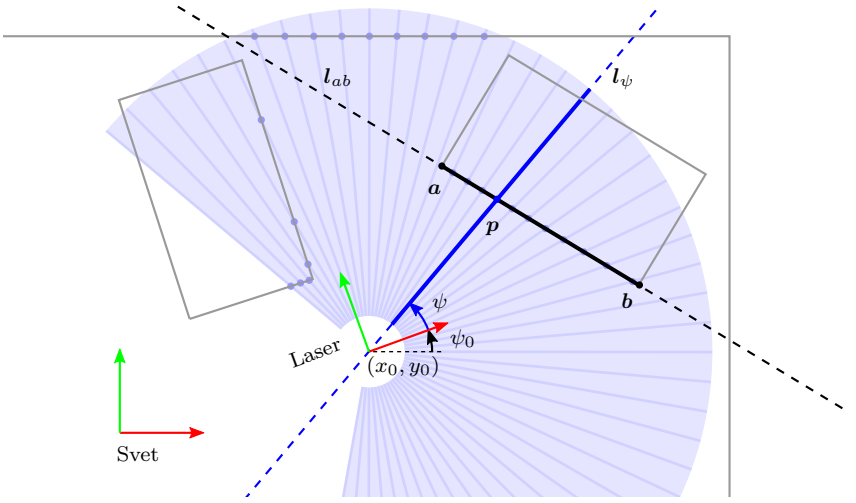
Zaradi majhnosti miniaturnega AGV-ja, smo omejeni z naborom senzorjev, ki jih lahko uporabimo, saj določenih senzorjev, ki se običajno uporabljajo na AGV-jih, ni na voljo v tako majhni izvedbi. Takšen primer je laserski merilnik razdalj (LMR), ki je pri sodelujočih AGV-jih običajno obvezen kot varnostni element. AGV-ji so lahko opremljeni celo z več LMR-ji, ki omogočajo pokrivanje čim večjega vidnega kota. LMR-ji se pogosto uporabljajo za namen lokalizacije in gradnje zemljevida okolja [11].

Čeprav LMR-ja ne moremo primerno pomanjšati, da bi ga vgradili na miniaturni AGV, pa lahko simuliramo meritve tega senzorja. Globalni sistem s strojnim vidom nam omogoča merjenje leg miniaturnih AGV-jev, poznamo pa tudi njihove oblike in obliko poligona. Oblike vseh objektov lahko opišemo z daljicami, nato pa uporabimo algoritem za detekcijo presečišč laserskih žarkov z daljicami ovir. Algoritem se sprehodi čez vse daljice vseh objektov, ki predstavljajo statične ali dinamične ovire, pri čemer za vsako daljico naredimo naslednje (slika 7.30):

1. Če sta $\boldsymbol{a}$ in $\boldsymbol{b}$ robni točki daljice zapisani v homogenih koordinatah, potem je premica skozi ti dve točki $\boldsymbol{l}_{ab} = \boldsymbol{a} \times \boldsymbol{b}$ in enotski vektor daljice $\boldsymbol{e}_{ab} = \frac{\boldsymbol{b}-\boldsymbol{a}}{\|\boldsymbol{b}-\boldsymbol{a}\|}$.
2. Za vsak žarek LMR-ja

   $$\boldsymbol{l}_\psi = \begin{bmatrix} -\sin\psi & \cos\psi & x_0 \sin\psi - y_0 \cos\psi \end{bmatrix}$$

   naredimo naslednje:



Slika 7.30: Modeliranje laserskega merilnika razdalj

(a) Poiščemo presečišče

$$\boldsymbol{p} = \begin{bmatrix} \kappa x & \kappa y & \kappa \end{bmatrix}^T = \boldsymbol{l}_\psi \times \boldsymbol{l}_{ab}$$

žarka s premico daljice in izračunamo faktor $q = \boldsymbol{e}_{ab}^T(\frac{\boldsymbol{p}}{\kappa} - \boldsymbol{a})$.

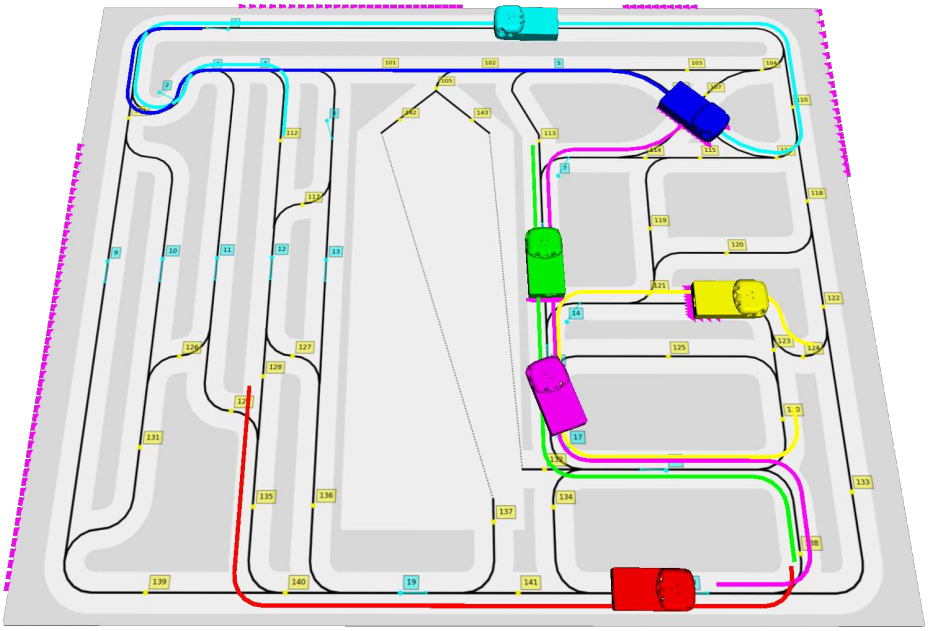
(b) Če velja $0 \leq q \leq 1$, potem presečišče $\boldsymbol{p}$ leži na daljici. V tem primeru izračunamo razdaljo r od objekta do izhodišča laserskega merilnika (v smeri vektorja žarka):

