Evolving principal component clustering with a low run-time complexity for LRF data mapping

Gregor Klančar[a,*], Igor Škrjanc[a]

[a]Laboratory of Modelling, Simulation and Control, Faculty of Electrical Engineering, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia

Abstract

In this paper a new approach called evolving principal component clustering is applied to a data stream. Regions of the data described by linear models are identified. The method recursively estimates the data variance and the linear model parameters for each cluster of data. It enables good performance, robust operation, low computational complexity and simple implementation on embedded computers. The proposed approach is demonstrated on real and simulated examples from laser-range-finder data measurements. The performance, complexity and robustness are validated through a comparison with the popular split-and-merge algorithm.

Keywords: line extraction, evolving clustering, laser range finder

1. Introduction

In mobile robotics, localization plays the key role in most applications. In most cases advanced sensor systems are required to estimate the robot pose, mostly due to the fact that there is no single and effective sensor that would directly measure the robot pose in an indoor environment. For an outdoor location the closest approximate is GPS, but this can fail in areas with no satellite signal reception. Therefore, various different sensors are fused together to improve the robustness and quality of the pose estimate ([1, 2, 3, 4]). In

*Corresponding author. Tel.: +386-1-4768764; fax:+386-1-4264631
Email address: `gregor.klancar@fe.uni-lj.si` (Gregor Klančar)

mobile robotics a very popular sensor for this purpose is the laser range finder (LRF), which has good coverage, dense information, high accuracy and a high sampling rate. It can be used for localization purposes, map building or SLAM, as in [5], [6], [7], [8], [9], [10]. Using a LRF the robot pose can be estimated by comparing a locally sensed map given by a cloud of reflection points and a known map of the environment. This comparison is usually made by comparing simple geometric features that are extracted from the LRF reflection points. The simplest features are straight lines.

To estimate the line feature parameters from 2D laser-range-finder data several line-extraction algorithms have been proposed. The process of line fitting generally requires two steps: first, the input points are investigated to find clusters of points that can be described by a line and, second, a line-fitting method is applied to estimate the straight-line parameters for the identified clusters. This process is usually done recursively. When the points are obtained from a 2D LRF the first step of the procedure can be simplified, because the clusters always consist of consecutive LRF points (here, it is called a sorted data stream). For the second step a least-squares method is normally used.

A very popular and powerful approach in image processing is the Hough transform, [11] where the data are transformed to a parameter space, and by locating the maxima the number and the parameters of the lines are obtained. Some drawbacks of the classic algorithm are parameter-space quantization as well as substantial computational and storage requirements [12]. To increase the accuracy and avoid the required predefined fine grid of the accumulator in the parameter space, several studies on the randomized Hough transform [13] were proposed that reduce the computational time and the storage requirements. The online adaptive implementations proposed in [14] reduce the space requirements and retain a high parameter precision. However, in general the position and the length of the line segments cannot be determined, and also collinear line segments cannot be separated. Some additional algorithm needs to be implemented to locate the line segments on the identified straight lines.

For the data obtained from a LRF used in indoor environments the split-

and-merge algorithm [15] is a very common choice made by robot developers. An extensive comparison of line-extraction algorithms is reported in [12], where the split-and-merge and incremental algorithms were preferred because of their simplicity, low computational complexity and good estimation results. Primarily because of their simplicity, incremental algorithms have been used in many applications. The data are incrementally added to the initial line cluster until the data fit the model; otherwise a new cluster is constructed.

A binary regression tree obtained by recursive fuzzy clustering and the identification of a hinging hyperplane that consists of two linear submodules is described in [16]. Similarly, a generalized fuzzy C-means clustering is proposed by [17]. A recursive clustering and fuzzy Takagi Sugeno identification is presented in [18], [19], [20]. Fuzzy approaches can also be applied to identify a low size feature subset which maximize information and minimize data redundancy as in [29]. These approaches can effectively model general nonlinear dynamics systems, but for linear data several more effective approaches can be applied, as stated in [12].

The robust fitting of models in the presence of outliers that can be obtained using the RANSAC (Random Sample Consensus) algorithm is introduced in [21]. A robust expectation-maximization estimate for a mixture of linear-regression models is proposed in [22] and robust clustering around the regression models is proposed in [23].

