UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

Žiga Petrič

Vodenje invertiranega sferičnega nihala z robotom tipa SCARA

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Ljubljana, 2014

UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

Žiga Petrič

Vodenje invertiranega sferičnega nihala z robotom tipa SCARA

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Mentor: prof. dr. Igor Škrjanc

Ljubljana, 2014

Zahvala

Zahvaljujem se mentorju prof. dr. Igorju Škrjancu za podporo pri izbiri teme in pomoč pri izdelavi diplomske naloge.

Kazalo

Povzetek

V izhodišču smo želeli diplomsko delo povezati z delom v podjetju, ki se ukvarja z zastopanjem opreme za avtomatizacijo znamke *Allen-Bradley* (sedaj del skupine *Rockwell Automation*). Ta oprema vključuje: programabilne krmilnike (*PLC*), vhodno-izhodne (I/O) enote, frekvenčne in servo pogonske naprave, vmesnike človek-stroj (*Human-Machine Interface*), industrijska vodila (*Fieldbus*), senzorje, programsko opremo in druge industrijske komponente.

Cilj naloge je preučiti, uporabiti in ovrednotiti nekatere napredne funkcije, ki jih ponuja oprema Allen-Bradley, vendar so pri realizaciji praktičnih projektov večinoma neizkoriščene. Te funkcije so mehatronika oz. robotika, napredno vodenje in regulacije, možnosti simulacije in modeliranja sistemov. Nastala naj bi demonstracijska in učna platforma, uporabna za izobraževalne namene in za prikaz zmožnosti krmilnega sistema potencialnim uporabnikom.

Za nalogo smo izbrali robot tipa SCARA, ki ima na prijemalo pritrjeno invertirano sferično nihalo. Nihalo je prosto gibljivo v dveh prostostnih stopnjah. Robot zaniha nihalo v navpično lego in jo vzdržuje, medtem pa se lahko giblje v delovnem območju in izvaja druge naloge. Delo vključuje izdelavo matematičnega modela nihala, algoritem za zanihanje nihala v navpično lego, regulator za vodenje nihala, simulacijo in 3D vizualizacijo celotnega sistema. Narejen je tudi grafični vmesnik za upravljanje s poskusom in različne nastavitve parametrov.

Ključne besede: invertirano nihalo, sferično nihalo, robot SCARA, regulator stanj, simulacija, Allen-Bradley

Abstract

This work links to author's professional occupation of representing and supporting *Allen-Bradley* industrial automation products (now part of *Rockwell Automation* company). Allen-Bradley products include programmable logic controllers (PLCs), input-output (I/O) devices, variable frequency and servo drives, human-machine interfaces (HMI), industrial fieldbus solutions, sensors, software and other industrial components.

We used and evaluated advanced functions of Allen-Bradley control equipment (which are seldom used in most automation tasks), including robot kinematics, advanced control algorithms and simulation options. Resulting platform should prove useful for training and commercial demonstration purposes.

Selected control task involves SCARA type robot with spherical (2 DOF) inverted pendulum, pivoting freely on its effector point. Robot will swing up the pendulum and maintain upright position, while being able to move inside its working envelope and execute secondary tasks. Work includes development of spherical pendulum governing equations, swing-up and control procedures, simulation and 3D visualization of complete system. Graphical interface enables user to command the experiment and set different variables.

Key words: inverted pendulum, spherical pendulum, SCARA robot, state space controller, simulation, Allen-Bradley

1 Uvod

1.1 Cilj naloge

Cilj naloge je preučiti, uporabiti in ovrednotiti nekatere napredne funkcije, ki jih ponuja oprema Allen-Bradley, vendar so pri realizaciji praktičnih projektov večinoma neizkoriščene. Te funkcije so mehatronika oz. robotika, napredno vodenje, možnosti simulacije in modeliranja sistemov. Nastala naj bi demonstracijska in učna platforma, uporabna za izobraževalne namene in za prikaz zmožnosti krmilnega sistema potencialnim uporabnikom.

Fizična naprava je za takšen namen nepraktična, ker ni prenosljiva, poleg tega pa potrebujemo delovno mesto za več udeležencev šolanja ali predstavitve hkrati. Tipičen primer opreme, ki jo uporabljamo za šolanje, je prikazan na sliki 1.1. Gre za dvoosni servopogonski sistem male moči: za mrežastimi vratci sta dva servomotorja, na osi je montiran disk z oznako položaja, vratca so opremljena z varnostnim zaklepom, komplet pa vsebuje še nekaj tipk, stikal in signalnih svetilk za simulacijo različnih I/O stanj. Napajanje naprave je omrežna napetost 230 VAC. Celotna naprava se lahko zapre kot kovček in ima vgrajen ročaj in kolesa za transport.

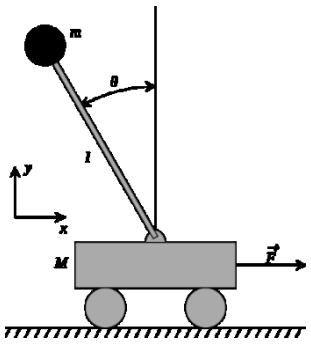
Slika 1.1: Demonstracijska oprema za izobraževanje

Demonstracijska platforma naj bi torej dopolnjevala tak sistem, bila naj bi modelirana in simulirana, vendar do te mere, da bi bili rezultati hitro in z minimalnimi posegi prenosljivi na realno, fizično napravo. Oprema, ki smo uporabili, je naslednja:

- PLC krmilniki družine *Logix;*
- servoojačevalniki in servomotorji družine *Kinetix* (kot na sliki 1.1);
- programska oprema *RSTestStand.*

Slika 1.2: Modularni PLC družine Logix, CompactLogix (od leve proti desni: CPU enota, modul 8xAI, napajalnik, modul 16xDI, modul 16xDO)

Kot delovno platformo smo izbrali invertirano nihalo na vozičku, ki je klasičen referenčni sistem za preizkušanje različnih sistemov vodenja.



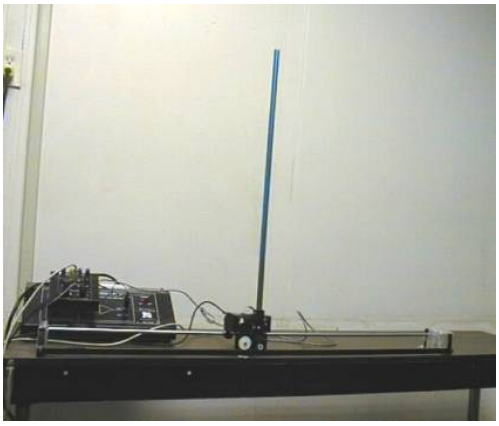
Slika 1.3: Shema invertiranega nihala na vozičku

1.2 Invertirano nihalo kot eksperimentalna platforma

Praktično je tak sistem navadno izdelan kot laboratorijska naprava za izobraževalne namene, ki jo sestavlja:

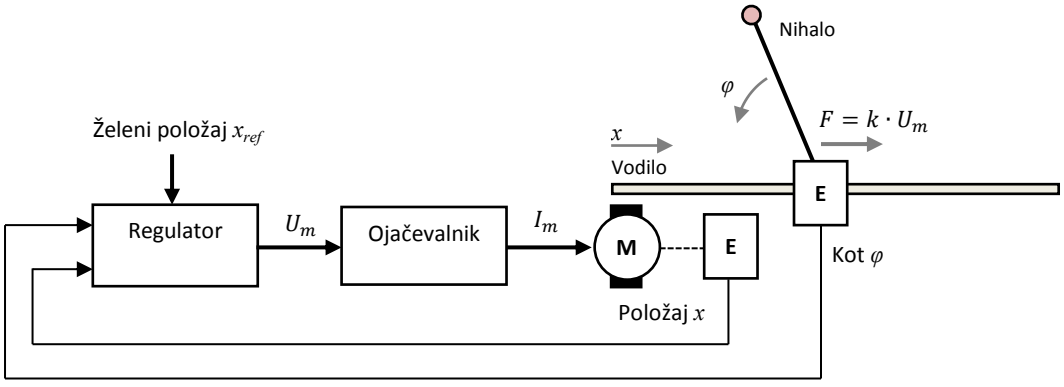
- linearno vodilo (dolžine 1 do 2 m);
- voziček s pritrjenim nihalom (0,2 do 1 m);

- DC elektromotor, ki žene voziček direktno ali preko jermena;
- napetostno krmiljen ojačevalnik (*driver*) za motor;
- dva optična linijska dajalnika (*enkoderja*) za merjenje kota odmika nihala in pomika vozička, z vezjem za pretvorbo merjene veličine v analogno (napetost) ali digitalno vrednost;
- regulator, ki je lahko analogno vezje z operacijskimi ojačevalniki, programabilni (mikro)krmilnik ali PC programska oprema (npr. *Matlab* paket) z ustreznimi I/O vmesniki.



Slika 1.4: Praktična izvedba eksperimenta z invertiranim nihalom

Povezavo teh komponent prikazuje slika 1.5.



Slika 1.5: Bločna shema eksperimentalne naprave z invertiranim nihalom na vozičku

Vhodne veličine regulatorja so želeni in dejanski položaj vozička ter dejanski položaj nihala. Želeni položaj oz. kot nihala je 0 (navpična lega); če je ta različen od 0, se mora sistem pospešeno gibati v pozitivno ali negativno smer, to gibanje pa ima mehanske omejitve. Izhodna veličina regulatorja je napetost U_m, ojačana v krmilni tok motorja I_m. Ta generira

moment M in preko jermenskega ali zobniškega prenosa silo F na voziček. Navadno se smatra, da so U_m, I_m, M in F linearno odvisne veličine:

$$F = k_1 \cdot M = k_2 \cdot I_m = k_3 \cdot U_m. \quad (1.1)$$

Regulator, ki vzdržuje navpično lego nihala, je lahko različnih izvedb; v literaturi lahko najdemo klasično rešitev z regulatorjem stanj (*State Space Controller*), PID regulator, uporabo mehke logike (*Fuzzy Logic*), nevronske mreže (*Neuron Network Controller*) in druge različne sisteme.

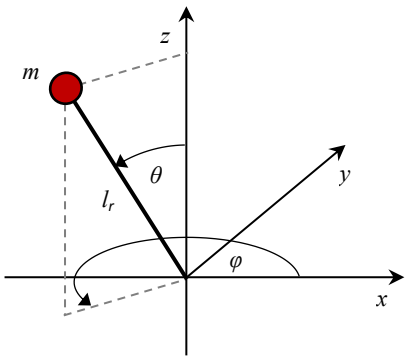
Alternativna oblika te naprave je »vozilo«, ki je v mirovanju nestabilno, vendar lahko lovi ravnotežje z vožnjo naprej in nazaj. Praktična izvedba je npr. *Segway*, pri katerem je "nihalo" voznik in krmilo, "voziček" pa sta dve kolesi, ki sta gnani neodvisno. Naprave takšnega tipa za izobraževalne namene lahko najdemo tudi na osnovi *Fischertechnic* in *LEGO Mindstorms* sestavljank.

Slika 1.6: *Segway*, *Lego* in *Fischertechnic* – invertirano nihalo kot vozilo

1.3 Dodatne zahteve

Problem invertiranega nihala je torej že mnogokrat teoretično in praktično obdelan, zato smo ga razširili z dodatnimi zahtevami:

1. Nihalo je sferično nihalo, naj ima dve prostostni stopnji, gre torej za »balansiranje metle na roki.



Slika 1.7: Sferično nihalo s prostostnima stopnjama θ (elevacija) in φ (azimut), z utežjo mase m in dolžino l_r

Slika 1.8: Otroci lovijo metlo v ravnotežju na roki

Večja kot je razdalja do težišča balansiranega objekta, lažja je naloga, ker je lastna frekvenca nihala manjša.

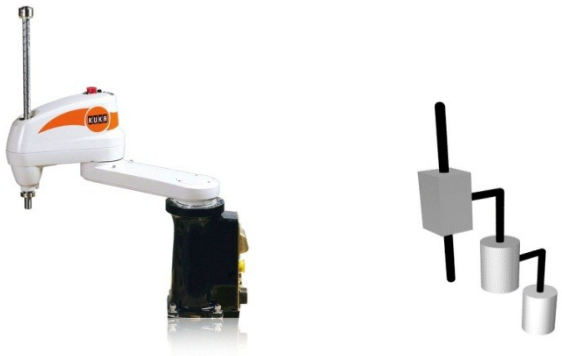
2. Za vodenje sferičnega nihala je potreben sistem, ki lahko premika pivot nihala v smereh x in y. Analitično je najpreprostejša uporaba kartezičnega sistema, katerega prostostni stopnji gibanja sta x in y (lahko tudi z), primer praktične izvedbe je prikazan na sliki 1.9.

Slika 1.9: Kartezični (*Gantry*) robot, sestavljen iz linearnih vodil s servopogoni

Zdi se, da je regulator za takšen sistem preprosta razširitev že prikazane naprave (slika 1.5) na dve neodvisni koordinati gibanja.

Kot zahtevnejšo rešitev smo izbrali industrijski robot tipa SCARA, ki ravno tako omogoča gibanje v ravnini *x–y*, pri čemer pogoni za posamezne osi gibanja niso neodvisni.

Kratica SCARA pomeni *Selective Compliance Articulated Robot Arm* in je pogosta oblika robota za preprostejša opravila (npr. *"Pick-and-place"*).



Slika 1.10: Industrijski SCARA-robot s 4 prostostnimi stopnjami (*Kuka*) in kinematični diagram

Sedaj naprava ni več preprosta razširitev regulatorja invertiranega nihala na dve prostostni stopnji. Za to je več razlogov:

- soodvisnost osi A1 in A2: narekuje uporabo inverzne kinematike, torej pretvorbe ukazov gibanja iz kartezičnega v SCARA koordinatni sistem;
- mehanske omejitve: sferično nihalo se ne more gibati v polnem prostorskem kotu, ne da bi zadelo v druge dele naprave;
- nabor ukazov za programiranje robota: ta je navadno namenjen pozicioniranju, gibanju po vnaprej izračunani poti in ne prostemu pospeševanju, kot to počne voziček invertiranega nihala. Seveda lahko pričakujemo, da bo osnovni regulator še vedno uporaben, bo pa potrebno razviti vmesni blok med izhodom regulatorja U_m in robotskim aktuatorjem. Ta blok je program, ki na nek način prevede »analogni« izhod regulatorja v gibanje robota v ravnini *x–y*.

3. Sistem mora imeti možnost, mehansko in v smislu krmiljenja, da zaniha nihalo iz spodnje (stabilne) lege v zgornjo (nestabilno), kjer vodenje prevzame regulator. Za ta namen se zdi SCARA robot primerna konstrukcija, ker minimalno ovira prosto gibanje nihala.

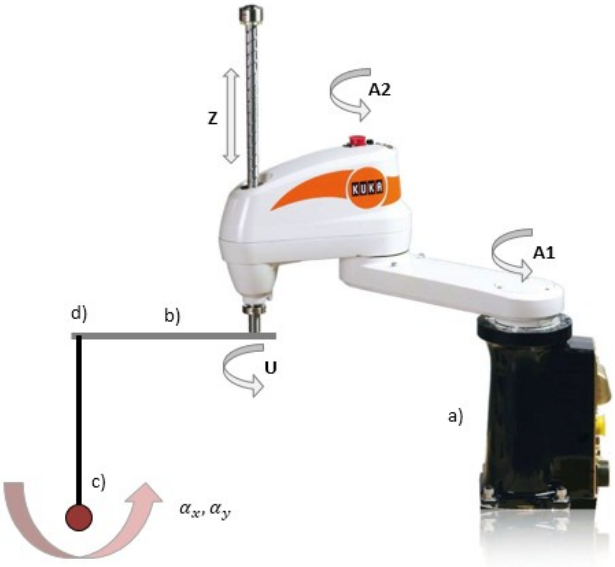
4. Realiziranih naj bo več tipov regulatorjev, med katerimi lahko preklapljamo in primerjamo delovanje.
5. Na voljo imamo krmilnik, servoojačevalnike in motorje, ne pa tudi mehanske konstrukcije robota in nihala. Ta naj bo torej modelirana in simulirana, in sicer v celoti na krmilniku družine Logix.
6. Simuliran sistem bo grafično prikazan v realnem času in v 3D izrisu v programu RSTestStand, kjer naj bo tudi operaterski vmesnik za upravljanje s poskusno napravo.

Realizirana naprava bo odprla možnosti za množico nadaljnjih poskusov (nekateri bodo našteti v zaključku) in bo v ta namen tudi na voljo za bodoče razvijalce.

2 Izvedba naprave

2.1 Mehanska izvedba

Idejno izvedbo naprave prikazuje slika 2.1.



Slika 2.1: Idejna shema montaže nihala na robot

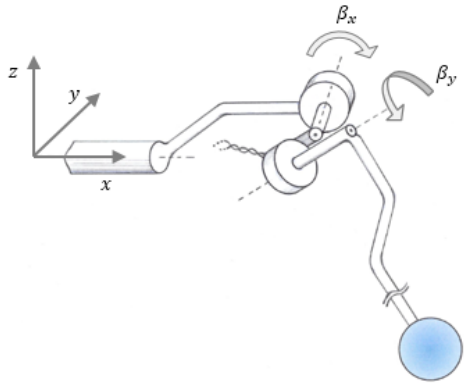
Poskusna naprava je torej robot tipa SCARA (a) s pripadajočim krmilnikom. Na mesto za vpetje orodja je pritrjen tog nosilec (b), ter nanj nihalo (c) preko krogličnega zgloba (*universal joint*) (d). Nosilec omogoča, da se nihalo lahko prosto giblje v vsaj eni ravnini za polni kot 360° in pri tem ne zadane v druge dele robota. Prostostne stopnje robota so označene kot **A1**, **A2**, **Z** in **U**, žene jih ustrezen električni motor. A1 in A2 določata položaj roke v *x*–*y* ravnini, os Z ustreza kartezični koordinati *z*, os U pa opravlja vrtenje orodja okoli osi *z*. Vrtenje osi **U** sledi vrtenju **A1** in **A2**, tako da je nosilec vedno obrnjen v isto smer (pivot nihala se torej ne vrti okrog osi **Z**). V pravilnem položaju roke robota je možno zanihati nihalo v navpično lego. Na nosilec sta pritrjena tudi dva dajalnika položaja (optični linijski enkoder), ki dajeta sistemu informacijo o legi nihala (α_x in α_y, torej kota, ki ju dobimo s projekcijo nihala na ravnino *x*–*z* in *y*–*z*).

Ugotovimo lahko, da uporaba robota ni trivialna izpeljanka naprave z vozičkom. Medtem ko za sistem z vozičkom velja, da je sila F v smeri x približno proporcionalna I_m (tok motorja oz. izhodnega ojačevalnika), to ne drži posplošeno za ravnino x-y, ki jo pokriva robot z prostostnima stopnjama A1 in A2:

$$F_x \neq k_x \cdot I_{A1}\ ;\ F_y \neq k_y \cdot I_{A2}. \tag{2.1}$$

Direktno krmiljenje SCARA robota z izhodom regulatorja I_m ni možno, potreben je vmesni člen, ki ga bomo opisali v nadaljevanju.

Zajemanje položaja nihala je predvideno z dvema dajalnikoma položaja, kot prikazuje slika 2.2.



Slika 2.2: Vpetje nihala na nosilec in zajemanje položaja kotov z dvema enkoderjema

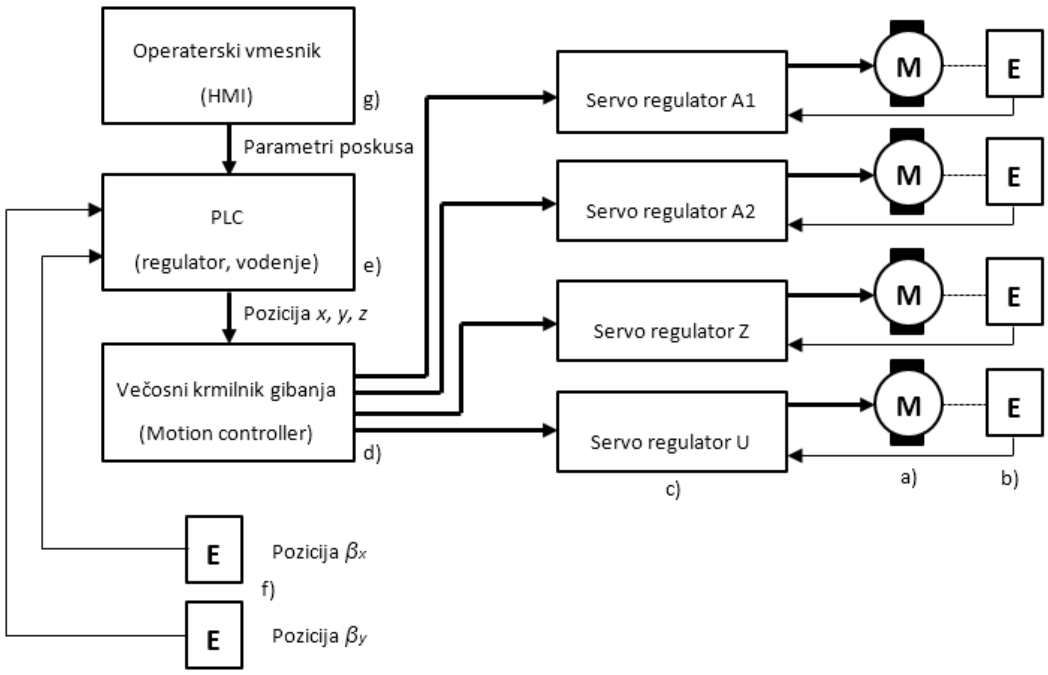
Kota sta sedaj označena z β_x in β_y, ki sta Eulerjeve rotacije – najprej okoli osi x za kot β_y in nato okoli osi y za kot β_x. Indeksa x in y se nanašata na ravnino rotacije (x-z in y-z). Pri majhnih odmikih od ravnovesne oz. navpične lege (in ko je os robota U = 0°), velja, da je $\beta_x \approx \alpha_x$ in $\beta_y \approx \alpha_y$.