$$r = (\frac{\boldsymbol{p}^T}{\kappa} - \boldsymbol{p}_0^T) \begin{bmatrix} \cos\psi & \sin\psi & 0 \end{bmatrix}^T$$

Če velja $0 \leq r \leq r_{max}$ in če je r tudi manjši od trenutne najkrajše razdalje r_ψ, posodobimo najkrajšo razdaljo: $r_\psi = r$.

Virtualni LMR lahko namestimo glede na katerikoli koordinatni sistem (npr. na prednji del AGV-ja). Nastavimo lahko različne parametre senzorja, kot so doseg, natančnost, vidni kot, kotna ločljivost itd. S stališča podatkov ne moremo ločiti med načinom uporabe realnega ali virtualnega senzorja. Virtualni senzor ne more zaznati objektov, katerih leg ne poznamo oz. jih ne merimo z globalnim sistemom za merjenje lege — poleg AGV-jev lahko z značkami označimo tudi druge objekte na poligonu in tako omogočimo zaznavanje tudi teh.

Na sliki 7.31 je vizualizacija laserskih meritev (vijolične točke), kjer se laserski merilnik razdalj nahaja na vijoličnem AGV-ju. Vidimo odboje laserskih žarkov na zelenem, modrem in rumenem AGV-ju, ne pa tudi na sinje modrem AGV-ju, saj je zasenčen z zelenim AGV-jem; rdeči AGV pa je izven vidnega kota senzorja. Zaradi omejenega dosega senzor ne zaznava objektov, ki so preveč oddaljeni.



Slika 7.31: Vizualizacija meritev LMR-ja, leg AGV-jev in načrtovanih poti

Nekoordinirano večagentno vodenje

Na podlagi algoritmov vodenja, lokalizacije in planiranja poti lahko zagotovimo, da AGV deluje avtonomno, če je edini agent v okolju. V primeru več agentov pa so za primerno delovanje potrebni dodatni sistemi vodenja, sicer lahko prihaja do blokiranj poti, zastojev ali celo trkov in drugih napak, ki lahko zahtevajo tudi ročno ukrepanje operaterja. Za usklajeno delovanje vseh AGV-jev lahko tako skrbi centralni nadzorni sistem, lahko pa imajo AGV-ji lastne sisteme, ki omogočajo razreševanje danih situacij v okolju. Večagentni sistemi so pogosto zasnovani tudi hierarhično, kjer imajo AGV-ji vgrajene sisteme za varno in predvidljivo delovanje, centralni nadzorni sistem pa bedi nad vsemi sistemi in koordinira celotno floto AGV-jev, tako da so vse naloge opravljene optimalno.

AGV-ji lahko z LMR-senzorjem zaznavajo ovire v svoji okolici. Tako lahko AGV določi, če se v njegovi neposredni okolici na predvideni poti nahajajo ovire. V primeru zaznane ovire mora AGV prilagoditi svojo hitrost, da ne pride do trka. Hitrost lahko prilagodijo proporcionalno glede na oddaljenost do najbližje točke možnega trka. Ko se ovira sprosti, lahko nadaljujejo z vožnjo brez omejitev. V primeru, da je zaznana ovira drug AGV, mu AGV lahko sledi na primerni varnostni razdalji, vse dokler potujeta po isti progi. Takšen sistem preprečevanja trkov je enostaven za izvedbo in je v praksi običajno zahtevan kot varnostni mehanizem, ki mora biti tudi ustrezno certificiran.

Na sliki 7.31 so prikazane poti po katerih potujejo AGV-ji. Poti so bile načrtane povsem neodvisno (za vsak AGV ločeno). Miniaturni AGV-ji imamo vgrajen preprost sistem za preprečevanje trkov na podlagi virtualnega LMR-senzorja,

ki je nameščen na prednjem delu AGV-ja. V situaciji na sliki 7.31 bodo rdeči, rumeni, zeleni in sinje modri AGV predvidoma lahko dosegli zadani cilj, brez da bi na poti naleteli na oviro. Modri AGV je že na cilju in miruje, zato vijolični AGV ne bo mogel povsem doseči ciljne postaje, dokler modri AGV te postaje ne zapusti. Vijolični AGV pa se cilju ne bo mogel predvsem približati, saj je na njegovi poti pred njim tudi zeleni AGV, ki bo na svoji končni postaji blokiral še pot vijoličnega AGV-ja.