In most presented algorithms, fitting the model to the data involves least-squares methods, which can be hard to code on simple embedded hardware and also require some computational power because the data fitting is carried out many times in the recursive line-extraction algorithms.

Our idea is to fit the model using recursive principal component analysis (PCA), which is very easy to implement and computationally effective. A recursive PCA for adaptive process monitoring is used in [24]. An expectation-maximization approach for high-dimensional data model-based clustering using an incremental PCA is presented in [25]. In our study only a recursive covariance matrix needs to be evaluated. From the covariance matrix the model parameters that optimally fit the data in the sense of squared errors are defined by eigenvectors of the covariance matrix. The main advantage of the proposed approach is its low computational complexity and the very simple implementation of the algorithm, as it only requires basic arithmetic operations, such as additions and multiplications, together with high accuracy. This makes the proposed algorithm especially appropriate for SLAM problems where localization is done online.

The paper is organized as follows. After the introduction, the structure of the evolving principal component clustering (EPCC) algorithm is presented. Next the performance, computational complexity and robustness of the EPCC are evaluated and compared to the Split–and–Merge algorithm. The experimental and simulation results are described and at the end some conclusions are drawn.

2. Evolving principal component clustering

In batch clustering the usual problems are the initialization of the clusters and the in-advance guess of the correct number of clusters. The number of clusters can be determined by iteratively increasing the number of clusters and performing a clustering algorithm for each iteration until the terminating criteria are reached (e.g., the split-and-merge algorithm).

An alternative approach is recursive clustering, which is especially suitable for data streams. Here, the algorithm recursively estimates the simple statistical properties of the data (mean and variance) and clusters the data into clusters whose centers are defined with linear prototypes. The algorithm starts with one cluster and adds a new cluster when the current data sample does not belong to the existing prototypes. For the dimension of the data r the dimension of the prototype is $s = r - 1$ or less, so $s = 1$ for a straight line, $s = 2$ for a plane and $s > 2$ for a hyperplane.

2.1. Estimate of cluster mean and variance

The mean value and the variance for each cluster are calculated recursively. The mean value of the data in cluster j is defined by

$$\boldsymbol{\mu}_j(k_j) = \frac{k_j - 1}{k_j}\boldsymbol{\mu}_j(k_j - 1) + \frac{1}{k_j}\boldsymbol{z}(k_j) \tag{1}$$

where k_j is the current index of the data in cluster j and $\boldsymbol{z}(k_j)$ is the current data sample belonging to the cluster. The initial mean value of the cluster is $\boldsymbol{\mu}_j(k_j) = \boldsymbol{z}(k_j)$, where $k_j = 1$. The covariance matrix of the cluster j is defined as follows

$$\boldsymbol{\Sigma}_j(k_j) = \tfrac{k_j-2}{k_j-1}\boldsymbol{\Sigma}_j(k_j-1) + \tfrac{1}{k_j}\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j(k_j-1)\right)\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j(k_j-1)\right)^T \tag{2}$$

which is equivalent to the non-recursive form

$$\boldsymbol{\Sigma}_j = \frac{\sum_{k_j=1}^{n_j}\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j\right)\left(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j\right)^T}{n-1}$$

where n_j is the number of data in the cluster j.

The covariance matrix $\boldsymbol{\Sigma}_j$ contains the elements $\boldsymbol{\Sigma}_j = \left[\sigma_{io}^2\right],\ i = 1, ..., r,\ o = 1, ..., r$, which can be used to estimate the j-th linear prototype parameters, as shown in subsection 2.1.1.