Za zaznavanje položaja kotov je lahko uporabljen komercialno dostopen enkoder primerno majhnih dimenzij, kot npr. model na sliki 2.3.

Slika 2.3: Enkoder majhnih dimenzij, “mikroenkoder”

2.2 Izvedba krmilnih in močnostnih komponent

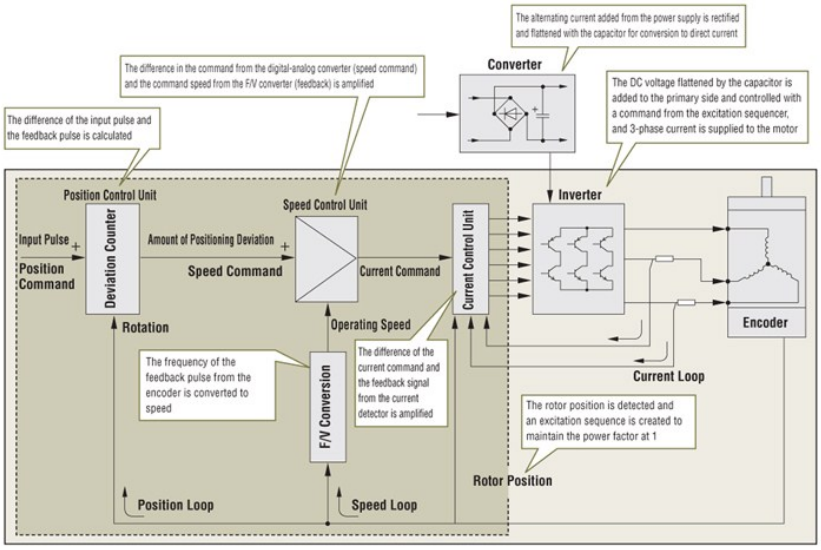
Slika 2.4 prikazuje splošno bločno shemo naprave.



Slika 2.4: Bločna shema naprave

V bločni shemi vidimo logične povezave med elektronskimi gradniki naprave. Motorji robota (a) so v splošnem brezkrtačni sinhronski motorji s permanentnimi magneti, napajani s trifaznim izmeničnim (AC) tokom. Dajalnik (b) je navadno integriran na os motorja, enoto bomo imenovali servomotor (slika 2.5). Servomotor poganja močnostna enota (c) z zaprtozančno regulacijo toka, hitrosti in položaja – servoregulator (bločna shema na sliki 2.6). Želeno vrednost toka, hitrosti in položaja daje krmilnik gibanja (*Motion controller*) (d), katerega naloga je koordinacija gibanja osi, da te sledijo želeni poti gibanja. Pot gibanja določa regulator, ki je realiziran na platformi programabilnega krmilnika (PLC) (e). Vhodno veličino v regulator predstavlja odklon nihala (β_x, β_y), ki ga merimo z dajalniki (f), potreben pa je ustrezen vmesnik za PLC, navadno hitroštevni modul (100kHz do 1Mhz) z več kanali. PLC vsebuje tudi program za zanihanje nihala in druge pomožne funkcije. Poskus poženemo in nadzorujemo preko HMI (*Human Machine Interface*) vmesnika (g), ki je lahko kar osebni računalnik.

Slika 2.5: Brezkrtačni industrijski AC servomotor (*Allen-Bradley* tip MPL) s konektorji za napajanje in dajalnik položaja (enkoder), nazivni navor 1.25 Nm, max. hitrost 4000 obr/min

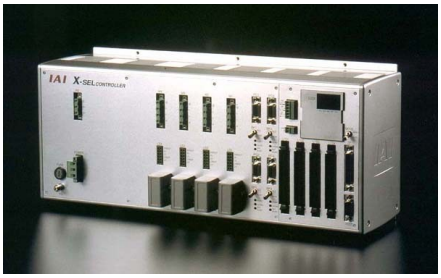


Slika 2.6: Bločna shema zaprtozančnega servopogona in motorja

Močnostni del servoregulatorja sestavlja:

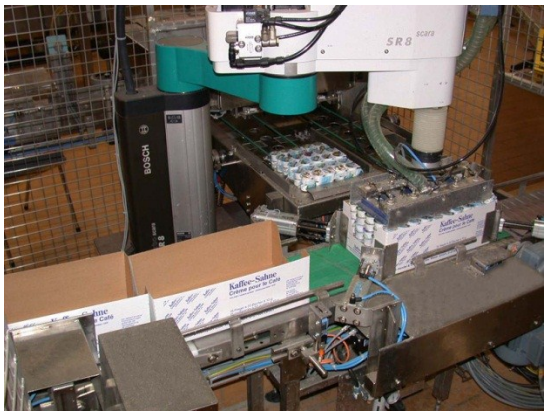
- usmernik (*converter*), ki vhodno napetost (230 VAC/1ph ali 400 VAC/3ph) usmeri in zgladi s kondenzatorji;
- razsmernik (*inverter*), ki enosmerno (DC) napetost pretvori v impulzno-širinsko moduliran (PWM) izhodni signal. Izhodni tok PWM signala je po srednji vrednosti podoben trifaznemu izmeničnemu toku določene frekvence (od 0 do 300 Hz) in amplitude (vsebuje seveda še višje harmonske komponente, te se delno odstranijo s pasivnimi LC filtri in dušilkami). Odprtozančna izvedba pogona je navadno namenjena standardnim AC motorjem z indukcijsko kletko, zaprtozančna pa servomotorjem.

Industrijski roboti manjših moči (300 do 1000 W) imajo navadno enote c), d) in e) s slike 2.4 integrirane v eno napravo. Tak krmilnik je namenski, s specifično programsko opremo (*Firmware*) za izbran tip in model robota. Možne so razširitve za I/O (vhodno/izhodne) kartice in komunikacijske povezave (Ethernet, Profibus, RS-232 ipd.).



Slika 2.7: Namenski 4-osni robotski krmilnik male moči, *IAI X-Sel*

SCARA roboti manjših moči so namenjeni predvsem t.i. »*Pick-and-place*« operacijam, torej opravilom, pri katerih robot prime nek objekt na izhodiščni točki, ga po programirani poti prenese do ciljne točke in tam odloži.



Slika 2.8: Tipična *Pick-and-place* operacija s SCARA robotom *Bosch*, aplikacija je zlaganje izdelkov v škatle

Pick-and-place je torej manj zahtevna robotska naloga, zato je v splošnem možno, da krmilnik takega robota (kot npr. zgoraj prikazani X-Sel krmilnik, slika 2.7) ni primeren za zadano nalogo.

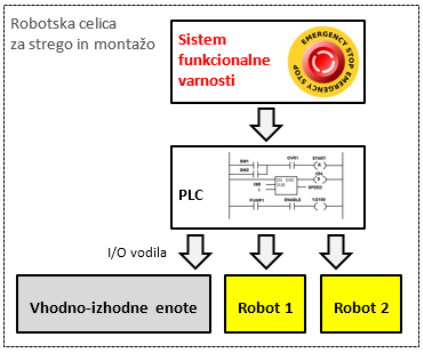
Podpirati bi namreč moral tudi naslednje funkcije:

– priklop dveh dajalnikov položaja, programski dostop do položaja v realnem času

(priklop za en dajalnik je običajno možen, vendar nabor funkcij lahko podpira samo sinhronizacijo hitrosti prijemala s hitrostjo dajalnika – npr. za pobiranje objektov s tekočega traku, t.i. »*on the fly*«);

- programske bloke ali funkcije PID, integracije, odvajanja, izračuna SIN, COS ipd. (kar ni običajen nabor ukazov za male robote);
- izvajanje programa regulatorja v ponavljajoči zanki z deterministično periodo (robotski program ima običajno strukturo zaporedja korakov, katerih začetek in konec je pogojen z zunanjimi digitalnimi signali (fotocelica, stikalo ipd.));
- dinamično spremembo poti gibanja prijemala glede na ukaze regulatorja (krivulja gibanja je navadno določena vnaprej (t.i. »*point-to-point move*«), možno je določen gib v hipu zaustaviti (zasilna ustavitev), ne pa tudi pretvoriti poljubno v novo gibanje).

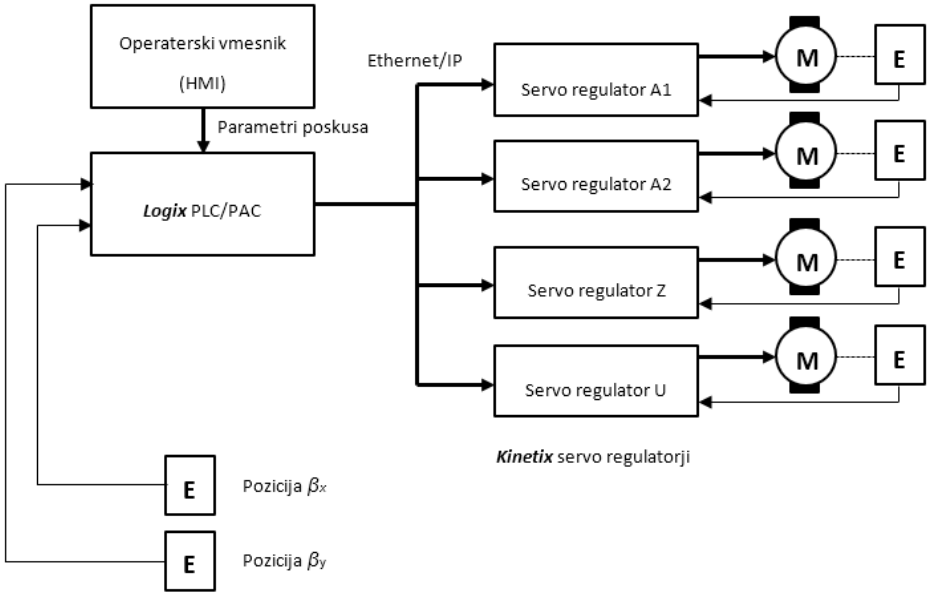
Druga možnost je uporaba ločenega PLC krmilnika, kar je običajen primer v industrijski praksi. Redko namreč krmilnik robota nadzoruje delovanje celotne naprave (celice), katere del je robot. Čeprav ima krmilnik robota vgrajene nekatere splošne PLC funkcije (npr. časovnike, števce, bitne, primerjalne, matematične in ASCII ukaze, if...then...else strukture), je število I/O točk omejeno, cena posamezne točke je višja, težja je integracija HMI in drugih naprav...
Bolj pogosto je robot (ali več robotov) podrejena naprava PLC krmilniku, ki preko vodila ali namenskih I/O signalov starta vsak delovni cikel, izvrši inicializacijo in zaustavitev stroja, določa izbiro podprogramov (vrsta izdelka) in krmili pomožne sklope stroja (tekoči trak, izmetala, orodja, regulacija temperature in tlaka ipd.).



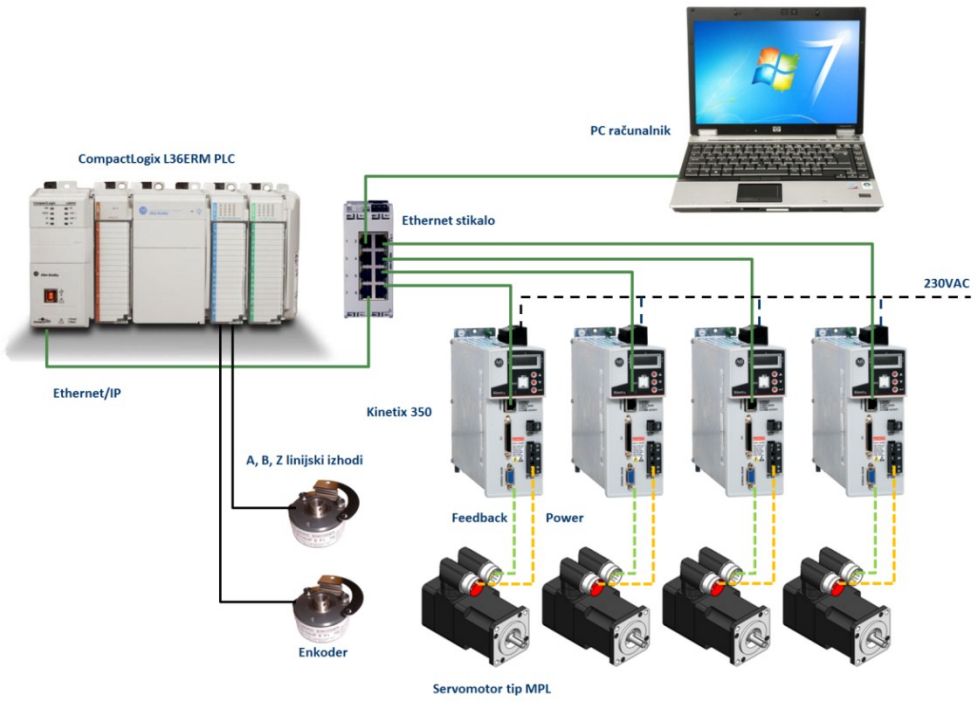
Slika 2.9: Relacija oz. hierarhija med različnimi krmilnimi sistemi v tipični robotski celici

V primeru, da zgoraj naštete funkcije izvaja ločen PLC, pa sta še vedno vprašljiva način in hitrost povezave med obema sistemoma (analogni signali, digitalno vodilo).

V nalogi bo uporabljen drugačen pristop. Večosni krmilnik gibanja in PLC bo združen v eno napravo – krmilnik družine Logix. Kot servoregulatorji bodo uporabljene naprave družine Kinetix. Bločno shemo prikazujeta sliki 2.10 in 2.11.



Slika 2.10: Združitev kinematičnih in krmilnih funkcij v eno napravo



Slika 2.11: Izvedba z izbrano opremo

Sodobni PLC krmilniki, kamor sodi tudi Logix, imajo tudi naziv PAC – *Programable Automation Controller*. Lastnosti PAC-sistemov so v splošnem naslednje:

- univerzalnost PC računalnikov in zanesljivost PLC krmilnikov;
- zamenljiv operacijski sistem;
- x86 ali RISC arhitektura (32 ali 64-bitna);
- večopravilnost (*Multitasking*);
- velik pomnilnik (>MB);
- industrijski Ethernet;
- več programskih jezikov (po standardu IEC 61131-3);
- objektno programiranje;
- inteligentne, "*Plug & Play*" enote;
- preprosta integracija podrejenih naprav.

V primerjavi s PLC ponuja PAC večopravilen sistem, ki lahko vključuje tudi funkcije:

- pozicioniranje – robotika – mehatronika;
- funkcionalna varnost (Functional Safety);
- računalniški vid, obdelava slik;
- direktne povezave s podatkovnimi bazami (SQL ipd.);
- napredne komunikacijske funkcije;
- možnost postavitve redundantnih oz. večprocesorskih sistemov;
- precizna sinhronizacija časa (< ms), po standardu IEEE1588 ali preko sistema GPS.

Z Logix PAC lahko torej nadomestimo namenski krmilnik robota in uporabimo »*Open platform*« tip robota. To je robotski mehanizem, na katerega lahko uporabnik sam montira primerne motorje in krmiljenje in eliminira namenski večosni krmilnik gibanja. Primer izdelovalca takih robotov sta podjetji *Codian Robotics* (Nizozemska) in *Afast* (ZDA) – obe ponujata platformo z vgrajenim servomotorjem ali brez. Podoben, čeprav ne direkten pristop ponuja družina robotov *Adept Cobra ePLC*.

Slika 2.12: Open platform Delta robot, *Codian Robotics*

Slika 2.13: Open platform SCARA robot, *Afast Robotics*

Slika 2.14: SCARA robot, *Adept Cobra ePLC800*

Takšen pristop ima več zanimivih prednosti:

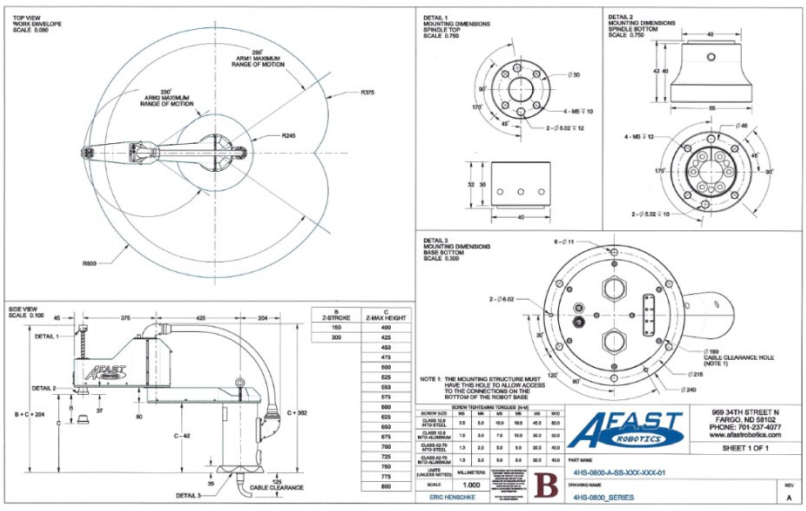
– odpade namenski, zgolj vodenju robota namenjen krmilnik;

- odpade posebna integracija (I/O signali ali vodilo) med sistemom PLC in robotom;
- en PLC/PAC lahko krmili več robotov;
- Proces učenja (*Learning curve*) uporabnika je hitrejši, ker ima ta opravka samo z enim programskim orodjem;
- robotski program lahko uporablja funkcije, ki jih primerljiv namenski krmilnik nima;
- robot lahko uporablja I/O naprave (senzorje, kamere za strojni vid, frekvenčne pogone, varnostne naprave), s katerimi se primerljiv namenski krmilnik ne more povezati.

Slabosti so lahko naslednje:

- mehanske lastnosti in geometrijske omejitve robota niso tovarniško dodane v *firmware* krmilja, zato je večja možnost napak in poškodb pri zagonu;
- manjša je performansa sistema, ker (ali če) ta ni optimalno sestavljen;
- namenski robotski sistem ima navadno pripravljeno programsko opremo za hitrejši zagon standardnih robotskih opravil (inicializacija položaja (*homing*), pick-and-place, zlaganje (*stacking*) predmetov v različne prostorske postavitve, sinhronizacija s tekočim trakom), PAC platforma pa je univerzalna in je potrebno takšne programe razviti od začetka. Seveda pa je v našem primeru naloga precej nestandardna in, kot omenjeno, potencialno neizvedljiva na namenski platformi. Za platformo Logix sicer so na voljo izdelane programske predloge za standardna robotska opravila, ki pa niso bile uporabljene pri razvoju tega projekta.

Naša naprava bo posnemala model robota SCARA *Afast Robotics*, katerega dimenzije so prikazane na sliki 2.15.



Slika 2.15: Shema dimenzij SCARA robota *Afast*

2.3 Izvedba programske opreme

Celotna programska oprema bo torej realizirana na PLC platformi. V grobem jo predstavlja bločna shema na sliki 2.16. Kota nihala β_x in β_y sta z ustrezno **transformacijo** pretvorjena v neodvisna kota α_x in α_y, ki predstavljata projekcijo nihala na ravnini *x*-*z* in *y*-*z*. Z odvajanjem (blok **diferenciator**) in nizkopasovnim filtrom (**LPF**) dobimo tudi informacijo o kotni hitrosti ω_x in ω_y. Bistven člen je seveda **regulator** – ta ima dva neodvisna kanala, ki krmilita gibanje po koordinati x in y.

Vhodi posameznega regulatorja so:

- trenutna in referenčna položaj *x* oz. *y,*
- trenutna hitrost v_x oz. v_y,
- vrednost kota α_x oz. α_y,
- kotna hitrost $\omega_{\alpha x}$ oz. $\omega_{\alpha y}$.

(Posamezni tipi regulatorjev lahko potrebujejo manj vhodnih spremenljivk.)

Izhod pa je pospešek "vozička", torej pivota nihala v smeri *x* (a_x ali *x''*) oz. *y* (a_y ali *y''*).