Pri uporabi le preprostega sistema za preprečevanje trkov hitro pride do situacij, kjer dva AGV-ja pripeljeta v križišče iz različnih smeri skoraj istočasno. Ker AGV-ja zaznavata eden drugega kot oviro, pride do zastoja, ki bi (brez poseganja operaterja ali višjenivojskega nadzornega sistema) trajal neskončno dolgo. Problem lahko rešimo z vpeljavo prednostnih pravil, podobno kot imamo v cestnem prometu desno pravilo in prednostne ceste. Zastoj lahko razrešimo tudi tako, da omogočimo komunikacijo med AGV-jema in vzpostavimo ustrezen sistem pogajanja. Lahko pa vpeljemo v križišča (vsa ali le določena) tudi semaforje. Sistem semaforjev, ki ureja prehode skozi določeno križišče, je tudi agent, ki lahko deluje povsem avtonomno ali pa sodeluje z ostalimi agenti v okolju. Ti agenti so lahko semaforji v drugih križiščih, AGV-ji ali pa tudi centralni nadzorni sistem.

S koordiniranim vodenjem vseh agentov, lahko zagotovimo optimalno delovanje z minimalnim številom zastojev. V kolikor optimiziramo poti za vse AGV-je hkrati, lahko poskusimo doseči tudi, da do zastojev na poti ne prihaja in da lahko vsi AGV-ji dosežejo svoj cilj, če le je končna postaja prosta (glejte poglavje 7.5.4). V primeru na sliki 7.31 bi želeli doseči, da je vijolični AGV pred zelenim — vsaj na delu kjer bi sicer prišlo do zastoja.

7.5.4 Večagentno planiranje vožnje transportnih vozil

Z vse bolj pogosto avtomatizacijo skladišč in proizvodnih obratov je postalo tudi raziskovanje na področju avtonomnih robotskih vozil zelo popularno. Eden bistvenih izziv avtonomnih robotskih vozil, je planiranje poti. To je lahko rešeno centralno, kjer centralna enota, ki povezuje vsa robotska vozila hkrati določi načrt poti za vsa vozila. Glavna prednost centralnih pristopov je večja optimalnost načrtanih poti [12, 13]. Pri bolj obsežnih zemljevidih in večjem številu vozil po navadi postanejo taki pristopi časovno preveč potratni, kar omejuje uporabo v realnih sistemih. Problem računske kompleksnost bolje naslavljajo decentralizirani pristopi, ki so v splošnem hitrejši od centraliziranih, bolj prilagodljivi na spremembe okolja in delovnih nalogov a večinoma zagotavljajo manj optimalne rešitve [14, 15]. V teh pristopih se naloga določanja poti iz višjega nivoja prestavi na sama vozila, tako da avtonomno določajo vsak svojo pot, hkrati pa sprotno rešujejo konflikte in zbirajo informacije o drugih vozilih.

Prikazan je pristop planiranja poti za usklajeno delovanje skupine mobilnih robotskih vozil pri transportu materiala v proizvodnih obratih. Pristop je razvit v okviru magistrske naloge in je objavljen tudi v prispevku [16]. Predstavljen algoritem temelji na znanem algoritmu A*, ki je nadgrajen za planiranje poti več robotskih vozil, tako da najde kompromisno rešitev brez trkov in nepotrebnih zastojev. Pristop upošteva prioritete transportnih nalogov, zemljevid v obliki uteženega usmerjenega grafa ter predvidena časovna okna zasedenosti segmentov zemljevida. Algoritem najprej poišče pot za vozila z višjimi prioritetami. Ob vsakem planiranju poti vozila, algoritem na koncu zabeleži predvidena časovna okna zasedenosti cest in vozlišč na zemljevidu za najdeno pot. Te zasedenosti se nato upoštevajo pri iskanju poti za vozilo z nižjo prioriteto, tako da ne ovira vožnje vozil z višjo prioriteto in se izogne konfliktom. Dve pomembni možnosti, ki jih algoritem upošteva in predlaga, sta čakanje pred vozliščem, da se pot sprosti in pa možnost umika na stransko cesto v primeru onemogočenega čakanja. Pristop je ilustriran na simulacijskih primerih.

Planiranje z upoštevanjem prioritet in oken zasedenosti

Predlagan algoritem je nadgradnja algoritma A* in omogoča planiranje poti za več robotskih vozil, ki se hkrati vozijo na istem zemljevidu. Algoritem se izvaja ločeno za vsako vozilo posebej po prioritetnem sistemu. Vsakemu vozilu določimo stopnjo prioritete. Večja kot je stopnja prioritete, bolj pomembno je, da to vozilo doseže cilj v najkrajšem možnem času. Preden izvedemo algoritem na kateremkoli vozilu, je zemljevid popolnoma prost. Vsa vozlišča in povezave so brez predvidenih časovnih oken zasedenosti. Algoritem najprej izvedemo na problemu vozila z najvišjo prioriteto. Ker je zemljevid še prost, bo algoritem zanj našel optimalno pot. Hkrati bo določil časovna okna zasedenosti za ceste in vozlišča v časovnih trenutkih, ki jih narekuje rezultat planiranja poti. Ko zaženemo algoritem na robotu z nižjo prioriteto, bo z upoštevanjem zasedenosti časovnih oken cest in vozlišč zanj našel čim hitrejšo možno pot, ki ne bo ovirala vozil z višjo prioriteto.