2.1.1. Estimation of eigenvectors for low-dimensional data

The normal vector of the j-th linear prototype (hyperplane model of the cluster j) is the eigenvector $\boldsymbol{p}_j$ with the smallest eigenvalue. The eigenvectors can be obtained by a singular value decomposition of the covariance matrix $\boldsymbol{\Sigma}_j$, where an algorithm like [26] could be applied. However, for two- or three-dimensional data ($r = 2$, the prototype is a straight line or $r = 3$, the prototype is a plane) it is computationally much more efficient if the normal vector is obtained from $\boldsymbol{\Sigma}_j$. In the following the normal vector $\boldsymbol{p}_j$ to the linear prototype is estimated for two-dimensional data and three-dimensional data. The normal vector $\boldsymbol{p}_j$ belongs to the smallest eigenvalue, which equals

$$\lambda_j = \boldsymbol{p}_j^T\boldsymbol{\Sigma}_j\boldsymbol{p}_j \tag{3}$$

The orthonormal eigenvectors of the cluster j are obtained from the covariance matrix $\mathbf{\Sigma}_j$ using its elements. The normal vector in the case of $r = 2$ is

$$\boldsymbol{p}_j = \begin{bmatrix} -\frac{\theta_1}{\sqrt{\theta_1^2+1}} \\ \frac{1}{\sqrt{\theta_1^2+1}} \end{bmatrix}$$

where

$$\theta_1 = \frac{\sigma_{12}}{\sigma_{11}} \tag{4}$$

and where $\sigma_{io},\ i,o \in \{1,2\}$ are the elements of the covariance matrix.

In the case of three-dimensional data ($r = 3$) the normal vector is defined by

$$\boldsymbol{p}_j = \begin{bmatrix} -\frac{\theta_1}{\sqrt{\theta_1^2+\theta_2^2+1}} \\ -\frac{\theta_2}{\sqrt{\theta_1^2+\theta_2^2+1}} \\ \frac{1}{\sqrt{\theta_1^2+\theta_2^2+1}} \end{bmatrix}$$

where

$$\theta_1 = \frac{\sigma_{13}\sigma_{22} - \sigma_{23}\sigma_{12}}{\sigma_{11}\sigma_{22} - \sigma_{12}^2}$$

$$\theta_2 = \frac{\sigma_{13}\sigma_{12} - \sigma_{23}\sigma_{11}}{\sigma_{12}^2 - \sigma_{11}\sigma_{22}} \tag{5}$$

and where $\sigma_{io},\ i,o \in \{1,\cdots,3\}$ are the elements of the covariance matrix.

This approach can also be extended to higher dimensional data, but it becomes more computationally intense. Therefore, for higher dimensions ($r > 3$) the normal vector could instead be obtained by the singular value decomposition of $\mathbf{\Sigma}_j$.

2.2. Linear prototype estimation and clustering criteria

The normal vector $\boldsymbol{p}_j$ of the j-th prototype that models the data $\boldsymbol{z}(k_j)$ ($k_j = 1,\cdots,n_j$) in the cluster j defines the j-th prototype equation as follows

$$(\boldsymbol{z}(k_j) - \boldsymbol{\mu}_j)^T \cdot \boldsymbol{p}_j = 0$$

For the current datum sample $\boldsymbol{z}(k)$, which needs to be classified in one of the existing prototypes j ($j \in \{1, \cdots, m\}$), the orthogonal distance $d_j(k)$ (in the direction of the normal vector) from the j-th prototype is calculated as

$$d_j(k) = |(\boldsymbol{z}(k) - \boldsymbol{\mu}_j)^T \cdot \boldsymbol{p}_j| \tag{6}$$

If $d_j(k) = 0$ the data sample lies on the linear prototype j. This orthogonal distance $d_j(k)$ is used to determine wether the current data sample $\boldsymbol{z}(k)$ belongs to the $j-th$ cluster. To do this test robustly (in the presence of system noise and different data scaling) a criterion compares the orthogonal distance of the current sample to the j-th prototype with the variance of the orthogonal distance σ_j of all the data samples in the cluster. The clustering criterion is formulated as

$$d_j(k) < \kappa_{max}\sqrt{\sigma_j} \tag{7}$$

where κ_{max} is a positive constant defining the sensitivity of the clustering criteria relative to the cluster distance variance $\sqrt{\sigma_j}$. If normal LRF sensor noise distribution is supposed then selecting $\kappa_{max} = 3$ would imply 99.7 % off all samples to be properly classified or $\kappa_{max} = 4$ would result in theoretically 99.9 % of all samples belonging to the prototype being also correctly clustered. However by further increasing κ_{max} this percentage converges to 100 % but also the probability of wrong clustering increases.