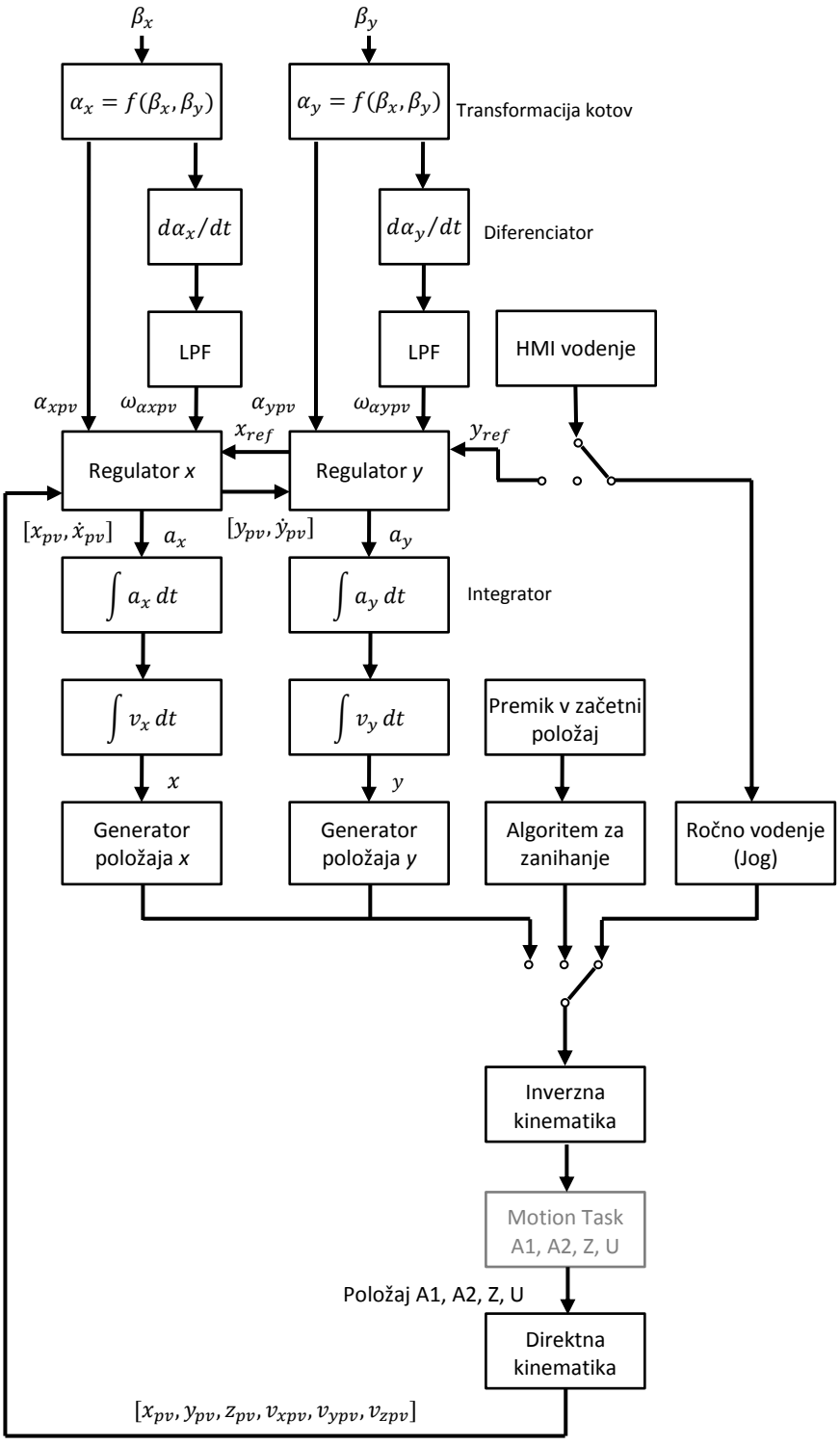
Z dvojno integracijo (blok **integrator**) a_x in a_y pridemo do želenega položaja *x* in *y*, ki ji mora sistem slediti. **Generator položaja** imenujemo programski blok, ki posreduje želen položaj ustreznim instrukcijam za izvajanje gibov.

Blok za **inverzno kinematiko** pretvarja ukaze za gibanje iz kartezičnega v SCARA koordinatni sistem.

»**Motion Task**« je sistemski del PLC programske opreme, ki dejansko izvaja gibanje, blok za **direktno kinematiko** pa sproti izračunava trenuten položaj (hitrost in pospešek) prijemala robota v *x*, *y* in *z* koordinatah.

Del programa je namenjen **ročnemu vodenju** robota s HMI vmesnikom. HMI ukazi za vodenje v smeri *x* in *y* postanejo ukazi za nastavljanje referenčne vrednosti x_{ref} in y_{ref}, ko je regulator aktiven.

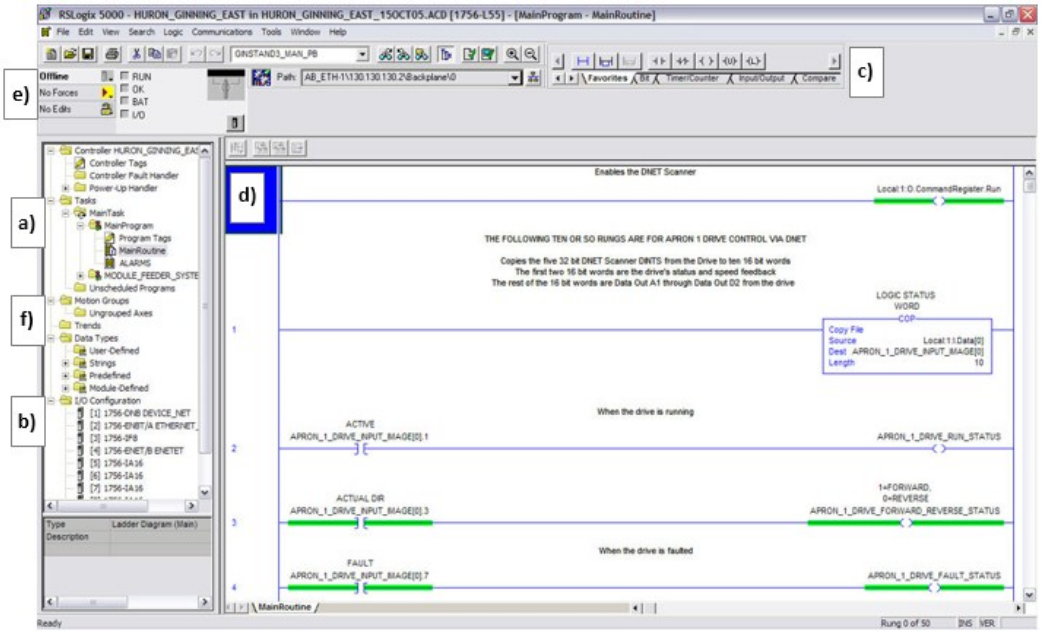
Potreben je tudi **algoritem za zanihanje nihala**, detekcijo navpične lege in vklop regulatorja, ki pa tu še ni podrobneje razdelan.



Slika 2.16: Bločna shema programa

2.4 Izvedba pozicioniranja na izbrani platformi

V tem poglavju bomo podali osnove krmiljenja gibanja in kinematike na platformi Logix in povezavo s simulacijo predstavljenega sistema (nekateri koraki so poenostavljeni).
Orodje za programiranje PLC družine Logix je *RSLogix 5000*. Omogoča izdelavo delovnega programa v »*Offline*« in »*Online*« načinu v 4 različnih programskih jezikih (*Ladder Logic – LD, Function Blocks – FB, Structured Text – ST* in *Sequential Function Chart – SFC* ter konfiguracijo podrejenih (I/O in drugih) naprav na vodilu Ethernet/IP (v našem primeru so to servopogoni Kinetix).

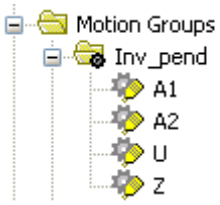


Legenda: a) mape s programi, b) mape z I/O napravami, c) menu z instrukcijami, d) urejevalnik programa (lestvična logika), e) nadzorni menu krmilnika, f) mapa za objekte krmilnika gibanja (*Motion Groups*).

Slika 2.17: Okno RSLogix 5000

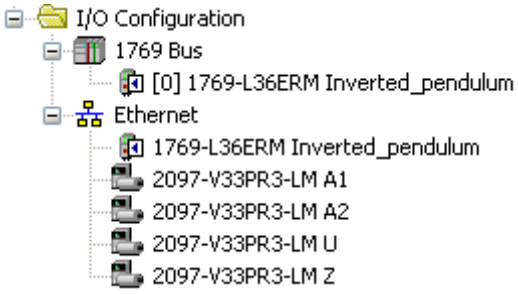
Konfiguracija programa krmilnika za delo s servosistemom je naslednja:

1. Za vsako os gibanja je potrebno kreirati objekt *Axis* v mapi *Motion Groups*. Ta objekt je skupek analognih veličin kot so položaj, hitrost, pospešek… in binarnih stanj, kot so mirovanje, gibanje, pospeševanje, napaka… V našem primeru so to osi **A1**, **A2**, **Z** in **U**.



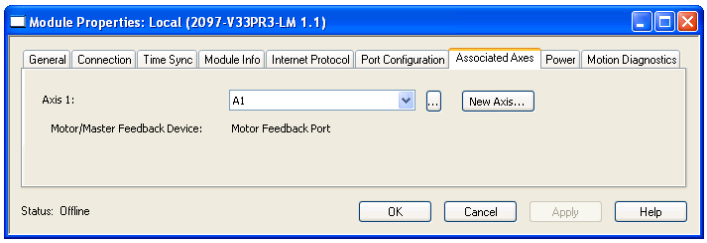
Slika 2.18: Mapa *Motion Groups* in objekti *Axis*

2. Za vsako os je potrebna logična povezava s servoregulatorjem z motorjem, ki ustvarja gibanje. Servoregulator predstavlja I/O napravo, povezano s PLC preko TCP/IP vodila. I/O naprave dodamo v mapo I/O. Iz menija naprav je potrebno izbrati napravo in vpisati njen IP naslov. Na samem regulatorju je potrebno zgolj določiti IP naslov (z uporabo tipk ali DHCP/BOOTP protokola). Ob zagonu se PLC poskuša povezati z nastavljenim IP naslovom – če ga najde in je tip naprave s tem naslovom enak definiciji v I/O mapi, postane PLC "lastnik" naprave in lahko z njo upravlja.



Slika 2.19: I/O mapa s podrejenimi napravami

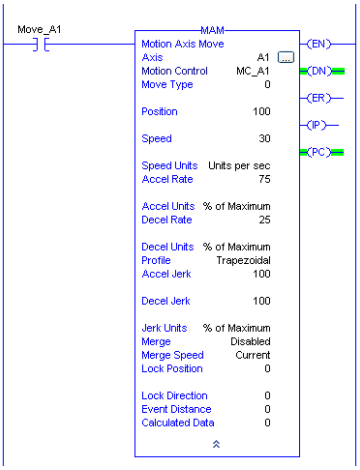
3. V meniju servoregulatorja je potrebno še narediti logično povezavo z *Axis* objektom – s tem je konfiguracija zaključena.



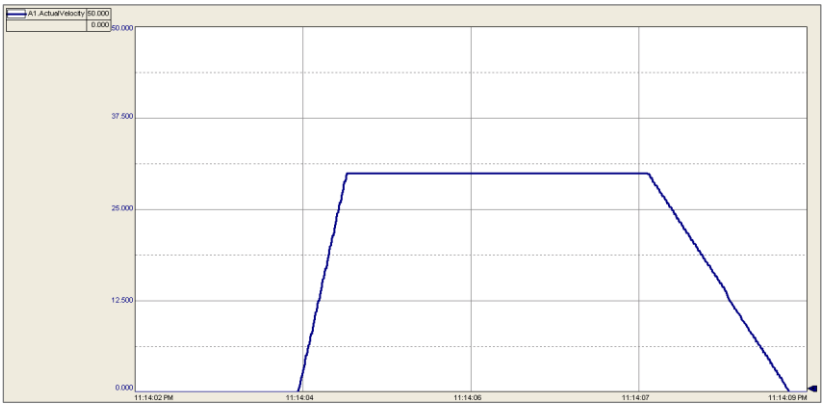
Slika 2.20: Povezava med objektom *Axis* in pogonom (tip 2097-V33PR3-LM)

4. Za programsko generiranje gibanja je na voljo množica instrukcij (združene v mapo *Motion Instructions*). Vsaka v programu uporabljena instrukcija se nanaša na izbran

Axis objekt, le-ta pa na fizični pogon. Preprost primer ukaza je premik s točke A na točko B z blokom MAM (*Motion Axis Move*). Bitna instrukcija na začetku vrstice deluje kot stikalo, ki sproži MAM ukaz. Ključni parametri MAM so: ciljni *Axis* objekt (npr. A1), tip giba (0 = absolutni), končna položaj (100°), hitrost (30°/s), pospešek (75 % max. vrednosti = 7500°/s^2), pojemek (25 % max. vrednosti = 2500°/s^2) – rezultat izvedene instrukcije s temi parametri prikazuje slika 2.22. Ko je premik končan (bit PC=1), lahko ukaz poljubno ponovimo. Namesto konstant lahko v instrukcijo vnesemo spremenljivke (Tags), katerih vrednosti se lahko dinamično spreminjajo, kot rezultat drugih operacij programa ali direktnega uporabniškega vnosa (preko HMI).



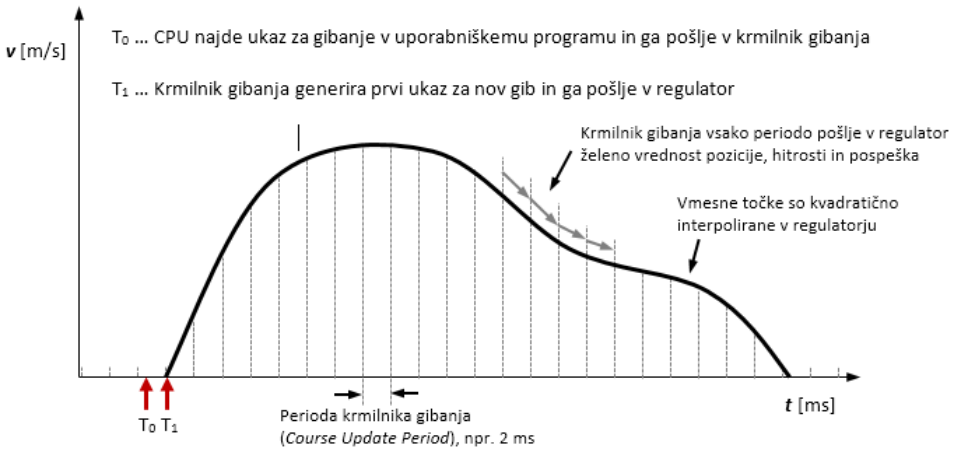
Slika 2.21: Programska vrstica lestvične logike z ukazom MAM



Slika 2.22: Rezultat gibanja z ukazom MAM – časovni diagram hitrosti A1 (v °/s, razdelek 2 s/12,5°). Ta in nadaljnji izrisi so narejeni s funkcijo Trend v orodju RSLogix 5000.

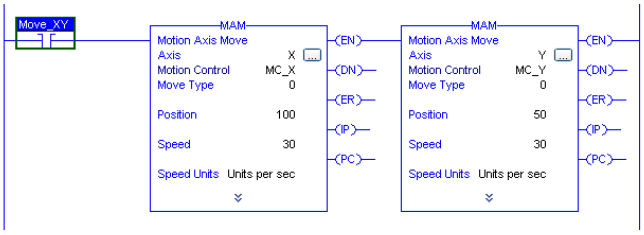
5. Ko sprožimo MAM (ali katerokoli drugo Motion instrukcijo) v uporabniškem programu, se ta izvede v t.i. *Motion Task* rezini časovnega cikla CPU. Motion Task se

izvaja z neko periodo (ki je nastavljiva, v našem primeru 2 ms, in ima najvišjo prioriteto med opravili procesorja), ob vsaki izvedbi cikla pa pošlje servoregulatorjem TCP/IP paket z informacijo o želenem položaju, hitrosti in pospešku. Vmesne vrednosti pogon interpolira z lastno CPU enoto. Dejanski položaj, hitrost in pospešek so, kot že opisano, regulirane preko PID zank. TCP/IP paketi so opremljeni s časovno značko (resolucije 1 us), s čemer je možno kompenzirati naključne zakasnitve pri transportu po Ethernet omrežju (t.i. *Jitter*).

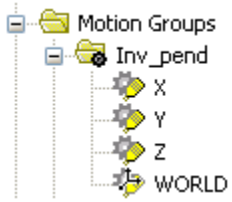


Slika 2.23: Profil gibanja je definiran v diskretnih intervalih izvajanja Motion Task rezine in interpoliran v servopogonu

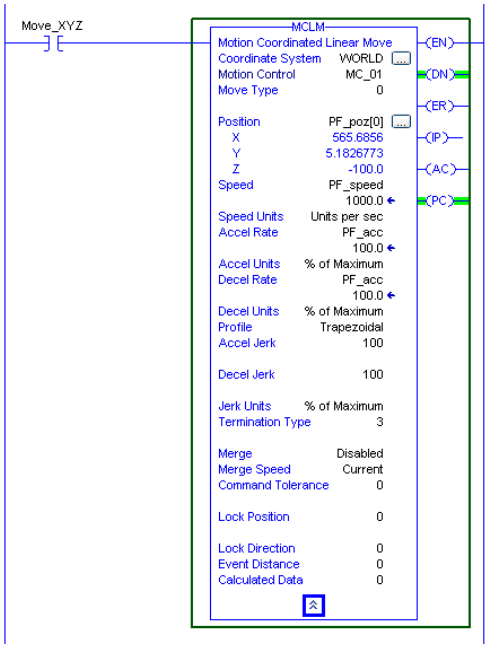
6. Koordinirano gibanje več osi: predpostavimo, da sta X in Y osi postavljeni v kartezični sistem (slika 1.7) in želimo premik iz točke A(x_1,y_1) v B(x_2,y_2). Program na sliki 2.24 bo izvedel takšen premik, vendar ne po premici. V ta namen je potrebno v Motion Group definirati koordinatni sistem, ki je, podobno kot *Axis*, objekt, na katerega se bodo sklicevale ustrezne instrukcije. Uporabljen ukaz je sedaj MCLM (*Motion Coordinated Linear Move*), ki operira nad koordinatnim sistemom WORLD (in pripadajočimi osmi gibanja). Rezultat je premik v ravnini x-y (ali v prostoru x-y-z) po ravni črti. Možne so seveda tudi druge oblike gibanja v 2D ali 3D prostoru.



Slika 2.24: Dva ukaza MAM nimata mehanizma za medsebojno sinhronizacijo



Slika 2.25: Objekt WORLD je kartezični koordinatni sistem, sestavljen iz osi X, Y in Z



Slika 2.26: MCLM instrukcija

7. Inverzna kinematika: Krmiljenje ne-kartezičnega robota (kot je SCARA) zahteva uporabo inverzne kinematike. Definicija je naslednja:

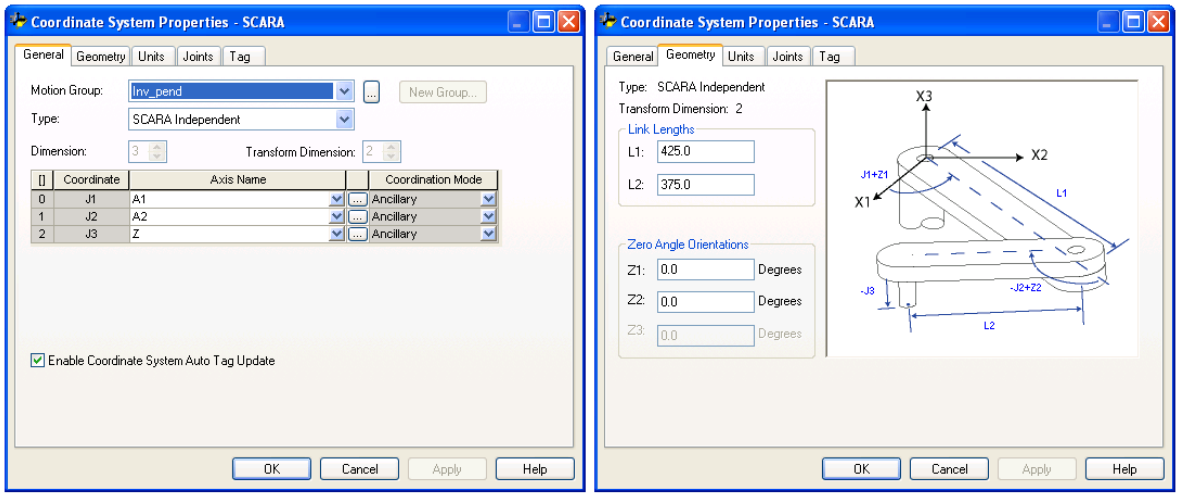
   Inverzna kinematika določa položaj sklepov glede na dan položaj (in orientacijo) efektorja (prijemala) - ima lahko več možnih ali pa nobene rešitve:

$$(A1, A2, Z) = f\big(P(x, y, z)\big). \tag{2.2}$$

   Direktna kinematika določa položaj (in orientacijo) efektorja glede na dane položaje sklepov - ima samo eno rešitev:

$$P(x, y, z) = f(A1, A2, Z). \tag{2.3}$$

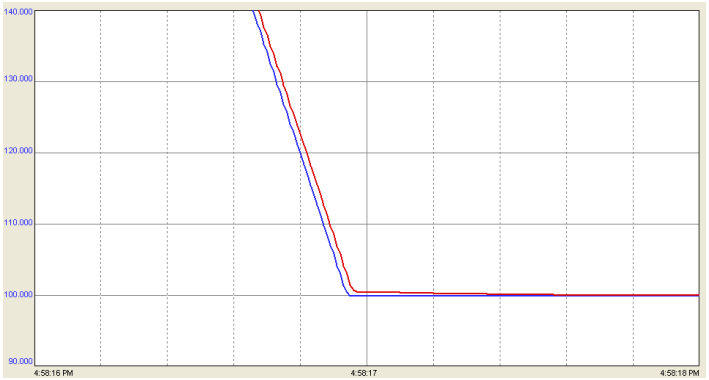
Obe transformaciji sta že vdelani v krmilnik gibanja Logix. Potrebno je definirati nov koordinatni sistem, ki opisuje SCARA geometrijo, in ga sestavljajo osi A1, A2, in Z (os U, ki predstavlja vrtenja orodja okoli osi Z, ni predmet transformacije).