Osnovni potek iskanja poti za posamezno vozilo je enak kot pri algoritmu A*. Algoritem najprej doda začetno vozlišče na odprti seznam. Nato v zanki izvaja iterativni postopek, dokler ni odprti seznam prazen oz. dokler iz odprtega seznama na zaprti seznam ne prestavi ciljnega vozlišča, kar pomeni, da je pot najdena. V iterativnem postopku najprej iz odprtega na zaprti seznam prestavi vozlišče z najmanjšo skupno ceno. Nato za vsako sosednje vozlišče določi cene ter preveri zasedenost vozlišča in pripadajoče ceste. Če je pot prosta, algoritem nadaljuje z enakim potekom kot A*. Glavna razlika med algoritmoma nastopi, kadar algoritem pri odpiranju sosednjega vozlišča in preverjanju zasedenosti naleti na konflikt. Konflikt predstavlja zasedenost vozlišča ali ceste do sosednjega vozlišča. Prisotnost konflikta se ugotavlja s pomočjo časovnih oken zasedenosti vozlišča ali cest, ki jih za svojo pot definirajo vozila z višjo prioriteto. V primeru konflikta se s poizkušanjem išče prosta mesta za čakanje in izogib konfliktu z

vračanjem po poti, po kateri je prišel. Pseudo-algoritem z opisanim postopkom izogibanja konfliktov je podan v algoritmu 6. S tem se začne rekurziven postopek

Algorithm 6 Planiranje poti za več vozil.

Inicializacija:
Dodelitev prioritet transportnim nalogom za N vozil.
Predstavitev prostora z grafom prehajanja stanj in okni zasedenosti.

for vozilo s prioriteto $i = 1, \ldots, N$ **do**
V zanki izvajaj iterativni postopek iskanja poti po algoritmu A*.

V primeru konflikta izvajaj:
while $mestoKonflikta \neq zacetniPolozajVozila$ **do**
if prosto(cesta pred $mestoKonflikta$) **then**
Predlagaj čakanje na cesti pred $mestoKonflikta$.
break
end if
if prosto(stranska cesta pred $mestoKonflikta$) **then**
Predlagaj umik in čakanje na stranski cesti pred $mestoKonflikta$.
break
end if
Prestavi $mestoKonflikta$ na predhodno cesto po poti nazaj.
end while

Določi okna zasedenosti za vozlišča in ceste na poti.
end for

v katerem algoritem s poizkušanjem išče primerno mesto za čakanje na sprostitev poti in izogib konfliktu. Najprej poizkusi s čakanjem na predhodni cesti. Če tudi ta cesta ni prosta, poizkusi s čakanjem na kateri od stranskih cest. Stranske cesta so vse ceste povezane s predhodnim vozliščem, razen predhodne in trenutne ceste. Če nobena od stranskih cest ni prosta za čakanje, se algoritem rekurzivno pomakne po poti nazaj. Algoritem nadaljuje z novo iteracijo, kjer poizkusi s čakanjem na novi predhodni cesti, če ta ni prosta na novih stranskih cestah itd., dokler ne najde prostega mesta ali ne pride do začetka poti.

Določitev cen povezav in oken zasedenosti povezav in vozlišč

Predlagan algoritem pri raziskovanju zemljevida prednostno izbira vozlišča z najmanjšo skupno ceno. Cena povezave predstavlja čas, ki ga vozilo porabi, da prevozi pot med dvema vozliščema. Pri tem predpostavimo konstantno hitrost vozila med premikanjem, ki jo podamo kot vhod algoritma. Predlagan algoritem vpeljuje možnost čakanja vozila na mestu, da se cesta ali vozlišče sprosti, vozilo pa lahko nadaljuje pot brez konfliktov. Čas, da vozilo prevozi neko pot se tako

lahko podaljša s pribitkom cene zaradi čakanja na tej poti. Povečana cena torej predstavlja seštevek časa vožnje in časa čakanja med vozliščema. V splošnem izbira take cene pomeni, da bo algoritem iskal pot, po kateri bo najhitreje prispel do cilja. Ko algoritem najde pot za vozilo ji doda še časovna okna zasedenosti cest in vozlišč, ki jih bo vozilo prevozilo na najdeni poti ob predvidenih časovnih trenutkih.

Teoretično se vozilo na vozlišču nahaja zgolj časovni trenutek. A je za določitev okna zasedenosti vozlišča potrebno upoštevati tudi dimenzije vozila in želeno varnostno razdaljo ter njegovo hitrost vožnje, da se prepreči trk z drugim vozilom, ki pripelje v vozlišče tik za prvim. **Časovno okno zasedenosti vozlišča** tako določa interval $t_{VS} \leq t < t_{VE}$, kjer so $t_{VS} = t_V - \Delta t_{varn}$, $t_{VE} = t_V + \Delta t_{varn}$, t_V čas prihoda vozila v vozlišče, $\Delta t_{varn} = \frac{D}{v}$ varnostni čas, D seštevek polovične dimenzija vozila in varnostne razdalje in v hitrost vožnje.

Časovno okno zasedenosti ceste določajo naslednji podatki: čas prihoda na cesto (t_{vstop}), čas vožnje (Δt_{cest}), čas čakanja (Δt_{cak}) na cesti (pred končnim vozliščem) in smer vožnje. Vozilo lahko čaka le na cestah (povezavah), čakanje v vozliščih ni dovoljeno. Ko čaka na mestu, je to vedno na cesti pred vozliščem. **Časovno okno zasedenosti ceste** tako določa interval $t_{CS} \leq t < t_{CE}$, kjer je upoštevan varnostni čas Δt_{varn} (dimenzije vozila in želena varnostna razdalja), $t_{CS} = t_{vstop} - \Delta t_{varn}$, in morebiten čas čakanja, ki je vključen v $t_{CE} = t_{vstop} + \Delta t_{cest} + \Delta t_{varn} + \Delta t_{cak}$.

Preverjanje zasedenosti in določitev časa čakanja

Preden algoritem doda vozlišče na odprti seznam, preveri če je vozlišče prosto. Pri tem preveri vstopno cesto in vozlišče posebej, saj je lahko zasedena samo cesta ali samo vozlišče. Algoritem najprej preveri zasedenost ceste in nato zasedenost vozlišča. V kolikor algoritem zazna zasedenost ceste ali vozlišča izlušči podatek o koncu zasedenosti ceste oz. vozlišča. Na podlagi tega podatka v nadaljevanju določi čas čakanja vozila. Vozilo lahko čaka na trenutni cesti (cesta pred vozliščem), ko je zasedeno vozlišče oziroma na predhodni cesti trenutne ceste, če je zasedena trenutna cesta.

Preverjanje zasedenosti ceste in določitev časa čakanja na predhodni cesti Algoritem pri preverjanju zasedenosti ceste za trenutno vozilo primerja predvideno časovno okno s časovnimi okni zasedenosti, ki jih je za to cesto prej določil vozilom z višjo prioriteto. Pri tem ločimo časovno okno vozila, ki vozi v nasprotni smeri vožnje trenutnega vozila (z indeksom i) in časovno okno vozila, ki vozi v smeri vožnje trenutnega vozila.

V primeru iste smeri vožnje po cesti je čas čakanja trenutnega vozila (Δt_{cak_i}) pred cesto določen upoštevajoč okna zasedenosti vozil z višjo prioriteto ($t_{CS} \leq t < t_{CE}$)

ter s parametri trenutnega vozila (vstopni čas t_{vstop_i}, varnostni čas t_{varn_i})

$$\Delta t_{cak_i} = \begin{cases} 0 & ; \quad |t_{vstop_i} - t_{CS}| \geq \Delta t_{varn_i} \\ |t_{CS} - t_{vstop_i}| + \Delta t_{varn_i} & ; \quad |t_{vstop_i} - t_{CS}| < \Delta t_{varn_i} \end{cases}$$

Pri izračunu časa čakanja trenutnega vozila Δt_{cak_i} na cesti je potrebno upoštevati tudi morebiten čas čakanja predhodnega vozila z višjo prioriteto. Kjer trenutno vozilo lahko na cesto vstopi brez čakanja pred ali za predhodnikom ($\Delta t_{cak_i} = 0$), če je vsaj za varnostni čas Δt_{varn_i} in za svoj predviden čas čakanja (brez upoštevanja predhodnega čakanja na cesti) pred predhodnikom oziroma, če je vsaj za varnostni čas Δt_{varn_i} in čas čakanja predhodnika za njim. V nasprotnem pa mora svoj čas čakanja ustrezno podaljšati.

V primeru nasprotne vožnje po cesti, pa je čas čakanja trenutnega vozila z indeksom i določen kot

$$\Delta t_{cak_i} = \begin{cases} 0 & ; \quad \begin{array}{l} t_{vstop_i} \geq t_{CE} + \Delta t_{varn_i} \text{ ali} \\ t_{izstop_i} \leq t_{Cs} - \Delta t_{varn_i} \end{array} \\ t_{CE} - t_{vstop_i} + \Delta t_{varn_i} & ; \quad \begin{array}{l} t_{vstop_i} < t_{CE} + \Delta t_{varn_i} \text{ in} \\ t_{vstop_i} > t_{CS} - \Delta t_{varn_i} \end{array} \end{cases}$$

Preverjanje zasedenosti vozlišča in določitev časa čakanja na trenutni cesti Potreben čas čakanja na cesti pred vozliščem (trenutni cesti) za vozilo i določimo glede na okno zasedenosti, ki so ga za to vozlišče določila vozila z višjo prioriteto. Čas čakanja lahko poenostavljeno (brez upoštevanja smeri vožnje skozi vozlišče) določimo kot

$$\Delta t_{cak_i} = \begin{cases} 0 & ; \quad |t_{vstop_i} - t_V| \geq \Delta t_{varn_i} + \Delta t_V \\ \Delta t_{varn_i} + t_{VE} - t_{vstop_i} & ; \quad |t_{vstop_i} - t_V| < \Delta t_{varn_i} + \Delta t_V \end{cases}$$

kjer je $t_V = \frac{t_{VS}+t_{VE}}{2}$ trenutek prihoda (težišča) predhodnega vozila v vozlišče in $\Delta t_V = \frac{t_{VE}-t_{VS}}{2}$ je polovični interval zasedenosti vozlišča (kar je enako varnostnem času predhodnika).

Izjema je preverjanje zasedenosti vozlišča, ki za trenutno vozilo predstavlja ciljno vozlišče. V tem primeru mora biti vozlišče prosto od trenutka prihoda nanj naprej.

Čakanje zaradi zasedenosti

Ko algoritem pri dodajanju vozlišč na odprti seznam naleti na vozlišče, ki je zasedeno, ali je zasedena cesta do njega, predlaga čakanje vozila pred zasedenim delom, da se ta sprosti. Ta čas čakanja algoritem prišteje k ceni-do-sem vozlišča, da se v nadaljevanju upošteva pri raziskovanju in izbiri poti.

Vozlišče, ki ga algoritem dodaja na odprti seznam je sestavljeno iz dveh delov; iz vozlišča samega in iz ceste, ki vodi do njega.