Slika 2.27: Konfiguracija SCARA koordinatnega sistema

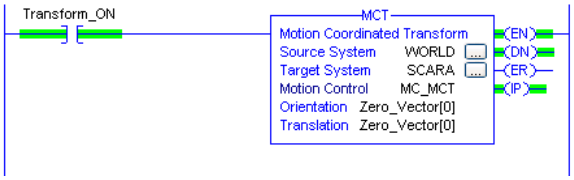
Na sliki 2.27 je razvidna pripadnost osi koordinatnemu sistemu tipa *SCARA Independent* (zavihek *General*) ter geometrijske lastnosti (zavihek *Geometry*), to so dolžina sklepov (*Link Lengths*) L1 in L2 in izhodiščna orientacija (*Zero Angle Orientations*). Geometrija je povzeta po sliki 2.15.

Že opisani kartezični koordinatni sistem (točka 6.) ostane kot referenčni sistem, v katerem bomo še naprej programirali gibe robota. Pripadajoče osi so X, Y in Z; vendar pa X in Y pa nista več povezani s fizičnimi pogoni. Brez te povezave postaneta virtualne osi (*Virtual Axis*). Virtualne osi se vedejo enako kot fizične v smislu odziva na programske ukaze. V primeru, da so povratne zanke sistema pogona in motorja dobro uglašene (to je pogoj za zadovoljivo delovanje robota in drugih togih mehanizmov), je razlika majhna in za naš namen relativno nepomembna. Spodnja slika kaže povečan izsek odziva na ukaz MAM za oba tipa osi (časovni zamik je narejen programsko za lažjo primerjavo).



Slika 2.28: Primerjava odziva položaja [v °] fizičnega pogona (rdeče) in virtualne osi (modro) na ukaz MAM, hitrost 200°/s, razdelek 0,2 s/10°

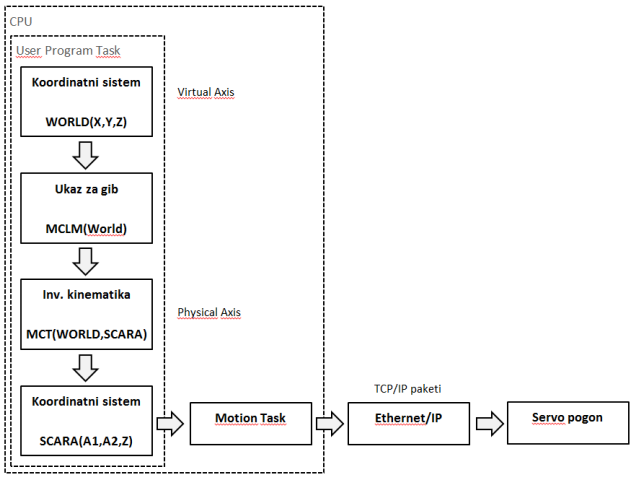
Inverzno kinematiko izvedemo z ukazom MCT (*Motion Coordinated Transform*), katere parametra sta referenčni (*Source System,* v našem primeru WORLD) in ciljni (*Target System*, SCARA) koordinatni sistem. Praviloma je ta ukaz lahko stalno aktiven, kot kaže slika 2.29.



Slika 2.29: Instrukcija MCT (*Motion Coordinated Transform*)

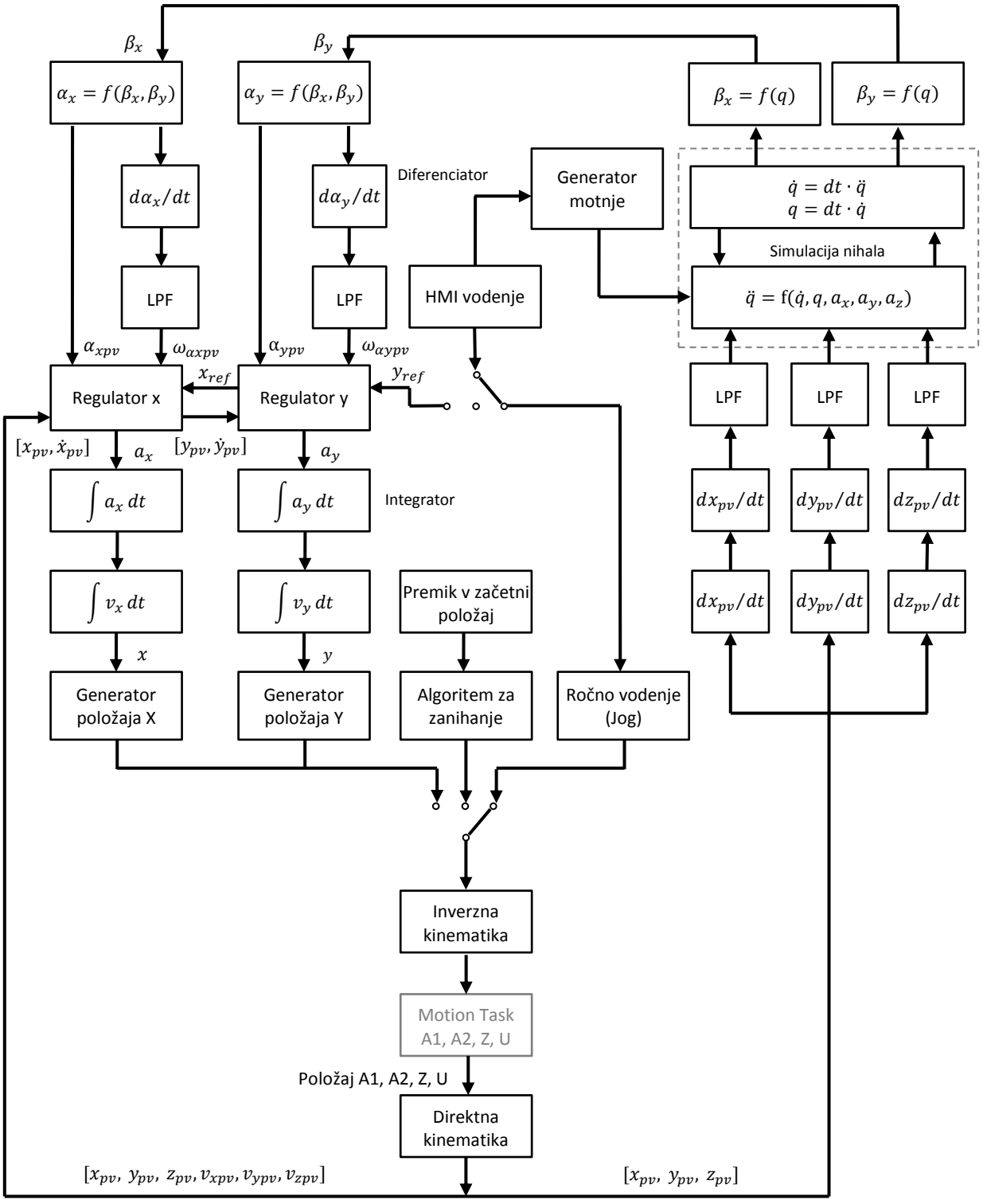
Komplementarna instrukcija izvaja direktno kinematiko, to je MCTP (*Motion Calculate Transform Position*).

Celotno zaporedje dogodkov v sistemu je sedaj takšno:



Slika 2.30: Bločna shema operacij pri uporabi inverzne kinematike

8. Za potrebe simulacije robota lahko tudi fizične pogone A1, A2, Z in U nadomestimo z virtualnimi. Program za krmiljenje gibanja robota, ki ga bomo razvili v nadaljevanju, bo potreboval le majhne spremembe za adaptacijo na realno napravo, če bo tudi simulacija vhodnih podatkov dovolj realna. V našem primeru je to simulacija odziva nihala (kota α_x in α_y) na premike robota, kot to prikazuje bločna shema 2.16. To predstavlja temeljno razliko med uporabljenim pristopom in običajno simulacijo v npr. paketu *Matlab Simulink*. Algoritem, razvit na platformi *Matlab,* je sicer možno povezati z realno napravo preko I/O strojne opreme, vendar takšna kombinacija ne bo sprejeta v industrijski praksi kot zanesljiva, integrirana naprava. Prenos programske kode iz ene platforme (*Matlab*) na drugo (PLC ali namenski robotski krmilnik) pa lahko predstavlja večji izziv kot sam razvoj algoritma. V našem primeru je možen enostaven, programski preklop med simulacijo in realno napravo, lahko preizkušamo kombinacijo obojega (kot bo pokazano v nadaljevanju) v različnih razmerjih, lahko pa tudi oba načina delujeta paralelno.



Slika 2.31: Bločna shema simulacije – krmilni program je enak kot za realno napravo (slika 2.16)

2.5 Vizualizacija

Vizualizacija naprave je pomembna, saj je sistem kompleksen in dinamičen, po drugi strani pa intuitivno lahko dojemljiv – tudi nepoučen opazovalec lahko presodi, ali se sistem obnaša po pričakovanjih.

Za vizualizacijo smo izbral orodje *RSTestStand*, proizvajalca *Rockwell Automation*. *RSTestStand* omogoča interaktivni razvoj in preizkušanje krmilnega programa na namizju – vhodne in izhodne veličine PLC zamenja s simuliranim procesom. Zgradimo lahko virtualno upravljalno konzolo s stikali in signalnimi svetilkami in opravimo sprejemljivostne teste s pomočjo avtomatiziranih testnih procedur.

Nekatere lastnosti:

- povezava s PLC preko OPC-standarda;
- knjižnica diskretnih in zveznih funkcij za izgradnjo simulacije (npr. zakasnilni člen drugega reda, mrtvi čas, premikalni register, generator funkcij, integrator, flip-flop);
- knjižnica naprav (črpalka, ventil, sklopka, reaktor, temperaturni proces...);
- elementi HMI vmesnika (tipka, stikalo, signalna svetilka, digitalni in analogni instrument, potenciometer…);
- 3D izris (po standardu *OpenGL*), sestavljanje hierarhičnih objektov iz osnovnih geometrijskih teles (t.i. *primitives*, to so kvader, krogla, valj, stožec…) ali uvoženih modelov formata *dfx;*
- animacija parametrov 3D elementov (položaj, rotacija, velikost, barva…) z neodvisnimi spremenljivkami ali časovnimi funkcijami (t.i. *frame keying*);
- skriptni jezik VBA in diagram poteka (*flowchart*) za avtomatizacijo procedur;
- simulacija diskretnih procesov (tok materiala, polizdelkov v proizvodnji, delovnih operacij…);
- izvoz in uvoz podatkov, spremenljivk in simulacijskih modelov v *Excel* in XML-format.

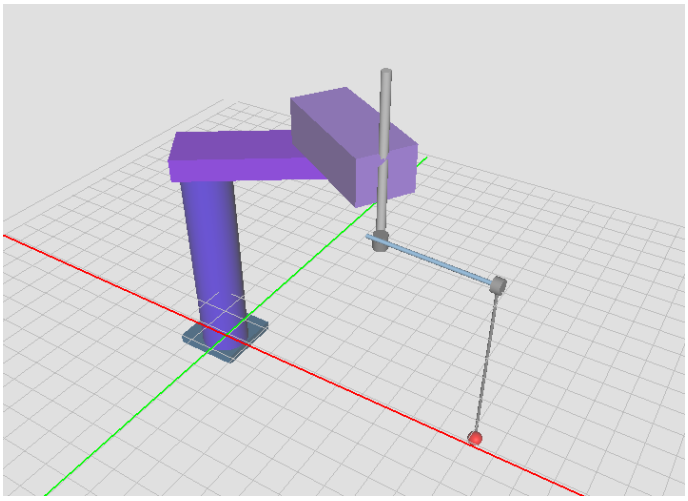
Za namen naloge smo uporabili samo del teh funkcij, in sicer za:

- 3D vizualizacijo robota in invertiranega nihala;
- HMI (*Human Machine Interface*) vmesnik, ki omogoča ročno vodenje virtualnega robota, start in prekinitev poskusa in nastavitev nivoja simulirane motnje.

V tej vlogi se je RSTestStand pokazal kot učinkovita in primerna rešitev, čeprav ima nekaj slabosti:

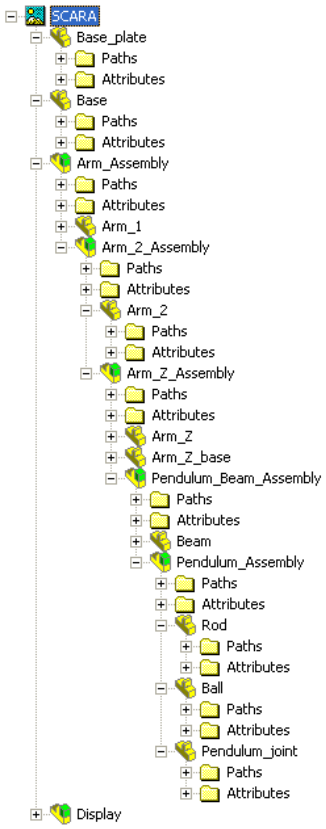
- 3D izris v načinu *OpenGL* je zastarel (v primerjavi z novejšimi možnostmi, ki jih ponuja 3D izris v realnem času) in ne deluje najbolje – mestoma se objekti, ki naj bi bili skriti pod "površino", prikažejo nad njo. Preklop med strojnim in programskim načinom *OpenGL* pospeševanja pri tem ni pomagal;
- očitno je, da je nadaljnji razvoj paketa ustavljen – zadnja kompatibilna verzija OS je Windows XP SP2.

Model realnega robota je narejen po tehnični risbi robota Afast Robotics (slika 2.15).



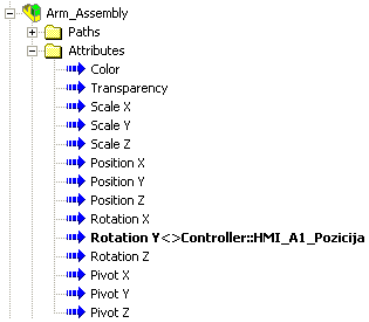
Slika 2.32: Virtualni robot z nihalom, izrisan v programu RSTestStand

Model je sestavljen iz osnovnih geometrijskih oblik (kvader, valj, krogla) v 3D editorju programa RSTestStand. Posamezni gradniki so hierarhično gnezdeni in povezani v skupine. Koren hierarhije predstavlja montažna plošča in podnožje (valjasti del), ki so na absolutni koordinati (0,0,0) in se ne premikajo. Zadnji odvisni člen predstavlja nihalo, ki je sestavljeno iz valja in krogle (rdeče barve). Vmesni gradniki so dva kvadra (sklepa A1 in A2) in štirje valji (os Z, vodoravni nosilec in sistem enkoderjev).



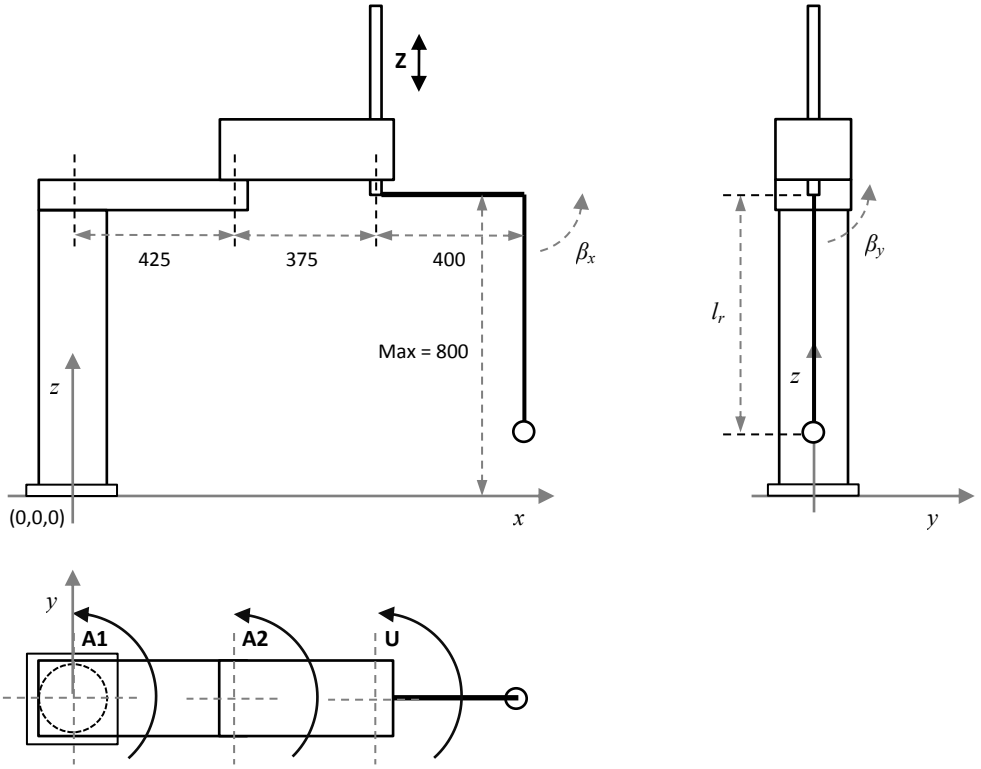
Slika 2.33: Hierarhija 3D elementov, ki sestavljajo robot

Objekti ali skupine objektov imajo možnost povezave atributov rotacije, translacije in velikosti (po koordinatah *x*, *y* ali *z*) s spremenljivkami, katerih vir je krmilnik ali interni modeli oz. generatorji RSTestStand okolja.



Slika 2.34: Povezava objektov roke robota z osjo A1 (koordinatni sistem v RSTestStand je zasukan glede na našo definicijo na sliki 2.35 – višina je *y* in širina *z*)

Tako lahko animiramo sklepe robota (A1, A2, Z, U), oba kota nihala (β_x, β_y) in dolžino nihala (l_r).



Slika 2.35: Shema dimenzij robota [mm]

Drugi del RSTestStand simulacije je vmesnik za upravljanje z robotom in poskusom (start, prekinitev in reset napak). Sestavlja ga panel s 14 tipkami (*momentary push button*), signalnimi svetilkami in dvema drsnikoma (potenciometroma). Tudi ti objekti so povezani z dinamičnimi PLC spremenljivkami (bitne (*Boolean*) za tipke in svetilke, realne (*Floating point*) za potenciometre). Ta HMI vmesnik omogoča:

- neodvisno krmiljenje vsake posamezne osi robota (t.i. *Jogging*) z nastavljeno hitrostjo v pozitivno in negativno smer (modre tipke **A1 FWD**, **A1 REV**, **A2 FWD**, … **U REV**, **Z UP**, **Z DOWN**);
- krmiljenje robota neodvisno po *x* in *y* koordinatni osi z nastavljeno hitrostjo (rumene tipke **X FWD**, **X REV**, **Y FWD**, **Y REV**);
- nastavitev hitrosti za *Jogging* funkcijo (**JOG SPEED**) od 0 do 100 %;
- vklop poskusa (zelena tipka **START**);

– izklop poskusa in reset napak (rdeča tipka **RESET**);
– nastavitev nivoja motnje (**NOISE**) od 0 do 100 %.

Signalizacija pa javlja:

– ročno vodenje aktivno (**JOGGING**);
– dosežen limit katerega koli sklepa robota (**LIMIT**);
– vklop poskusa regulatorja (**CNTRL ON**);
– napako (**FAULT**).