Kadar je zasedeno zgolj vozlišče, bo vozilo čakalo na trenutni cesti pred vozliščem. Kadar pa je zasedena trenutna cesta, vozilo ne čaka na predhodnem vozlišču, ampak na predhodni cesti. Vozlišča povezujejo več cest in predstavljajo križišča, čakanje na križišču pa bi zmanjšalo prehodnost zemljevida in zasedlo več poti hkrati. Vedno, ko vozilo čaka, je to na cesti pred vozliščem. Algoritem pri določanju cene čakanja vozila loči med čakanjem na trenutni cesti in čakanjem na predhodni cesti. Ceno-do-sem zato razdelimo na tri dele: cena-do-sem brez čakanja, cena čakanja na trenutni cesti in cena čakanja na predhodni cesti.

Kadar algoritem posodobi tudi ceno čakanja na predhodni cesti, s tem zakasni prihod vozila na konec predhodne ceste, kar podaljša čas nahajanja vozila na predhodni cesti. Potrebno je preveriti, če je v tem dodatnem času predhodna cesta še prosta in če je ob novem času prihoda na predhodno vozlišče le to še prosto. Če sta predhodna cesta in predhodno vozlišče prosta, se ponovi postopek od preverjanja trenutnega vozlišča in trenutne ceste naprej.

Če predhodna cesta ali vozlišče ob dodanem čakanju na predhodni cesti nista več prosta, trenutnega vozlišča ne moremo dodati na odprti seznam. V tem primeru algoritem predlaga umik na stransko cesto ali čakanje na cesti, ki je predhodna sedanji predhodni cesti. Kadar umik na stransko cesto ni možen, algoritem predlaga pomik čakanja po poti nazaj.

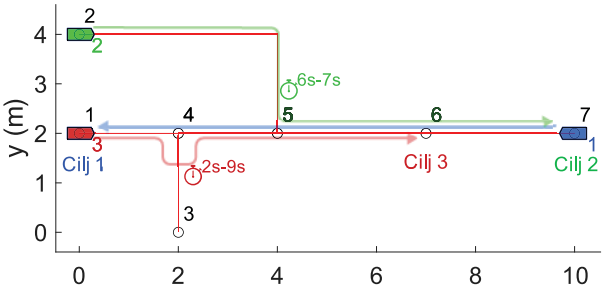
Rezultat algoritma

Ko algoritem razišče ciljno vozlišče je našel najhitrejšo pot. Eksplicitno ciljno vozlišče ne nosi informacije o celotni poti, ampak le o ceni te poti in predhodnem vozlišču. Kočno pot je potrebno sestaviti s sledenjem predhodnikov na zaprtem seznamu vozlišč.

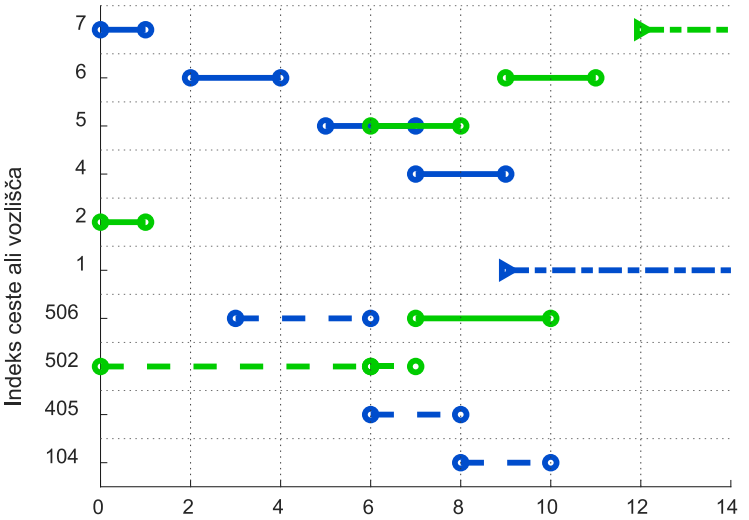
S sledenjem predhodnikov pridemo do začetnega vozlišča, če si za vsako najdeno vozlišče zapišemo še oznako pripadajoše ceste dobimo obrnjen seznam cest, ki vodi od začetka do cilja in sestavlja najdeno pot. Algoritem poleg seznama cest za vsako cesto vrne še podatek, če gre za umik na stransko ceste ter podatek o času čakanja na tej cesti.

Primeri delovanja

Algoritem bo predstavljen na primeru, ki je prikazan na sliki 7.32, kjer lahko vidimo začetne položaje treh robotskih vozil. Modro vozilo 1 z najvišjo prioriteto pot prične na vozlišču 7 in konča na vozlišču 1. Zeleno vozilo 2, ki je naslednje po prioriteti, prične pot na vozlišču 2 in konča na vozlišču 7. Zadnje, rdeče vozilo 3, katerega postopek iskanja poti bomo po korakih opisali, pot prične na vozlišču 1 in konča na vozlišču 6. Koordinate so predstavljene v metrih, hitrosti vseh vozil pa so 1m/s. Poti za vozila 1 in 2 sta že načrtani. Posledično so za te poti določena tudi časovna okna zasedenosti, ki za vsako cesto določajo čas prihoda, odhoda in čakanja vozila na njej. Prav tako za vsako vozlišče določajo trenutek



Slika 7.32: Zemljevid ter začetni položaji, cilji in končne poti treh robotskih vozil