Slika 2.36: Vmesnik za upravljanje

Povezava med programom in PLC je OPC (*OLE for Process Control*) standard.

Slika 2.37: Logo OPC združenja (https://opcfoundation.org)

Kot OPC strežnik je uporabljen program RSLinx, ki je sicer uporabljen tudi za povezavo med krmilnikom in orodjem RSLogix 5000 po vodilu Ethernet/IP (RSLinx je del instalacije RSLogix 5000). RSLinx navadno teče v ozadju kot servis brez GUI vmesnika. RSTestStand je torej OPC *klient*. Hitrost komunikacije (osveževanje spremenljivk) smo nastavili na 50 ms.

Vizualizacija je bila preizkušena s preprostim testnim programom, v katerem so bile spremenljivke animirane z uporabo PLC časovnikov (timerjev). S preizkusom smo potrdili, da je izdelana vizualizacija res uporabna za nadaljnji razvoj naloge:

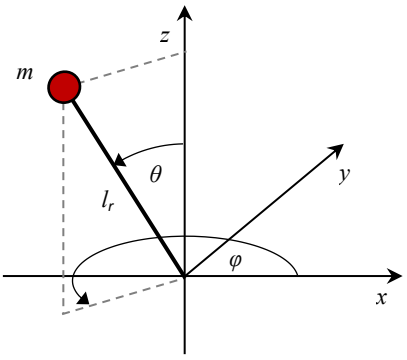
- odziv 3D modela RSTestStand zadostuje, da je možno realistično spremljati hitrosti gibanja, ki ekvivalentne realnim do 4 m/s. Povprečje izrisa na uporabljeni PC opremi je 22 FPS (*Frame-per-second*) oz. slik na sekundo;
- med gibanjem je možno z miško premikati 3D sceno (»kamero«) v smislu vrtenja (*Rotate*), približevanja (*Zoom*) in premikanja (*Pan*) brez reduciranja FPS vrednosti;
- odziv tipk in signalnih svetilk povsem zadostuje za upravljanje (odziv je reda 0,2 s);
- zdi se, da na začetku omenjene anomalije 3D izrisa ne degradirajo s kompleksnostjo modela.

3 Simulacija nihala

Simulacija nihala povezuje položaj pivota nihala x_p, y_p in z_p (ki je funkcija položajev, hitrosti in pospeškov osi A1, A2, Z in U) s položajem kotov β_x in β_y – kot kaže bločna shema 2.31. Vizualizacija nihala, v kateri je poleg kotov možno animirati tudi dolžino nihala (l_r), je bila pripravljena že v prejšnjem poglavju (2.5). Iz slik 2.32 in 2.35 vidimo, da konstrukcija robota omogoča dovolj prostora za gibanje nihala okoli ravnovesne točke in za zanihanje v navpično lego.

Za realizacijo regulatorja zadostuje model dveh neskloplјenih nihal, ki sta projecirani na ravnini *x*-*z* in *y*-*z*, želeli pa smo razviti kompleten model.

Ena možnost za praktično izvedbo simulacije je uporaba dinamičnih gradnikov v okolju RSTestStand, vendar smo se kasneje odločili za realizacijo na krmilniku Logix; simulacija je torej del PLC programa, ki se izvaja hkrati s programom za vodenje robota in regulacijo. Izbrana hitrost simulacije je bila 100 ciklov na sekundo (perioda izvajanja programa 10 ms), brez težav pa bi lahko dobili do 400 ciklov/s (osveževanje slike na vizualizaciji pa je, kot že omenjeno, nastavljeno na 50 ms).



Slika 3.1: Sferično nihalo, definicija spremenljivk

Razvoj simulacije naj bi potekal takole:

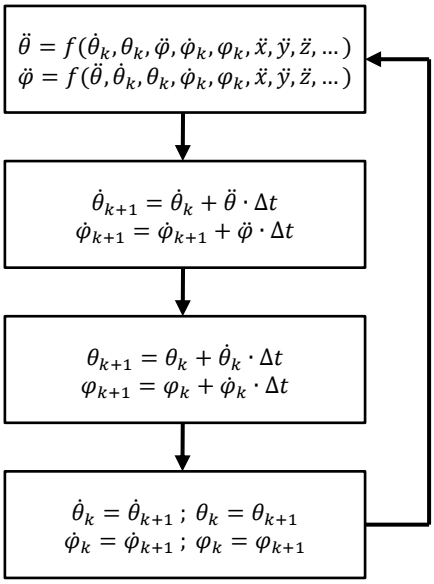
1. razvoj matematičnega modela nihala z diferencialnimi enačbami (Lagrangeova funkcija);

2. pretvorba enačb v eksplicitno obliko, kjer je najvišji odvod odvisne spremenljivke enak (nelinearni) funkciji nižjih odvodov, torej:

$$\ddot{q} = f(\dot{q}, q, \dots) \tag{3.1}$$

kjer je q posplošena koordinata;

3. numerično reševanje diferencialnih enačb z Eulerjevo metodo, pri kateri najprej izračunamo vrednost najvišjih odvodov z izrazom (3.1), nato s (približno) integracijo izračunamo vse nižje odvode, jih vstavimo nazaj v izraz (3.1) in to zanko ponavljamo – kot kaže slika 3.2.



Slika 3.2: Eulerjeva metoda reševanja diferencialnih enačb

Spremenljivke (na sliki 3.1), ki jih potrebujemo za opis nihala, so:

x, y, z … kartezični koordinatni sistem,

φ, θ ... sferična kota nihala,

l_r … dolžina nihala,

m … točkasta masa,

g … gravitacija.

Odnos med kartezičnimi in sferičnimi koordinatami je:

$$x = -l_r sin(\theta)\cos(\varphi), \tag{3.2}$$

$$y = -l_r sin(\theta)\sin(\varphi), \tag{3.3}$$

$$z = l_r cos(\theta). \tag{3.4}$$

Lagrangeova funkcija sistema je razlika med kinetično (*T*) in potencialno energijo (*V*) sistema (mehansko trenje, upor in izgube v tej izpeljavi zanemarimo):

$$L = T - V. \tag{3.5}$$

T je našem primeru:

$$T = \frac{m}{2}(\dot{x}^2 + \dot{y}^2 + \dot{z}^2). \tag{3.6}$$

Z vstavljanjem izrazov (3.2) do (3.4) in odvajanjem dobimo:

$$\dot{x} = -l_r\dot{\theta}cos(\theta)\,cos(\varphi) - l_r\dot{\varphi}sin(\theta)\,sin(\varphi), \tag{3.7}$$

$$\dot{y} = -l_r\dot{\theta}cos(\theta)\,sin(\varphi) + l_r\dot{\varphi}sin(\theta)\,cos(\varphi), \tag{3.8}$$

$$\dot{z} = -l_r\dot{\theta}sin(\theta). \tag{3.9}$$

In naprej:

$$\dot{x}^2 = l^2\dot{\theta}^2\,cos^2(\theta)\,cos^2(\varphi) - 2l^2\theta\dot{\varphi}sin(\theta)cos(\theta)\,sin(\varphi)\,cos(\varphi) + l^2\dot{\varphi}^2sin^2(\theta)sin^2(\varphi), \tag{3.10}$$

$$\dot{y}^2 = l^2\dot{\theta}^2\,cos^2(\theta)\,sin^2(\varphi) + 2l^2\theta\dot{\varphi}sin(\theta)cos(\theta)\,sin(\varphi)\,cos(\varphi) + l^2\dot{\varphi}^2sin^2(\theta)cos^2(\varphi), \tag{3.11}$$

$$\dot{z}^2 = {l_r}^2\dot{\theta}^2sin^2(\theta). \tag{3.12}$$

Tako je *T*:

$$T = \frac{m}{2}l_r^2\Big(\dot{\theta}^2\,cos^2(\theta)\,cos^2(\varphi) - 2\theta\dot{\varphi}sin(\theta)cos(\theta)\,sin(\varphi)\,cos(\varphi) + \dot{\varphi}^2sin^2(\theta)sin^2(\varphi) + \dot{\theta}^2\,cos^2(\theta)\,sin^2(\varphi) + 2\theta\dot{\varphi}sin(\theta)cos(\theta)\,sin(\varphi)\,cos(\varphi) + \dot{\varphi}^2sin^2(\theta)cos^2(\varphi) + \dot{\theta}^2\,sin^2(\theta)\Big) \tag{3.13}$$

Po poenostavitvi pa:

$$T = \frac{m}{2}l_r^2\Big(\dot{\theta}^2 + \dot{\varphi}^2sin^2(\theta)\Big). \tag{3.14}$$

Potencialna energija pa je:

$$V = mgz = mgl_rcos(\theta). \tag{3.15}$$

Celotna Lagrangeova funkcija je:

$$L = \frac{m}{2}{l_r}^2\Big(\dot{\theta}^2 + \dot{\varphi}^2sin^2(\theta)\Big) - mgl_rcos(\theta). \tag{3.16}$$

Rešitev za θ'' dobimo z Euler-Lagrangeovo enačbo (prim. [10]):

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}}\right) - \frac{\delta L}{\delta \theta} = 0, \tag{3.17}$$

kjer je:

$$\frac{\delta L}{\delta \theta} = m{l_r}^2 \dot{\varphi}^2 sin(\theta)cos(\theta) + mgl_r sin(\theta), \tag{3.18}$$

$$\frac{\delta L}{\delta \dot{\theta}} = m{l_r}^2 \dot{\theta}, \tag{3.19}$$

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}}\right) = m{l_r}^2 \ddot{\theta}. \tag{3.20}$$

Vstavimo v (3.17) in dobimo:

$$m{l_r}^2 \ddot{\theta} - m{l_r}^2 \dot{\varphi}^2 sin(\theta)cos(\theta) - mgl_r sin(\theta) = 0. \tag{3.21}$$

Po preureditvi pa:

$$\ddot{\theta} = (\dot{\varphi}^2 sin(\theta)cos(\theta) + gsin(\theta)/l_r). \tag{3.22}$$

Rešitev za φ'' dobimo z Euler-Lagrangeovo enačbo (prim. [10]):

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\varphi}}\right) - \frac{\delta L}{\delta \varphi} = 0, \tag{3.23}$$

kjer je:

$$\frac{\delta L}{\delta \varphi} = 0, \tag{3.24}$$

$$\frac{\delta L}{\delta \dot{\varphi}} = m{l_r}^2 \dot{\varphi} sin^2(\theta), \tag{3.25}$$

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\varphi}}\right) = m{l_r}^2 \ddot{\varphi} sin^2(\theta) + 2m{l_r}^2 \dot{\varphi}\dot{\theta} sin(\theta)cos(\theta). \tag{3.26}$$

Vstavimo v (3.23) in dobimo:

$$m{l_r}^2 \ddot{\varphi} sin^2(\theta) + 2m{l_r}^2 \dot{\varphi}\dot{\theta} sin(\theta)cos(\theta) = 0. \tag{3.27}$$

Po preureditvi:

$$\ddot{\varphi} = -2\dot{\varphi}\dot{\theta} cos(\theta)/sin(\theta). \tag{3.28}$$

Vidimo, da izraz (3.28) predstavlja težavo za reševanje z Eulerjevo numerično metodo, ker vsebuje singularnost v točki $sin(\theta) = 0$, torej v ravnovesni ($\theta = 0$) in navpični legi ($\theta = \pi$). Opisana metoda reševanja je zato tu neuporabna.

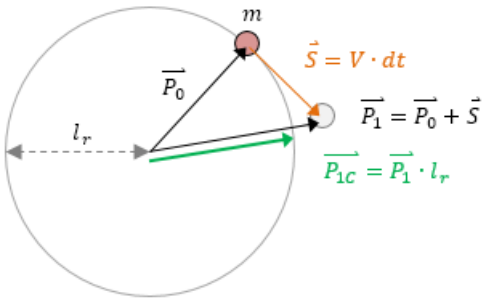
Po več poskusih smo razvili alternativno metodo, kjer problem rešujemo v *x-y-z* koordinatnem sistemu kot gibanje masnega delca *m* po površini sfere z radijem l_r.

- V nekem trenutku ima masa m položaj P0(x,y,z), ki je na površini sfere z radijem lr, in hitrost V, ki je pravokotna na normalo sfere oz. P0.
- Pod vplivom sile F(x,y,z) naredi masa m v času dt pot S(x,y,z) do novega položaja P1(x,y,z).
- P1 ni več na površini sfere, zato izvedemo korekcijo:

$$P_{1c} = P_1 \cdot l_r / |P_1|, \tag{3.29}$$

  pod predpostavko, da gre $P_{1c} \rightarrow P_1$, ko gre $dt \rightarrow 0$. V našem primeru je *dt* enak periodi izvajanja programa, stremimo torej k čim hitrejšemu izvajanju programa ter računsko enostavnejši rešitvi.
- Nova trenutna položaj P_0 postane enaka P_{1c} (prepis vrednosti).
- Zanko ponavljamo s periodo *dt.*

(Tak model ne upošteva rotacije nihala okoli lastne osi.)



Slika 3.3: Alternativna rešitev problema sferičnega nihala

Algoritem je narejen v strukturiranem tekstu (Logix ST editor) in je naslednji:

```
// dt … perioda izvajanja rutine [0.01ms];
// lr … dolžina nihala [0.6m];
// m … masa [0.3kg];
// f … konstanta trenja [0.02kg/s];
```

```
// g … gravitacija [9.8m/s2].

// V, Vx, Vy, Vz … vektor hitrosti mase m;
// F, Fx, Fy, Fz … vektor sil na maso m;
// a, ax, ay, az … pospeški sistema;
// S, Sx, Sy, Sz … razdalja na površini sfere lr;
// P0, Px0, Py0, Pz0 … trenutna položaj mase m;
// P1, Px1, Py1, Pz1 … nova položaj mase m;
// P1c, Px1c, Py1c, Pz1c … nova položaj mase m, korigirana.

//Vektor sil
if (Vx>0) then Fx:=-V*f-m*ax;
else Fx:=V*f-m*ax; end_if;
if (Vy>0) then Fy:=-V*f-m*ay;
else Fy:=V*f-m*ay; end_if;
if (Vz>0) then Fz:=-V*f-m*az-m*g;
else Fz:=V*f-m*az-m*g; end_if;

//Vektor razdalje, novi položaj in korekcija
Sx:=Vx*dt+Fx*dt*dt/m;
Sy:=Vy*dt+Fy*dt*dt/m;
Sz:=Vz*dt+Fz*dt*dt/m;
Px1:=Px0+Sx;
Py1:=Py0+Sy;
Pz1:=Pz0+Sz;
P1:=sqrt(Px1**2+Py1**2+Pz1**2);
if P1<0.05 then P1:=0.05; end_if;
Px1c:=Px1*lr/P1;
Py1c:=Py1*lr/P1;
Pz1c:=Pz1*lr/P1;

//Novi vektor hitrosti
Vx:=(Px1c-Px0)/dt;
Vy:=(Py1c-Py0)/dt;
```

```
Vz:=(Pz1c-Pz0)/dt;
V:=SQRT(Vx**2+Vy**2+Vz**2);

//Prepis položaja v naslednjo iteracijo
Px0:=Px1c;
Py0:=Py1c;
Pz0:=Pz1c.
```

Sintaksa Logix programa je podoba jeziku C in jo lahko z manjšimi spremembami prenesemo na kakšno drugo platformo.

Vhodi v sistem so pospeški a_x, a_y in a_z. Dobimo jih z dvojnim odvajanjem položaja pivota nihala. Položaj dobimo z ukazom za direktno kinematiko. Izhod sistema je položaj mase nihala P_{x0}, P_{y0}, P_{z0}. Kot vidimo, program popravlja položaj, tako da vedno velja:

$$l_r{}^2 = P_{x0}{}^2 + P_{y0}{}^2 + P_{z0}{}^2. \tag{3.30}$$

Položaj P_{x0}, P_{y0} in P_{z0} je pretvorjen v kota β_x in β_y s temi izrazi:

```
// Pi … 3.1417…;
// Pend_U … rotacija pivota okoli izhodišča (vsota rotacij A1, A2 in U);
// B_x, B_y … kota nihala.

Pend_X:=Px0*COS(Pend_U)+Py0*SIN(Pend_U);
Pend_Y:=-Px0*SIN(Pend_U)+Py0*COS(Pend_U);
if (Pz0>0) then
       B_x:=(-Pi-ASIN(-Pend_X/lr));
else
       B_x:=ASIN(-Pend_X/lr);
end_if;
B_y:=ASIN(Pend_Y/(lr*COS(B_x))).
```

Zgornja obdelava vhodov in izhodov poteka z enako periodo (10 ms) izvajanja kot program modela nihala.

V bližini navpične lege sta β_x in β_y približno enaka α_x in α_y. Ko se nihalo giblje zgolj v ravnini *y-z* (to je v procesu zanihanja), je β_y enak α_y, v izpeljavah pa uporabljamo slednjo oznako.

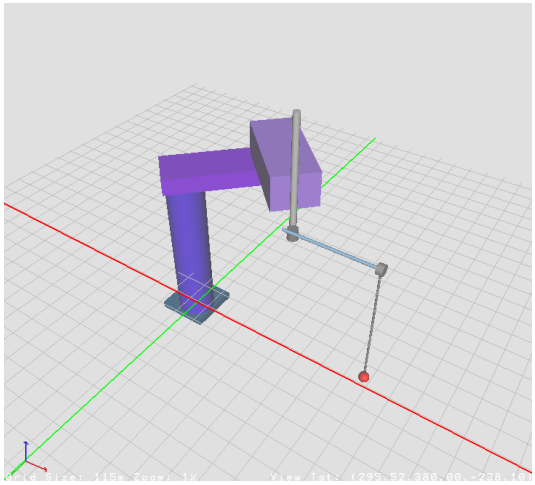
Tudi pri načrtovanju regulatorjev uporabljamo α_x in α_y.

4 Strategija za zanihanje

Z modelom invertiranega nihala lahko začnemo s poskusi za zanihanje nihala v navpično lego.

Strategija za zanihanje temelji na naslednjih izhodiščih:

- Smer zanihanja je v ravnini *y*-*z*, nekako na sredini dosega robota v koordinati *x*, kjer ima ta največji doseg v smeri *y*.



Slika 4.1: Robot v izhodiščni legi

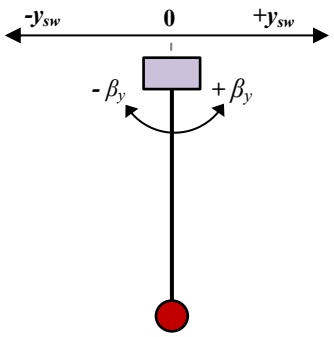
- Temu ustreza tudi izbrani tip robotske roke in izvedba pritrdilnega mehanizma (kot β_y, ki predstavlja odmik v ravnini *y*-*z,* nima omejitve gibanja, medtem ko je kot β_x (v ravnini *x*-*z*) omejen z vodniki za priključitev enkoderjev).
- Pri vzbujanju se sme nihalo gibati samo v izbrani ravnini, sicer pride do kroženja.
- Energijo za vzbujanje je potrebno dovajati v intervalih, ki so enaki naravni frekvenci sistema (ki pa ni konstantna).
- Postopek je uspešen, ko privede nihalo v bližino navpične lege pri dovolj majhni krožni hitrosti, oba parametra sta nastavljiva v programski proceduri. Vgrajen evaluator nato preklopi fazo zanihanja v regulacijo.

Predvidimo lahko dve vrsti strategij:

- prva doseže zgornji pogoj s primerno nastavitvijo (fiksnih) parametrov (metoda poskusa in napake); domnevamo, da bo ta metoda uspešna samo za specifična nihala (masa, dolžina, trenje);
- druga strategija je "adaptivna", torej prilagaja parametre glede na učinek in se lahko poljubno približa idealnemu rezultatu ($\alpha_y=0$, $\omega_y=0$) pri različnih modelih nihala;
- "idealni rezultat" morda sploh ni idealen – ne smemo pozabiti, da ima nihalo dve prostostni stopnji in bo v "idealnem" primeru neizbežno zdrsnilo v stran. Praktična strategija mora torej vsebovati primerno velikost radialne hitrosti pri bližanju navpični točki, sistem pa ne bo prenesel preveč zaporednih približevanj točki povratka, ne da bi prešel v neželeno kroženje. V tem primeru je razlika med ugibanjem in adaptivno strategijo manj očitna.

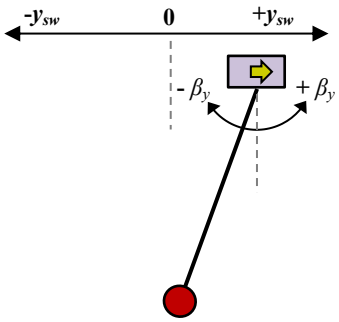
Prva strategija, ki se je izkazala za uspešno po le nekaj poskusih, ima naslednji algoritem:

1. robot se premakne v izhodiščno lego (A1 = +45°, A2 = -90°, Z = -50 mm);



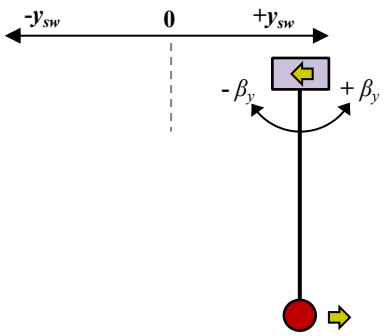
Slika 4.2: Izhodiščna lega

2. program čaka, dokler se nihalo ne umiri (kotne hitrosti padejo pod določen minimum);
3. robot se, z dovolj veliko hitrostjo (v_{sw}), premakne v pozitivni smeri y do nastavljenega položaja $+y_{sw}$ (*switch*). Nihalo zaradi vztrajnosti sledi premiku z zaostankom, kot β_y postane med premikom negativen;



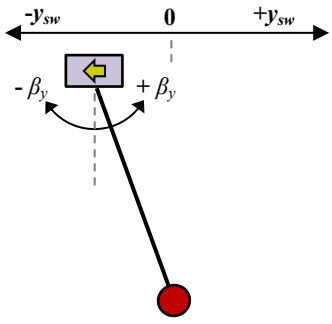
Slika 4.3: Premik do $+y_{sw}$

4. ko se roka ustavi v končnem položaju, se nihalo vrne proti ravnovesni legi s pozitivno kotno hitrostjo. V trenutku, ko postane β_y pozitiven, se sproži premik roke v negativni smeri y do nastavljenega položaja $-y_{sw}$;



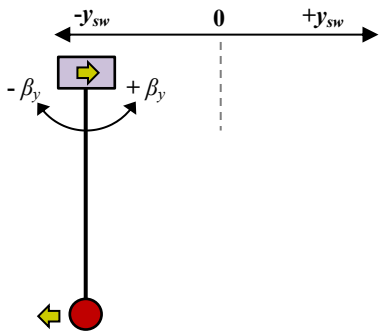
Slika 4.4: Preklop smeri gibanja v točki $+y_{sw}$

5. nihalo zaradi vztrajnosti sledi premiku z zaostankom, kot β_y postane med premikom pozitiven;



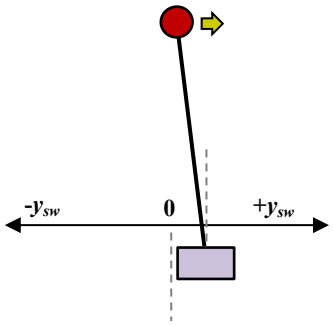
Slika 4.5: Premik do $-y_{sw}$

6. ko se roka ustavi v končnem položaju, se nihalo vrne proti ravnovesni legi z negativno kotno hitrostjo. V trenutku, ko postane β_y negativen, se sproži premik roke v pozitivni smeri y do nastavljenega položaja $+y_{sw}$;



Slika 4.6: Preklop smeri gibanja v točki $-y_{sw}$

7. skok na točko 4. - ta zanka se ponavlja, dokler ni presežena ustrezna absolutna vrednost kota β_y. V tem trenutku se program za zanihanje prekine (gib, ki je v teku, se zaustavi), gibanje roke pa prevzame regulator (ki na tej stopnji še ni realiziran).



Slika 4.7: Dosežena višina nihala, v kateri se vklopi regulator

Parametri, ki jih je potrebno določiti, in delujoče vrednosti, najdene s poskušanjem, so:

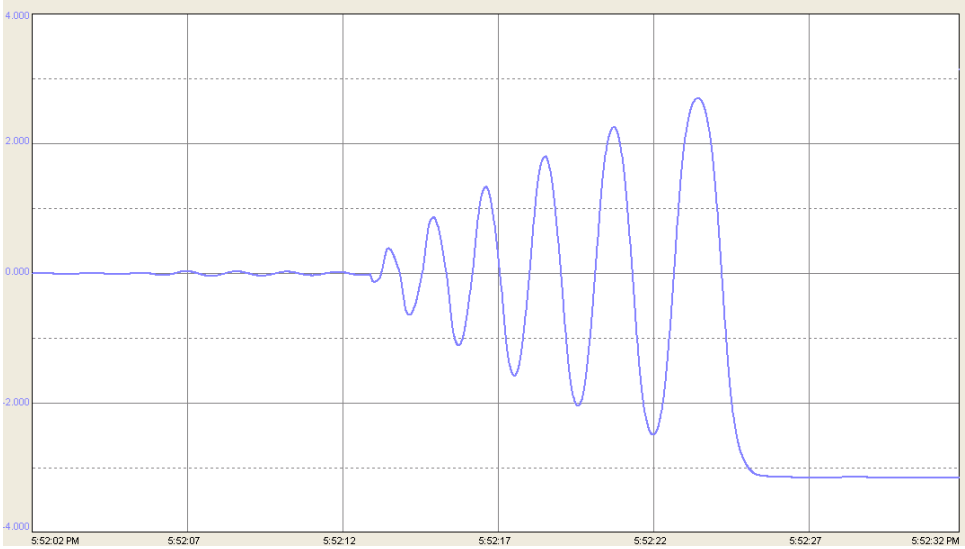
$+y_{sw}$, $-y_{sw}$ … položaj preklopa, velja $+y_{sw} = -(-y_{sw})$ (±125 mm);

v_{sw} … hitrost vzbujanja, premika med točkama $+y_{sw}$ in $-y_{sw}$ (1.300 mm/s);

a_{sw} … pospešek in pojemek vzbujanja (50.000 mm/s^2 ≈ 5g).

Če se spremenijo lastnosti nihala (m, l_r, f), je potrebno najti nove vrednosti parametrov.

Rezultat vidimo na sliki 4.8, prikazan je dosežen kot α_y, ki je v spodnji legi enak 0 in se v zgornji navpični legi približa $-\pi$ (v navpični legi nastopi nezveznost v točki $\pm\pi$, ki pa je za namene izrisa programsko omejena na bodisi $-\pi$ ali $+\pi$, odvisno od tega, s katere strani se je nihalo zadnjič približalo točki vklopa regulatorja). Celotno trajanje procesa zanihanja je 15 sekund.



Slika 4.8: Časovni diagram zanihanja od mirovne lege do vklopa regulatorja

V program vgrajen **evaluator** izklopi postopek zanihanja in vklopi regulator, ko se dosežen kot (višina) nihala in kotna hitrost hkrati pojavita v tem rangu vrednosti:

$\alpha_{sw} = \pm 0{,}125$ rad,

$\omega_{sw} = \pm 0{,}8$ rad/s.

Oba parametra sta ugotovljena s poskusi – je pa izbira vrednosti, ki še funkcionirajo, relativno široka. Pri preklopu prevzame nadzor gibanja robota regulator, pri čemer gibanje gladko preide iz enega načina v drugo – več o tem v poglavju 5.2. Generator položaja.

Adaptivna strategija je zelo podobna. Razlika je v tem, da se postopek začne s hitrostjo v_{swini}, ki je namenoma premajhna za doseganje zgornje lege. Hitrost se nato inkrementalno povečuje, dokler niso doseženi pogoji zgoraj omenjenega evaluatorja. Dodajanje energije je na ta način nadzorovano, se pa podaljša čas, potreben za uspešno izvedbo postopka.

Dodatni parametri, ki jih je potrebno določiti, in njihove uporabljene vrednosti so:

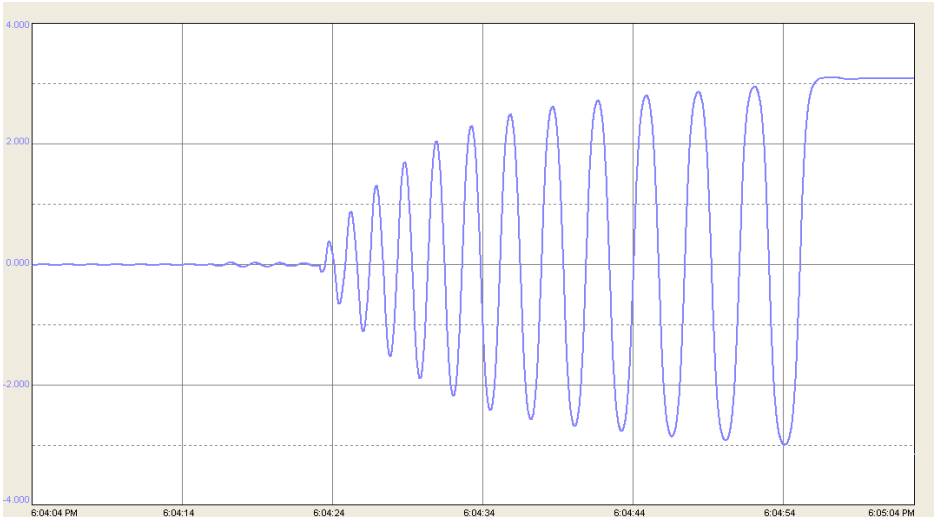
v_{swini} … začetna hitrost vzbujanja v_{sw} (1000 mm/s);

v_{swinc} … povečanje hitrosti vzbujanja v_{sw} (10 mm/s);

T_{sw} … čas, v katerem se v_{sw} poveča za v_{swinc}, če ni dosežena navpična lega (5 s);

v_{swmax} … maksimalna hitrost vzbujanja v_{sw}, pri kateri se povečevanje ustavi (1500 mm/s).

Rezultat je prikazan na sliki 4.9. Celotno trajanje zanihanja je 36 sekund.



Slika 4.9: Časovni diagram zanihanja od mirovne lege do vklopa regulatorja

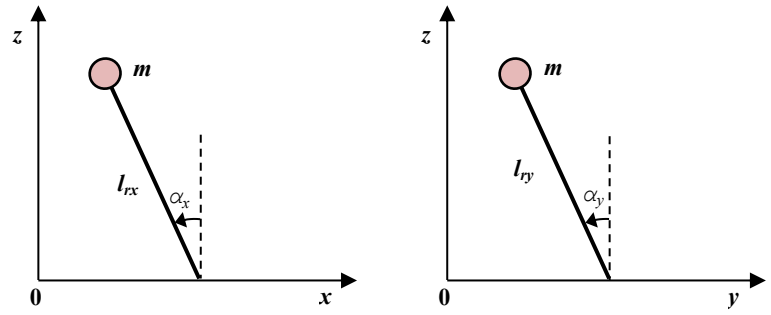
V nekaterih primerih (nastavitvah) lahko nihalo prekorači navpično lego in naredi cel obrat, vendar program nadaljuje s povečevanjem hitrosti zanihanja proti maksimalni vrednosti. To stanje je mogoče zaznati, zmanjšati hitrost na neko prejšnjo vrednost, jo spet povečevati z nekoliko manjšimi inkrementi itd. Vendar postopek v osnovi deluje bolje, kot smo že omenili, z bolj grobim približkom in primerno kotno hitrostjo pri bližanju navpični legi – sicer nihalo zdrsne v smeri kota α_x, poskus pa je potrebno prekiniti. Zato smo se z doseženim delovanjem obeh postopkov na tej stopnji zadovoljili.

V primeru, da se nihalo ne umiri popolnoma pred začetkom zanihanja, bo skoraj vedno začelo krožiti v ravnini *x-y* (premiki roke so sinhronizirani z lastno frekvenco nihala in ojačajo tudi kroženje) in poskus ne bo uspešen. Lahko pa si zamislimo naprednejši postopek, ki duši (namesto ojačuje) lateralno nihanje z ustreznim gibanjem v ravnini *x-z* in preprečuje kroženje.

5 Regulator

5.1 Regulator stanj

Invertirano nihalo je klasična platforma za preizkušanje različnih regulatorjev. V literaturi so opisani uspešni primeri vodenja s PID, regulatorjem stanj (*State Space Controller*), mehko logiko (*Fuzzy Logic*), adaptivnimi nevronskimi mrežami (*ANN*) in drugimi principi.
Najpogostejši regulator je regulator stanj, načrtan po metodi LQR. Ta tip regulatorja je bil prvi, ki smo ga uporabili na razvitem modelu.
Izhodišče za načrtovanje regulatorja je predpostavka, da sferično nihalo v okolici navpične (in ravnovesne) lege ($\alpha_x,\ \alpha_y \rightarrow 0$) lahko aproksimiramo z dvema neskloplјenima (neodvisnima) nihaloma – projekcijama na ravnini *x-z* in *y-z*.



Slika 5.1: Projekcija sferičnega nihala v ravnini *x-z* in *y-z*

To lahko pokažemo takole:
Kinetično energijo lahko razdelimo v vsoto komponent v ravnini *x-z* in *y-z*:

$$T = T_{xz}(x, \dot{x}, \alpha_x, \dot{\alpha_x}, l_{rx}) + T_{yz}(y, \dot{y}, \alpha_y, \dot{\alpha_y}, l_{ry}), \tag{5.1}$$

kjer sta projekciji dolžine nihala l_{rx} in l_{ry} enaki:

$$l_{rx} = l_r \cos \alpha_y, \tag{5.2}$$

$$l_{ry} = l_r \cos \alpha_x. \tag{5.3}$$

Potencialne energije ne moremo razdeliti na posamezne projekcije, vendar velja približek v bližini navpične lege:

$$\cos\alpha_x \approx \cos\alpha_y \approx 1 \rightarrow l_{rx} \approx l_{ry} \approx l_r. \tag{5.4}$$

Zato je potencialna energija projekcij približno enaka:

$$V(\theta, l_r) \approx V_x(\alpha_x, l_{rx}) \approx V_y(\alpha_y, l_{ry}). \tag{5.5}$$

Načrtovanje regulatorja torej lahko opravimo na osnovi modela za nihalo z eno prostostno stopnjo, kot je to v primeru klasičnega poskusa z vozičkom. Za razliko od izpeljave z Lagrangeovo funkcijo sistema (3.5), bomo tokrat upoštevali izgube nihala v obliki viskoznega trenja in enačbe modela izpeljali z izrekom, da je vsota vseh momentov (inercije, zunanjih momentov in trenja) enaka 0 (2. Newtonov izrek), (5.6).

α_x, α_y ... kot projekcije nihala (slika 5.1);
g ... gravitacijski pospešek (9,8 m/s^2);
x, y, z ... kartezični koordinatni sistem;
f ... konstanta viskoznega trenja (0,02 Nm/s);
l_r ... dolžina nihala (0,6 m);
m ... masa nihala (0,3 kg).

$$M_J + M_{zun} + M_f = 0, \tag{5.6}$$

$$M_J = J \cdot \ddot{\alpha_y}, \tag{5.7}$$

$$M_{zun} = F_{zun} \cdot l_r = (F_z + F_y) \cdot l_r = (-m \cdot g \cdot \sin\alpha_y - m \cdot \ddot{y} \cdot \cos\alpha_y) \cdot l_r, \tag{5.8}$$

$$M_f = f \cdot \dot{\alpha_y}. \tag{5.9}$$

Ker je:

$$J = m \cdot l_r^2, \tag{5.10}$$

dobimo

$$ml_r^2\ddot{\alpha_y} - ml_r g \sin\alpha_y - ml_r\ddot{y}\cos\alpha_y + f \cdot \dot{\alpha_y} = 0. \tag{5.11}$$

Izraz preoblikujemo v obliko, primerno za numerično integracijo po Eulerjevi metodi (izpostavimo najvišji odvod neodvisne spremenljivke):

$$\ddot{\alpha_y} = \left(g \cdot sin\,\alpha_y + \ddot{y} \cdot cos\,\alpha_y - \frac{f \cdot \dot{\alpha_y}}{m \cdot l_r}\right) \cdot \frac{1}{l_r}. \tag{5.12}$$

Podobno velja za projekcijo na ravnino *x-z*:

$$\ddot{\alpha_x} = \left(g \cdot sin\,\alpha_x + \ddot{x} \cdot cos\,\alpha_x - \frac{f \cdot \dot{\alpha_x}}{m \cdot l_r}\right) \cdot \frac{1}{l_r}. \tag{5.13}$$

Vidimo, da je rezultat enak izrazu, dobljenem za sferično nihalo v poglavju 3., če predpostavimo, da je v (5.12) $f=0$, $y''=0$ ter v (3.22) in (3.28) $\varphi=const$, $\theta=\alpha_y$.

V tem modelu ne nastopa masa »vozička« *M*. Namesto sile *F* na voziček nastopa kot vhodna veličina oz. vzbujanje pospešek sistema (pivota nihala) s komponentama a_x (oz. x'') in a_y (oz. y'').

Model lineariziramo za majhne premike v okolici navpične lege, pri čemer predpostavimo, da je:

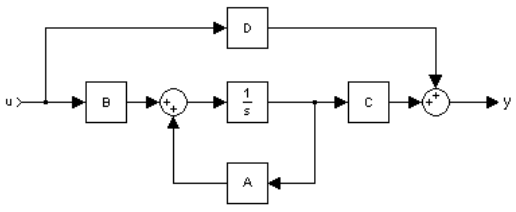
$$sin(\alpha) \approx \alpha,\ cos(\alpha) \approx 1,\ \dot{\alpha} \approx 0, \tag{5.14}$$

$$\ddot{\alpha_y} = \left(g \cdot \alpha_y + \ddot{y}\right) \cdot \frac{1}{l_r}, \tag{5.15}$$

$$\ddot{\alpha_x} = (g \cdot \alpha_x + \ddot{x}) \cdot \frac{1}{l_r}. \tag{5.16}$$

Model pretvorimo v obliko prostor stanj, ki ga določajo izrazi:

$$\dot{X} = A \cdot X + B \cdot u, \quad Y = C \cdot X + D \cdot u, \qquad X = \begin{bmatrix} y \\ \dot{y} \\ \alpha_y \\ \dot{\alpha_y} \end{bmatrix}, \qquad u = \ddot{y}. \tag{5.17}$$



Slika 5.2: Shema odprtozančnega sistema v obliki prostora stanj

Dobimo matrike sistema A, B, C in D:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & {}^{g}/_{l_r} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \\ 0 \\ {}^{1}/_{l_r} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = 0. \tag{5.18}$$

Prenosna funkcija je enaka:

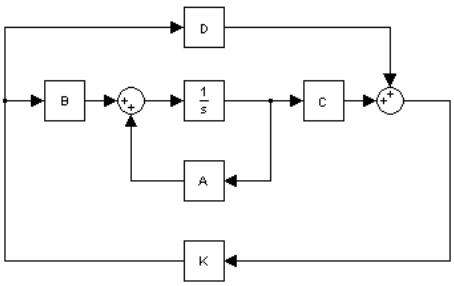
$$H(s) = \frac{Y(s)}{X(s)} = \frac{{}^{1}/_{l_r}}{{}^{g}/_{l_r} - s^2}. \tag{5.19}$$

Ker sta g in l_r vedno pozitivna, bo eden od polov odprtozančnega sistema na desni strani kompleksne ravnine faznega diagrama:

$$s_1 = +\sqrt{{}^{g}/_{l_r}}\,;\; s_2 = -\sqrt{{}^{g}/_{l_r}}\,. \tag{5.20}$$

Odprtozančni sistem je torej inherentno nestabilen, brez ustreznega zunanjega vzbujanja bo nihalo padlo iz navpične v spodnjo ravnovesno lego.

Regulator stanj dobimo, če sistem zaključimo z (negativno) povratno zanko preko ojačanja *K*:



Slika 5.3: Shema zaprtozančnega sistema v obliki prostora stanj

Vhod sistema *u* je sedaj povezan z vektorjem stanj *X* preko matrike ojačanj *K*:

$$u = -K \cdot X, \tag{5.21}$$

$$K = [k_1 \quad k_2 \quad k_3 \quad k_3]. \tag{5.22}$$

Z izbiro zaprtozančnih polov glede na želen odziv sistema lahko matriko *K* dobimo z rešitvijo enačbe:

$$0 = det\left(sI - (A - bK^T)\right). \tag{5.23}$$

Na tej točki smo za reševanje uporabili funkcije paketa *Matlab*. Do sedaj uporabljena orodja (RSLogix 5000 in RSTestStand) nimajo vgrajenih funkcij, ki bi lahko pomagale pri načrtovanju regulatorja stanj ali sploh pri vektorski in matrični aritmetiki (z izjemo kinematike, ki pa je vgrajena na ravni operacijskega in ne uporabniškega sistem Logix platforme).

Rešitev izraza (5.23) lahko dobimo s funkcijo *place* v Matlabu:

```
Pc = [-5 -4 -1+1i -1-1i]; % želeni poli zaprtozančnega sistema;
K = place (A, B, Pc); % matrika ojačanj K.
```

Dobljen rezultat je k1 = -2,45, k2 = -3,55, k3 = -35,27 in k4 = -8,73.