Slika 7.33: Časovna okna zasedenosti cest in vozlišč za modro in zeleno vozilo na sliki 7.32

prihoda vozila nanj. Časovna okna zasedenosti za načrtane poti vozil 1 in 2 lahko vidimo na sliki 7.33. Vsaka črta predstavlja časovni interval oz. časovno okno, ko vozilo zasede cesto ali vozlišče. Levo od črte je zapisana oznaka vozlišča ali ceste, ki je zasedena. Vse ceste so usmerjene in vodijo od nekega vozlišča k drugemu. Njihove oznake so sestavljene iz oznak vozlišč, ki jih povezujejo. Npr. cesta 502 vodi od vozlišča 5 proti vozlišču 2. Pri cestah polna črta predstavlja zasedenost z vožnjo v smeri ceste, črtkana črta pa zasedenost z vožnjo v nasprotni smeri ceste. Levo krajišče črte predstavlja čas prihoda na začetek ceste, desno krajišče pa čas prihoda na konec ceste. Čakanje na cesti je določeno z dodatno označeno točko na črti, kjer čas od srednje točke do desnega krajišča predstavlja čas čakanja. Pri vozliščih čas prihoda na vozlišče predstavlja sredina črte, ki je razširjena z namenom, da vozila na vozlišče prihajajo z razmakom varnostnega časa. Izjema je poltrak oblike črta-pika, ki označuje čas od prihoda vozila na ciljno vozlišče naprej. Ta časovna okna algoritem upošteva pri iskanju poti rdečega vozila 3.

Pri iskanju poti rdečega vozila algoritem upošteva okna zasedenosti in išče možno pot v grafu. Začne z vozliščem 1, nadaljuje do naslednjega možnega vozlišča 4 in

ker je v času vožnje po cesti 104 (0-2 s) in ob času prispetja v vozlišče 4 (2 s) prosto, ga doda na odprti seznam.

V drugem koraku algoritem preišče naslednike vozlišča 4, torej vozlišči 3 in 5. Vozlišče imata enako ceno-do-sem (4 s) a vozlišče 5 ima manjšo ceno-do-cilja (3 s, vozlišče 3 pa ima ceno-do cilja 7 s) in posledično tudi manjšo ceno celotne poti, zato algoritem razišče njegove naslednike in ga doda na zaprti seznam.

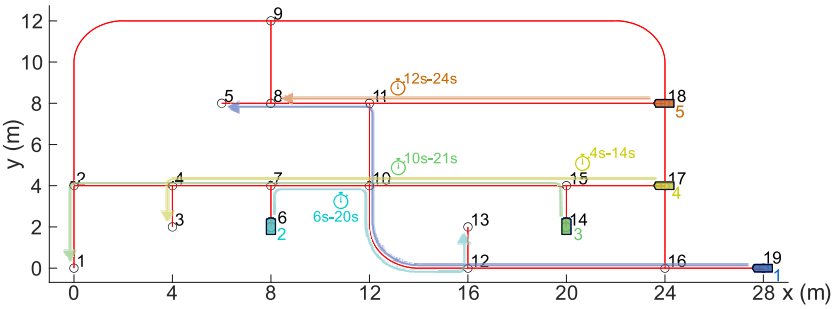
Nato v tretjem koraku algoritem izbere vozlišče 6, do katerega vodi cesta 506. Cena-do-sem za vozlišče 6 znaša 7 s in predviden interval vožnje po cesti 506 je 3-6 s. Cesta je znotraj tega intervala zasedena (glejte sliko 7.33) z vožnjo modrega vozila v nasprotni smeri do časa 6 s. To predstavlja konflikt, kateremu se želimo izogniti.

S tem se začne rekurziven postopek v katerem algoritem s poizkušanjem išče primerno mesto za čakanje na sprostitev poti in izogib konfliktu. Najprej poizkusi s čakanjem na cesti, ki se na poti do zasedene ceste nahaja pred njo, torej je njena predhodna cesta (cesta 405). Ta je v časovnem intervalu (2-6 s) (prihod na začetek ceste in zakasnjen prihod na konec ceste) prosta, a zasedeno je vozlišče 5 do časa 8s, kar še podaljša čakanje na cesti 405 za 2s, v tem dodatnem času pa cesta ni več prosta, torej podaljšanje čakanja ni mogoče.

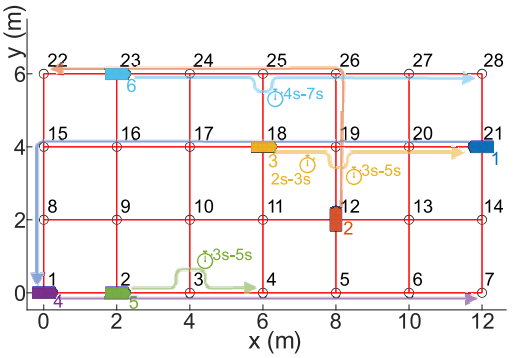
Nadalje algoritem preveri, če bi lahko vozilo čakalo na sprostitev ceste 506 na stranski cesti 502 predhodnega vozlišča 5. Tudi cesta 502 v intervalu 4-6 s (vključujoč potreben čas čakanja 2s) ni prosta. Algoritem v nadaljevanju preizkuša ostala možna mesta čakanja po poti, ki vodi do trenutnega vozlišča, nazaj, z namenom, da najde mesto, kjer lahko brez oviranja drugih vozil čaka na sprostitev poti do željnega vozlišča. Po nekaj iteracijah algoritem najde možno mesto čakanja na stranski cesti 403 za časovni interval 2-9 s kot je prikazano na sliki 7.32. Po dodanem čakanje na stranski cesti, algoritem v naslednjem koraku razišče vozlišče 5, ki nasledi umik na stransko cesto. Vozlišče 5 je ponovno raziskano, saj je vmes dodano čakanje in zakasnitev prihoda. V naslednjem koraku razišče še vozlišče 6 in s tem pride do cilja. S tem je algoritem določil optimalne poti za vsa tri vozila brez konfliktov in upoštevajoč prioritetno listo.