Vendar smo se odločili za metodo načrtovanja LQR (*Linear Quadratic Regulator*) s funkcijo *lqr* v Matlabu, ki poišče optimalni regulator oz. vrednosti *K*, s tem da minimizira kvadratično kriterijsko funkcijo *J(u)* in s tem velikost signala vhoda *u* in izhoda *X*:

$$J(u) = \int_0^\infty [X^T\, QX + u^T Ru]dt. \tag{5.24}$$

Matriki *R* in *Q* lahko izberemo z *Bryson-ovim* pravilom, upoštevajoč največjo želeno (ali možno) vrednost *X* in *u*:

$$Q_i = \frac{1}{\max\,(X_i^2)}, \tag{5.25}$$

$$R = \frac{1}{\max\,(u^2)}. \tag{5.26}$$

Vrednosti vstavimo v Matlab takole:

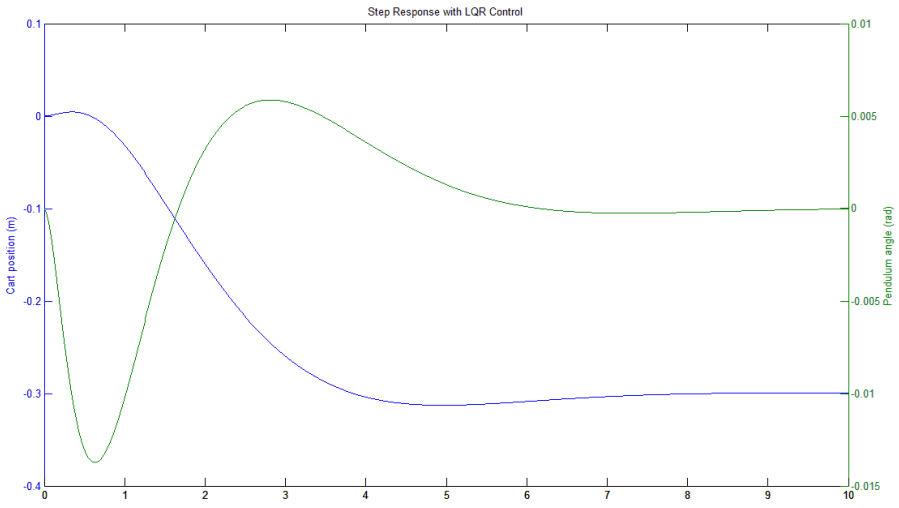
```
Q = zeros(4,4); % matrika Q;
Q(1,1) = 5; % max x = 0.4m;
Q(2,2) = 0.1; % max vx = 3m/s;
Q(3,3) = 5; % max a = 0.4rad;
Q(4,4) = 0.1; % max omega = 3rad/s;
R = 5; % matrika R, max u = 0.4m/s;
K = lqr(A,B,Q,R) % matrika ojačanj K.
```

Dobljen rezultat je k1 = -1,00, k2 = -1,91, k3 = -27,70 in k4 = -6,85.

Odziv zaprtozančnega sistema [5.18] na stopničasto vzbujanje (0,3 m) lahko simuliramo z ukazi:

```
Ac = [(A-B*K)];
states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};
sys_cl =
ss(Ac,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);
t = 0:0.01:10;
r =0.3*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','Cart position (m)');
set(get(AX(2),'Ylabel'),'String','Pendulum angle (rad)');
title('Step Response with LQR Control').
```

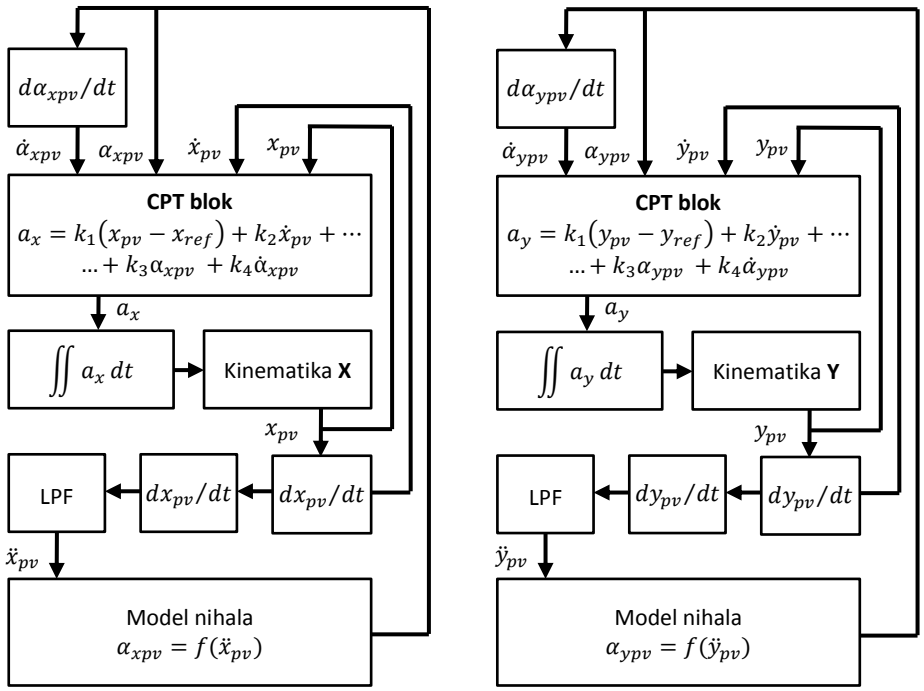
Dobimo izris rezultatov simulacije.



Slika 5.4: Simulacija odziva regulatorja v Matlab-u

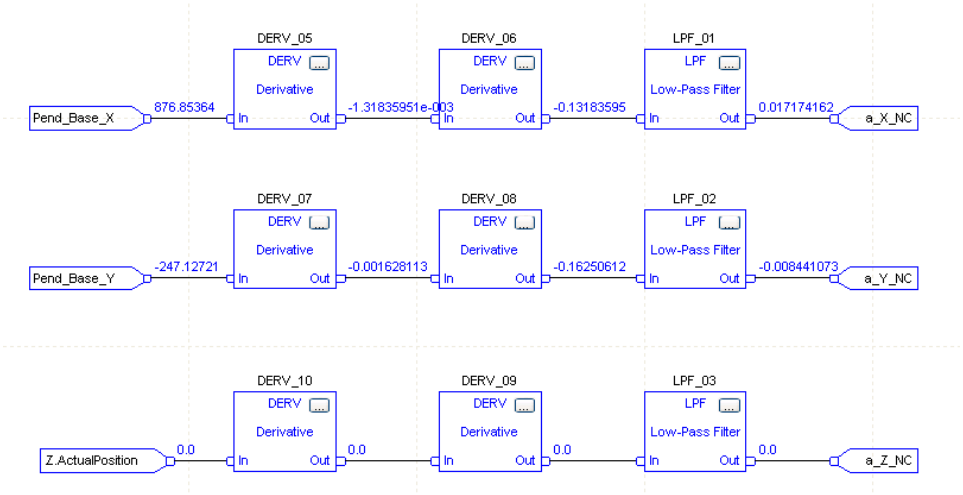
Dobljen umiritveni čas je 6 s, prenihaj pa 0,005 rad in 0,02 m.

Dobljeni rezultat je bil prenešen v PLC regulator kot izraz v CPT (Compute) bloku, kot kaže slika 5.5.



Slika 5.5: Regulator stanj za *x* in *y* kanal

Potrebno pa je bilo s poskušanjem nastaviti parametre nizkopasovnih filtrov. Uporabljen je v PLC vgrajen programski blok LPF (Low Pass Filter).



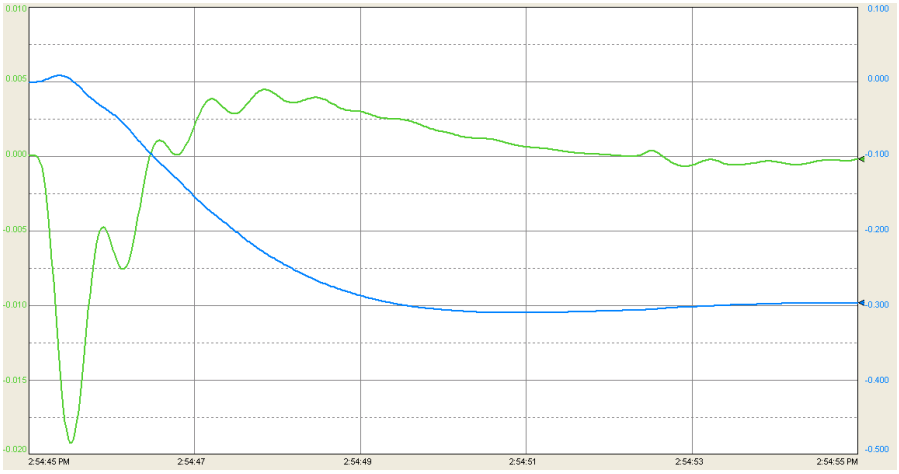
Slika 5.6: Dvojno odvajanje (DERV) in nizkopasovni filter (LPF)

Izbran je bil filter prvega reda s prenosno funkcijo:

$$H(s) = \frac{\omega}{s+\omega}. \tag{5.27}$$

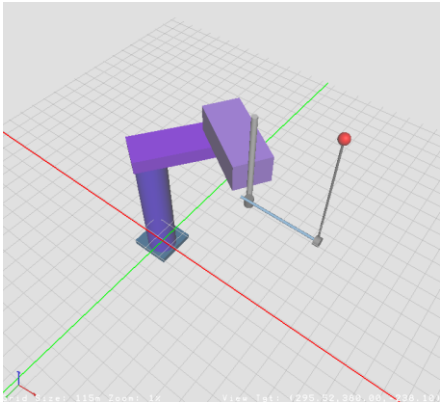
Parameter ω je bil nastavljen na 30 rad/s.

V fazi preizkušanja in razvoja regulatorja smo model sferičnega nihala (opisan v poglavju 3), nadomestili z modelom z eno prostostno stopnjo, α_y. Šele ko je regulator uspešno deloval z eno prostostno stopnjo, smo ga razširili na dve prostostni stopnji oz. na sferično nihalo. Slika 5.7 kaže odziv na referenčni signal 0,3 m (kot α_y zeleno, položaj y modro, za definicijo parametra motnje v % glej poglavje 5.4 Generator motnje).



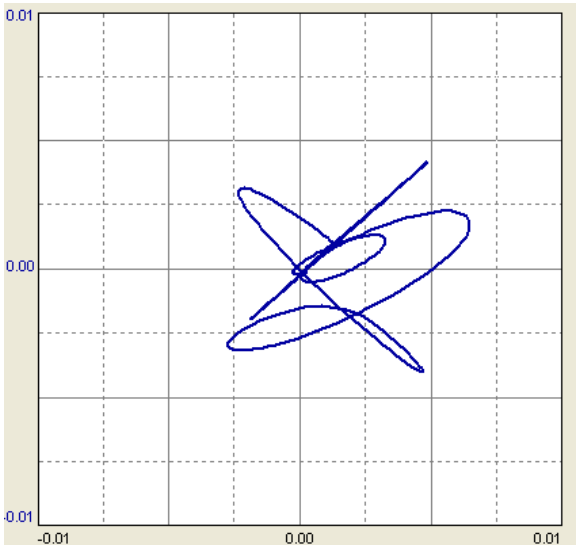
Slika 5.7: Odziv LQR regulatorja, območje -0,02 … 0,01 rad (kot, zeleno), -0,5 … 0,1 m (položaj, modro), čas 10 s, motnja 0 %

Odziv lahko primerjamo s simulacijo v Matlabu (slika 5.4), dobljen umiritveni čas je 7 s, prenihaj pa 0,005 rad in 0,01 m – torej dokaj podoben rezultat.

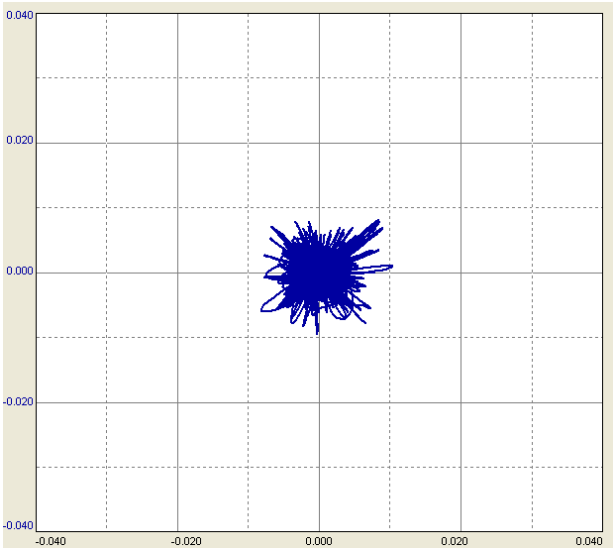


Slika 5.8: Regulator v delovanju

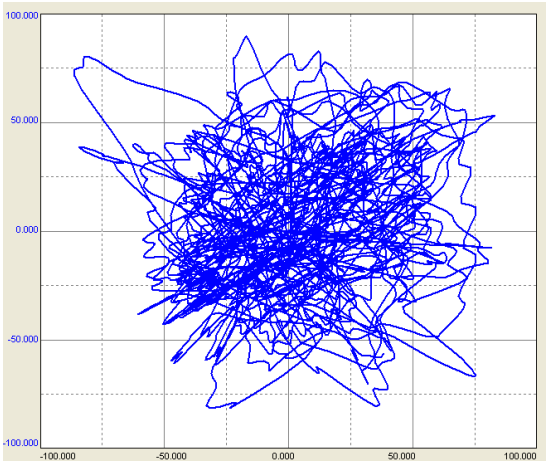
Spodnje slike prikazujejo delovanje obeh kanalov regulatorja kot trajektorijo *x*-*y* ravnini.



Slika 5.9: Trajektorija relativne lege konice nihala glede na pivot v ravnini *x*-*y*, območje ±10 mm, čas 5 s, motnja 100%



Slika 5.10: Trajektorija relativne lege konice nihala glede na pivot v ravnini *x*-*y*, območje ±40 mm, čas 10 min, motnja 100%



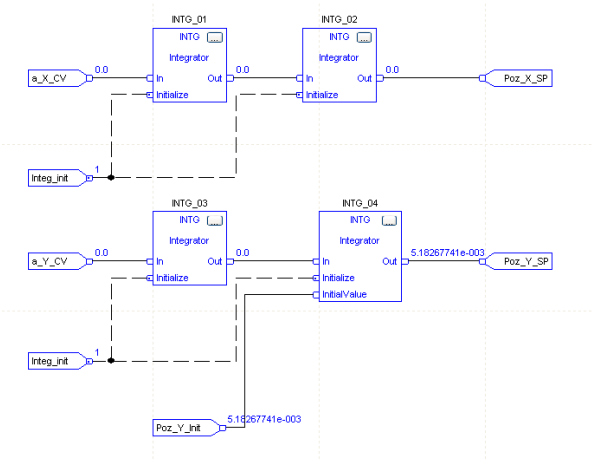
Slika 5.11: Trajektorija lege pivota nihala v ravnini *x*-*y*, območje ±100 mm, čas 10 min, motnja 100%

5.2 Generator položaja

Kot je razvidno iz bločne sheme regulatorja (slika 5.5) je izhodna (regulirana) veličina pospešek a_x in a_y, kar pa ni neposredno uporaben signal za regulacijo gibanja robota.

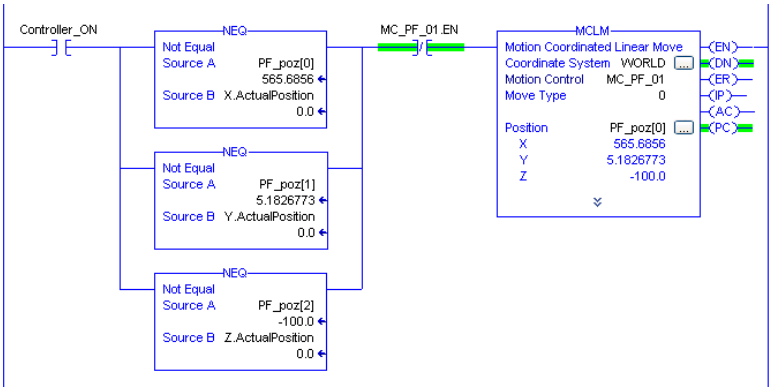
Servopogone *Kinetix* je sicer možno regulirati v načinu navora (t.i. *Torque Mode*, kjer je referenčna veličina tok motorja), vendar s tem odpade pozicioniranje, vodenje v koordinatnem sistemu in inverzna kinematika.

Rešitev je generator položaja – na bločni shemi 5.5 je vključen v blok Kinematika **X** in **Y**. Najprej torej potrebujemo referenčni (želeni) položaj x_{sp} in y_{sp}, ki mu bomo sledili. Tega dobimo tako, da izhod regulatorja a_x in a_y dvakrat integriramo (blok **INTG**).



Slika 5.12: Integracija izhoda regulatorja z bloki INTG

Kot je opisano v poglavju 2 (slika 2.26), je osnovni način gibanja robota linearni premik od točke do točke z ukazom MCLM (Motion Coordinated Linear Move). Programska vrstica na sliki 5.13 generira z MCLM nov linearni premik vsakič, ko se dejanski položaj (X.ActualPosition, Y.ActualPosition ali Z.ActualPosition) razlikuje od želenega (PF_poz[0], PF_poz[1] oz. PF_poz[2]). Največja hitrost proženja MCLM je enaka periodi izvajanja programa, to je 10 ms. Posamezni premiki so združeni brez zaustavljanja (izbran je parameter *Merge* v MCLM bloku); tako dobimo gladko gibanje, ki sledi zahtevam regulatorja.



Slika 5.13: Generator položaja

5.3 Referenčni signal

Želeni položaj x_{ref} in y_{ref} je možno spreminjati – lahko z direktnim vnosom v programskem orodju ali preko HMI vmesnika. V tem primeru delujejo tipke (**X FWD**, **X REV**, **Y FWD**, **Y REV**) kot ukaz za povečevanje ali zmanjševanje referenčnega položaja (do dinamične meje robota, x_{lim} = ±0,2 m in y_{lim} = ±0,35 m). Tipke za direktno vodenje osi A1 in A2 so blokirane, za Z in U pa delujejo enako kot pred začetkom poskusa.

Spreminjanje kota prijemala U deluje kot motnja na sistem in pri večjih hitrostih vrtenja (**JOG SPEED**) regulator pade iz ravnotežja. Tu naj ponovimo, da model nihala ne upošteva vrtenja okoli lastne osi (to je Z).

5.4 Generator motnje

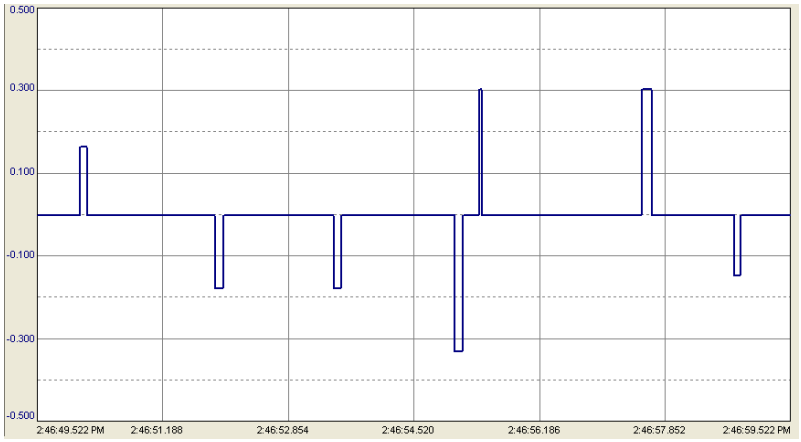
Generator motnje oz. šuma temelji na naključnem generatorju, ki je del RSTestStand aplikacije. Ta generira realna števila v rangu med -100,000 in +100,000 s periodo 1 s. S

pomočjo kombinacije naključnih števil je sestavljen stopničast signal z naslednjimi lastnostmi:

- dolžina signala = 10·(120+Rnd) [ms];
- dolžina pavze = 10+0,5·(100+Rnd) [ms];
- amplituda = HMI_Noise·(Rnd/30000) [m/s^2];

kjer je:
Rnd … trenutna vrednost naključnega generatorja RSTestStand (-100 do +100);
HMI_Noise … vrednost drsnika **NOISE** (0 do 100%) na HMI vmesniku (slika 2.36).

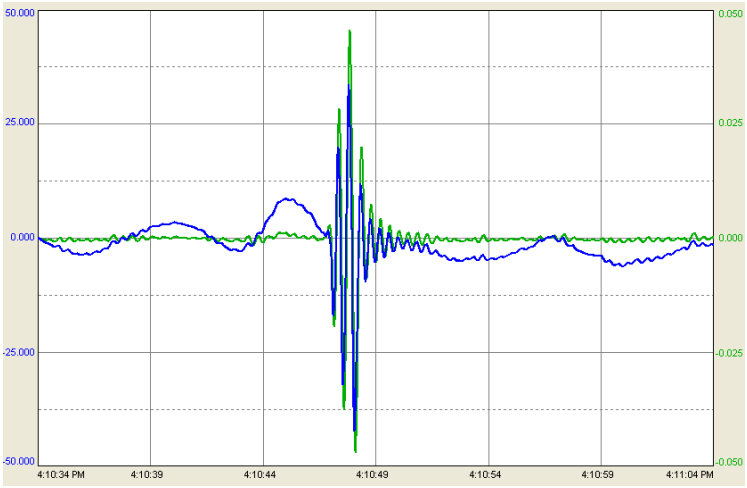
Ločeno je generiran signal za *x* in *y* komponento in prištet k vhodu v model nihala (slika 2.31), fizikalno je pomen signala torej zunanji pospešek (in s tem sila) na nihalo.