Na slikah 7.34 in 7.35 sta prikazana rezultata planiranja poti še na dveh drugih zemljevidih. Prikazani so začetni položaji vozil z njihovimi oznakami ter izris načrtanih poti. Vsa vozila vozijo s hitrostjo 1 m/s. Vozilo z oznako 1 ima najvišjo prioriteto in vozilo z najvišjo oznako ima najnižjo prioriteto. Čakanje je ponazorjeno z znakom ure s pripisom časa začetka in časa konca čakanja. Zemljevid na sliki 7.34 predstavlja skladišče z mesti dolaganja tovora na vozila (slepe ceste) ter z bolj obremenjenim osrednjim delom iz vozlišč 10 in 11. Zemljevid na sliki 7.35 predstavlja preprost a splošno uporaben mrežast zemljevid, ki ga lahko, s poljubno odstranitvijo posameznih cest ali vozlišč apliciramo na mnoga skladišča ali tovarne.

Prikazan pristop planiranja več robotskih vozil nadgrajuje algoritem A*, ki je v osnovi namenjen planiranju poti enega vozila. Algoritem upošteva prioritete



Slika 7.34: Prikaz rezultata planiranja na primeru s 5 vozili. Izrisani so začetni položaji vozil in načrtane poti. Simbol ure s pripisom časa začetka in časa konca čakanja označuje mesto čakanja vozila.



Slika 7.35: Prikaz rezultata planiranja na primeru mrežastega zemljevida cest. Izrisani so začetni položaji vozil in načrtane poti. Simbol ure s pripisom časa začetka in časa konca čakanja označuje mesto čakanja vozila.

transportnih nalogov, kar omogoča bolj učinkovito izvajanje prioritetnih zadolžitev, vozila z nižjo prioriteto pa se prilagodijo prvim. Strategija določanja časovnih oken zasedenosti omogoča zaznavo potencialnih konfliktov, ki se jim algoritem izogne z izbiro druge poti, s čakanjem pred konfliktom ali z umikom na stransko pot. Kompleksnost algoritma narašča s številom vozil, saj se vozila z nižjo prioriteto večkrat znajdejo v zasedenih delih zemljevida in morajo iskati alternativne poti brez konfliktov.

Literatura

[1] R. C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, zv. 8, št. 4, str. 92–112, 1989.

[2] R. A. Brooks. Intelligence Without Representation. *Artificial Intelligence*, zv. 47, str. 139–159, 1991.

[3] G. Klančar, M. Kristan in sod. Robust and efficient vision system for group of cooperating mobile robots with application to soccer robots. *ISA Transactions*, zv. 43, str. 329–342, 2004.

[4] G. Klančar in D. Matko. Strategy control of mobile agents in soccer game. V *The 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems CIRAS 2005 & FIRA RoboWorld Congress Singapore 2005 FIRA 2005*, str. 1–6. National University of Singapore: CIRAS & FIRA Organising Committee, 2005.

[5] G. Klančar, D. Matko in S. Blažič. A control strategy for platoons of differential-drive wheeled mobile robot. *Robotics and Autonomous Systems*, zv. 59, št. 2, str. 57–64, 2011.

[6] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, zv. 22, št. 11, str. 1330–1334, 2000.

[7] J. Perš in S. Kovačič. Nonparametric, model-based radial lens distortion correction using tilted camera assumption. V *Kropatsch (Eds.), Proceedings of the Computer Vision Winter Workshop 2002*, str. 286–295. 2002.

[8] S. Garrido-Jurado, R. Muñoz-Salinas in sod. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, zv. 51, str. 481–491, 2016.

[9] F. J. Romero-Ramirez, R. Muñoz-Salinas in R. Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, zv. 76, str. 38–47, 2018.

[10] W. Boehm in A. Müller. On de casteljau's algorithm. *Computer Aided Geometric Design*, zv. 16, št. 7, str. 587–605, 1999.

[11] L. Teslić, I. Škrjanc in G. Klančar. Using a LRF sensor in the Kalman-filtering-based localization of a mobile robot. *ISA transactions*, zv. 49, št. 1, str. 145–153, 2010.

[12] R. Olmi, C. Secchi in C. Fantuzzi. Coordination of multiple agvs in an industrial application. V *2008 IEEE International Conference on Robotics and Automation*, str. 1916–1921. IEEE, 2008.

[13] E. Gawrilow, E. Köhler in sod. Dynamic routing of automated guided vehicles in real-time. V *Mathematics–Key Technology for the Future*, str. 165–177. Springer, 2008.

[14] V. Digani, L. Sabattini in sod. Ensemble coordination approach in multi-agv systems applied to industrial warehouses. *IEEE Transactions on Automation Science and Engineering*, zv. 12, št. 3, str. 922–934, 2015.

[15] Z. Zhang, Q. Guo in sod. Collision-free route planning for multiple agvs in an automated warehouse based on collision classification. *IEEE Access*, zv. 6, str. 26022–26035, 2018.

[16] N. Presečnik in G. Klančar. Nadgradnja algoritma A* za planiranje poti več robotskih vozil po prioritetnem sistemu. V *Zbornik dvanajste konference Avtomatizacija v industriji in gospodarstvu (AIG'21)*, str. 1–9. Društvo avtomatikov Slovenije, 2021.


Dodatni material
http://msc.fe.uni-lj.si/ams-kv