Slika 5.14: Signal generatorja šuma, nastavljena 100 % vrednost, območje ±0,5 m/s^2, časovni razpon 10 s

5.5 Ročno vnašanje motnje

S priključenim servoregulatorjem in motorjem v vlogi osi A1, A2 ali U je možno »ročno« generirati motnjo. V primeru, da motor ni premočan (uporabljen je bil 150 W / 1,25 Nm model), lahko os premaknemo kar z roko, kar vpliva na sistem kot motnja, ki jo mora regulator izravnati.



Slika 5.15: Vpliv »ročno« vnešene motnje na kot α_y preko fizičnega pogona v vlogi osi U, območje ±0.05 rad (kot, zeleno), ±50 mm (položaj, modro), čas 30 s

Drugi način generiranja motnje je možen z ročnim krmiljenjem osi U (tipke **U FWD** in **U REV** na HMI vmesniku), ki je neodvisna od regulatorja.

5.6 PID regulator

PID regulator bo po pričakovanju manj stabilen od regulatorja stanj, saj ima kot vhodni veličini samo dve spremenljivki, x in α_x (oz. y in α_y).

Podobno kot prej sta potrebna dva ločena regulatorja za komponenti nihala x-z in y-z. Vsak regulator vsebuje CPT (Compute) in PID blok, ki sta vezana kaskadno. Uporabljen je v PLC vgrajen programski PID blok.

Regulator je bil uglašen s poskušanjem, podobno kot v prejšnjem primeru najprej na nihalu z eno prostostno stopnjo. Izhodišče za uglaševanje je bila predpostavka, da je vrednost K_p podobna vrednosti k_1 regulatorja stanj (≈ 25).

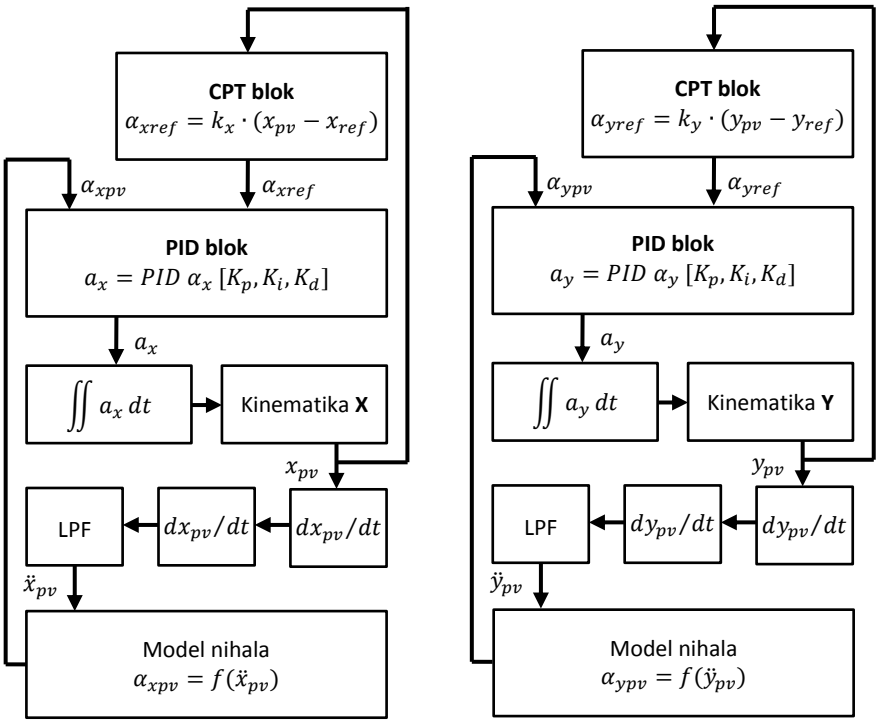
Najdene so bile naslednje vrednosti parametrov:

$k_x = k_y = 0{,}2$ [rad/m],

$K_p = 16{,}5$,

$K_i = 0{,}1$ [1/s],

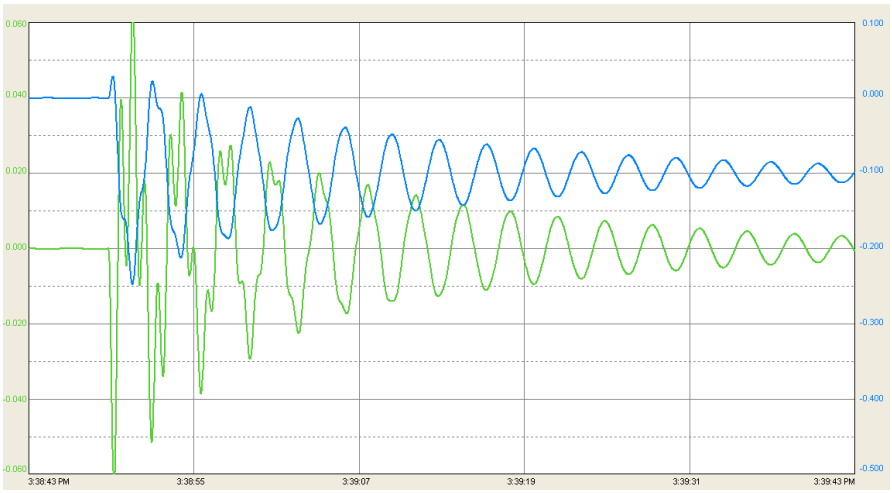
$K_d = 1{,}0$ [s].



Slika 5.16: PID-regulator za *x* in *y* kanal

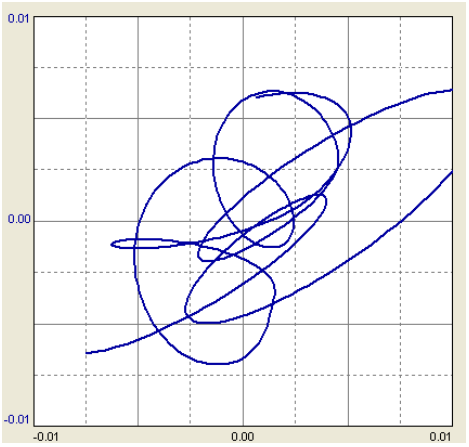
Na Logix platformi sicer obstaja možnost avtomatskega uglaševanja PID regulatorjev, ki pa temelji na odzivu odprtozančnega sistema na stopničasto vzbujanje in ni uporabna za naš primer (odprtozančno ni stabilen).

Odziv regulatorja na spremembo reference 0,1 m prikazuje slika 5.17 (Pri spremembi > 0,25 m regulator odpove in nihalo pade).

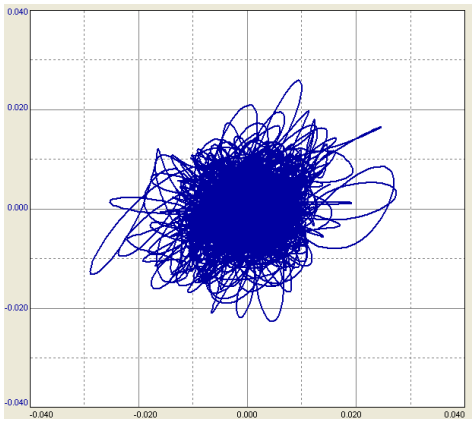


Slika 5.17: Odziv PID-regulatorja, območje -0,06 … 0.06 rad (kot, zeleno), -0,5 … 0,1 m (položaj, modro), čas 60 s, motnja 0 %

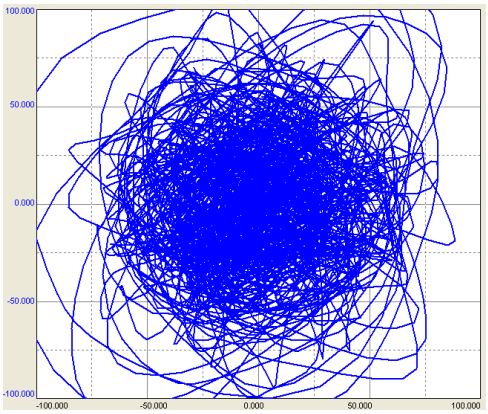
Odziv lahko primerjamo s prejšnjim regulatorjem (slika 5.7), dobljen umiritveni čas je daljši od 90 s, prenihaj pa 0,06 rad in 0,15 m. Sistem niha z lastno frekvenco in je na meji stabilnosti. Spodnje slike prikazujejo delovanje obeh kanalov regulatorja kot trajektorijo v *x*-*y* ravnini.



Slika 5.18: Trajektorija relativne lege konice nihala glede na pivot v ravnini x-y, območje ±10 mm, čas 5 s, motnja 100 %



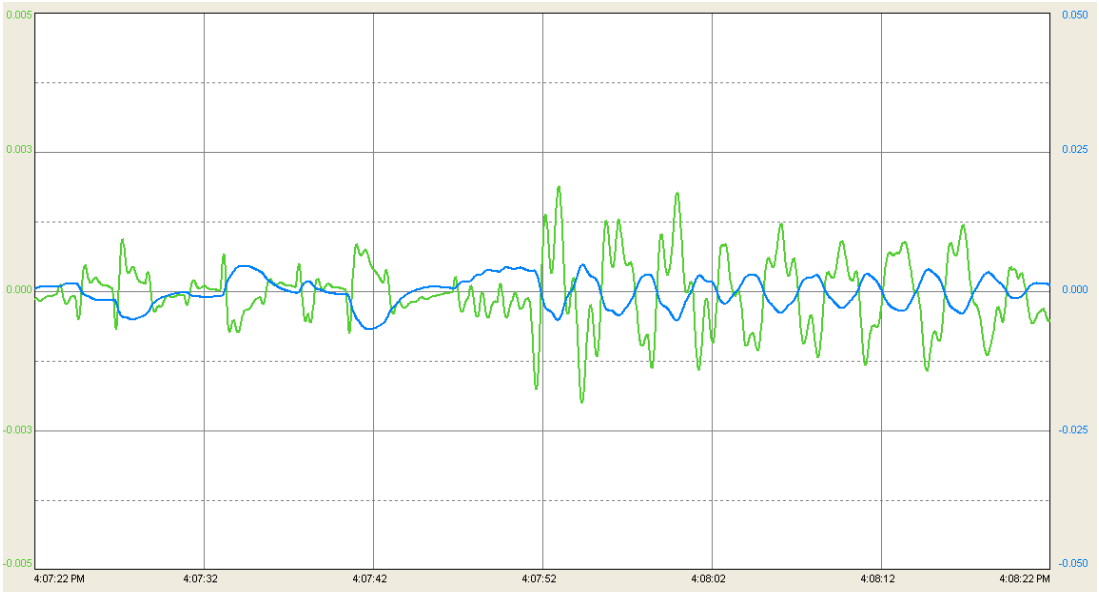
Slika 5.19: Trajektorija relativne lege konice nihala glede na pivot v ravnini *x*-*y*, območje ±40 mm, čas 10 min, motnja 100 %



Slika 5.20: Trajektorija lege pivota nihala v ravnini *x*-*y*, območje ±100 mm, čas 10 min, motnja 100 %

Vidimo, da ima PID regulator dejansko slabši odziv kot regulator stanj. Vprašljivo je, ali bi sploh deloval na realni napravi. V literaturi smo pravzaprav našli samo primere, izvedene v obliki računalniške simulacije (Matlab/Simulink, Java ipd.).

Med delovanjem poskusa je možno preklapljati med obema regulatorjema – slika 5.21.



Slika 5.21: Preklop med regulatorjem stanj in PID, območje ±0,005 rad (kot, zeleno), ±0,05 m (položaj, modro), čas 60 s, motnja 100 %, točka preklopa je na sredini izrisa

6 Zaključki

Glede na zastavljene cilje naloge lahko podamo te zaključke:

1. Izbrani gradniki in principi so bili primerni za realizacijo naloge.
2. Pri načrtovanju regulatorja stanj (izračun vektorja *K*) smo si pomagali s paketom Matlab. Končna izvedba naprave je uporabljala samo izbrano (programsko in strojno) opremo, ki je predvidena za industrijsko avtomatizacijo.
3. Uspešno sta bila realizirana dva tipa regulatorjev, LQR in PID. Rezultate je seveda možno izboljšati, vendar so bili cilji naloge multidisciplinarni, tako da smo se z doseženim delovanjem zadovoljili.
4. Simulacijo naprave smo še pred zaključkom te naloge demonstrirali strankam v podjetju. S tem smo na konkretnem primeru lahko pokazali eno ali več naslednjih točk:
   - obstaja prednost združevanja PLC in mehatroničnih funkcij na eni platformi (kot je Logix);
   - pred investicijo v kompleten (mehanski) sistem se lahko preizkusi izvedba različnih kompleksnih mehatroničnih sistemov in krmilnih algoritmov s pomočjo simulacije, modeliranja in 3D-vizualizacije;
   - na voljo je primerno podkovana tehnična podpora za strankin projekt;
   - v primeru, da je stranka vajena opreme drugega proizvajalca industrijske avtomatizacije (Siemens, Omron, Beckhoff …), je testna naprava primerna podlaga za primerjavo lastnosti sistemov in uskladitev tehničnih izrazov.

Prednost demonstracije je tudi to, da je nevtralna z aplikativnega stališča; ne sodi v nobeno posebno področje uporabe mehatronike (kot je pakiranje, strega in montaža, tisk in papir, avtomobilska industrija, polprevodniki, CNC obdelovanje itd.).

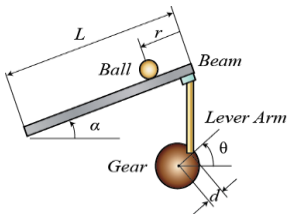
7 Nadaljnji poskusi

Realizirano platformo je mogoče prirediti za nadaljnje poskuse, povezane z interakcijo robota, dinamičnega sistema in uporabnika:

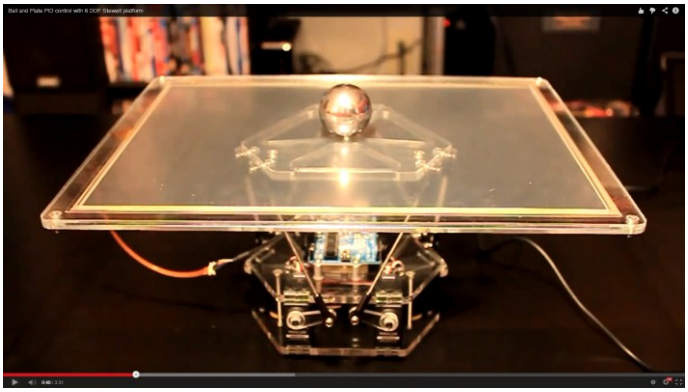
- regulator na osnovi pravil mehke logike – *Fuzzy logic;*
- regulator na osnovi umetnih nevronskih mrež – *Artificial Neuron Network (ANN);*
- *Furuta* nihalo – to je nihalo z eno prostostno stopnjo, ki pa ga vzbujamo z vrtenjem pivota okoli ekscentrične osi, namesto z linearnim gibanjem;

Slika 7.1: *Furuta* tip nihala na pilotni napravi, puščici prikazujeta dve osi vrtenja

- naprava s kroglico na klančini (*ball and beam system*) – je še ena klasična platforma za preizkušanje različnih krmilnih strategij. S pomočjo robota lahko izvedemo poskus z dvema prostostnima stopnjama (2 DOF), torej s kroglico na površini;



Slika 7.2: Naprava s kroglico na klančini, 1 DOF



Slika 7.3: Naprava s kroglico na klančini, 2 DOF. Mehanizem za nagib ploščadi je pravzaprav robot tipa Delta (slika 2.12).

– posoda s tekočino – je neke vrste nihalo, ki bi ga radi obdržali v ravnovesju, sicer se tekočina prelije čez rob. Aplikacija je torej lahko robot za strežbo, ki prenaša pladenj s pijačo;

Slika 7.4: Posoda s tekočino

– uporaba kamere za zaznavanje položaja nihala – z dvema sistemoma za strojni vid v ravnini *x-z* in *y-z* bi lahko zaznavali kot nihala, kar bi precej poenostavilo montažo nihala na prijemalo robota. Kot primer je spodaj prikazana kamera proizvajalca *Cognex*, ki združuje optični del in CPU za obdelavo slike v eni napravi. Krmilnik lahko dobi informacijo o legi objekta – nihala po vodilu *Ethernet/IP*.

Slika 7.5: Dve kameri za strojni vid proizvajalca *Cognex*, montirane pod pravim kotom

Literatura

[1] Control Tutorials for MATLAB and Simulink (CTMS). Dosegljivo: http://ctms.engin.umich.edu/CTMS/index.php?aux=Home. [Dostopano: 12. 8. 2014].

[2] System Dynamics. Dosegljivo: http://www.profjrwhite.com/system_dynamics/sdyn/s0/outl509/outl509.html. [Dostopano: 12. 8. 2014].

[3] R. Yang, *Geometric Techniques for Control of a 2-DOF Spherical Inverted Pendulum*. The Hong Kong University of Science and Technology, Department of Electrical and Electronic Engineering, 2000.

[4] G. Schreiber, C. Ott in G. Hirzinger, *Interactive Redundant Robotics: Control of the Inverted Pendulum with Nullspace Motion*. Wessling: German Aerospace Center – DLR, Institute for Robotics and Mechatronics, b. l.

[5] B. Sprenger, L. Kucera in S. Mourad, *Balancing of an Inverted Pendulum with a SCARA Robot*. Zurich: Swiss Federal Institute of Technology Zurich (ETHZ), Institute of Robotics.

[6] K. J. Astrom in K. Furuta, "Swinging up a Pendulum by Energy Control", *Automatica,* vol. 36, 2000.

[7] Control System Design. Dosegljivo: http://csd.newcastle.edu.au/control/simulations/pendulum.html. [Dostopano: 12. 8. 2014].

[8] J. P. Hespanha, *Undergraduate Lecture Notes on LQG/LQR controller design*, April 1, 2007.

[9] Wikipedia, The Free Encyclopedia, *»Inverted_pendulum«*. Dosegljivo: http://en.wikipedia.org/wiki/Inverted_pendulum. [Dostopano: 12. 8. 2014].

[10] Wikipedia, The Free Encyclopedia, *»Euler–Lagrange equation«*. Dosegljivo: http://en.wikipedia.org/wiki/Euler_lagrange. [Dostopano: 12. 8. 2014].

[11] Pendulum Project From Sccswiki. Dosegljivo: http://www5.in.tum.de/wiki/index.php/PendulumProject . [Dostopano: 12. 8. 2014].

[12] Y. Beceriklia in K. Celik, Fuzzy control of inverted pendulum and concept of stability using Java application. Mathematical and Computer Modelling, 46, str. 24–37, 2007.

[13] Damped Spherical Pendulum – Wolfram Demonstrations Project Dosegljivo: http://demonstrations.wolfram.com/DampedSphericalPendulum. [Dostopano: 12. 8. 2014].

[14] A. Zaidan, B. M. Y. Nouri in B. *Alsayid, Swing Up a Pendulum by Energy Control, International Journal of Engineering and Technology,* vol. 2, no. 3, 2012.

[15] K. J. Åström in K. Furuta, *Swinging up a pendulum by energy control.* Paper presented at IFAC 13th World Congress, San Francisco, California, 1996.

[16] Googol Technology,"Inverted Pendulum Experimental Manual suitable for GLIP series," 2nd edition, 2006.

[17] M. E. M. Janthong. *Doktorska disertacija: Design and Implementation of Control Concepts for Image-Guided Object Movement.* Von der Fakultät für Maschinenbau der Gottfried Wilhelm Leibniz Universität Hannover zur Erlangung des akademischen Grades, 2006.

[18] B. Zupančič, *Simulacija dinamičnih sistemov.* Ljubljana: Fakulteta za elektrotehniko in računalništvo, 1995.

[19] B. Zupančič, *Zvezni regulacijski sistemi* (2. del, 2. popravljena in dopolnjena izdaja), Ljubljana: Fakulteta za elektrotehniko in računalništvo, 1995.

[20] I. Škrjanc, *Regulacije I*, Ljubljana: Fakulteta za elektrotehniko in računalništvo, 2006.

[21] I. Škrjanc, *Seminar: vodenje sistemov II. Praktikum* (1. izdaja). Ljubljana: Fakulteta za elektrotehniko in računalništvo, 2006.

[22] T. Bajd, *Osnove robotike* (7. popravljena in dopolnjena izdaja). Ljubljana: Fakulteta za elektrotehniko, 2006.

[23] B. Jurčič in B. Orel, *Numerične metode.* Ljubljana: Fakulteta za elektrotehniko in računalništvo, 2004.

Izjava o avtorstvu diplomskega dela

Izjavljam, da sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Igorja Škrjanca.

V Ljubljani, 25. september 2014 Žiga Petrič