If the current data $\boldsymbol{z}(k)$ fulfils criterion (7) the number of data in the j-th cluster is increased ($n_j \longrightarrow n_j + 1$) and the cluster distance variance (together with the cluster mean (1) and variance (2)) is updated recursively as follows

$$\sigma_j(k_j) = \sigma_j(k_j - 1)\frac{k_j - 2}{k_j - 1} + \frac{1}{k_j} d_j(k)^2 \tag{8}$$

An additional criterion, when adding the data sample $\boldsymbol{z}(k)$ to the j-th cluster, could be the distance to the cluster mean value $\boldsymbol{\mu}_j$ or the distances between consecutive points (in the case of a data stream from a laser range finder). This distance should be below the average distance between the consecutive data samples in the cluster. Dislocated points will then form a new cluster, although the criterion (7) is fulfilled.

The identified strait line segments are defined by the linear prototypes parameters (clusters centers and eigenvectors) and by the end points which are obtained from the two points in the cluster with the maximum distance among them. In sorted data stream this are the first and the last sample in the cluster.

2.3. Cluster initialization

The proposed clustering algorithm starts with one cluster and adds a new cluster if the current data sample $\boldsymbol{z}(k)$ does not satisfy the criterion (7) for the current clusters $j \in \{1, \cdots, m\}$.

To initialize a cluster a set of at least k_{min} (for $r = 2$, $k_{min} = 3$) data sample candidates that reliably form a new prototype are required. k_{min} is defined by the data dimensionality r where the linear model can be estimated from at least r data samples. To increase robustness to noise and outliers k_{min} should be higher so that the identification becomes over determined. First, the mean and variance of the new cluster candidate are calculated using (1) and (2). Then, the distance criterion (7) to the new prototype is validated. To reliably form a new cluster in the initialization phase, the κ_{max} parameter in (7) can be lowered (e.g., by 50%). Additionally, the distance between consecutive data samples can be validated. The cluster is initialized with the mean value, variance, linear prototype (the last eigenvector of the covariance matrix) and orthogonal distance variance (8).

2.4. Outlier removal

In the case of data streams where the data are arriving in a sorted fashion, as in a laser range finder where consecutive data samples belong to one prototype, the process of outlier detection is as follows.

- When the current data $\boldsymbol{z}(k)$ does not belong to the current cluster it is stored in a buffer. When k_{min} or more data samples are stored in the buffer, they are checked for consistency to form a new cluster initialization. If the initialization is successful then the content of the buffer is cleared. If the buffer contains only outliers or too much outliers then condition (7) for at least k_{min} samples in the buffer is not fulfilled and the initialization phase is unsuccessful and the buffer data are kept for future iterations. If the buffer is full (i.e., it contains more than $2k_{min}$ samples) then the oldest sample is removed. The number of full buffer n_{buf} must be higher than k_{min} to be able to eliminate outliers. If $n_{buf} = 2k_{min}$ then correct cluster initialization can be obtained if the buffer contains less than 50 % outliers.

- If less than k_{min} samples are buffered and the current sample $\boldsymbol{z}(k)$ is successfully clustered to one of the existing prototypes, then those buffered samples are considered as outliers and are removed from the buffer.

For general data streams where the data samples are arriving randomly, consecutive data samples do not necessarily belong to the same prototype. In this case the data samples that are not clustered in one of the existing prototypes are stored in a buffer. When the buffer contains more than k_{min} data it is checked to see whether a new prototype (new cluster initialization) can be identified. When a new cluster is initialized using k_{min} or more data from the buffer the remaining buffer data must remain in the buffer for future iterations. To limit the required memory space the available buffer size n_{buf} is defined and when it is exceed the oldest sample is removed.

2.5. Clustering algorithm

The proposed evolving clustering algorithm is illustrated in Fig. 1. The pseudo-code of the principal component clustering algorithm in on-line identification is given in Algorithm 1 and Algorithm 2.

For data streams with data samples arriving in a sorted fashion (Algorithm 1) the code is more compact and computationally efficient. These algorithms can be applied in situations where consecutive data samples belong to one prototype only or to the neighboring prototype. Data belonging to one particular prototype are therefore always a sequence. An example of such a data stream is a 2D laser range finder where the reflection points belong to consecutive laser

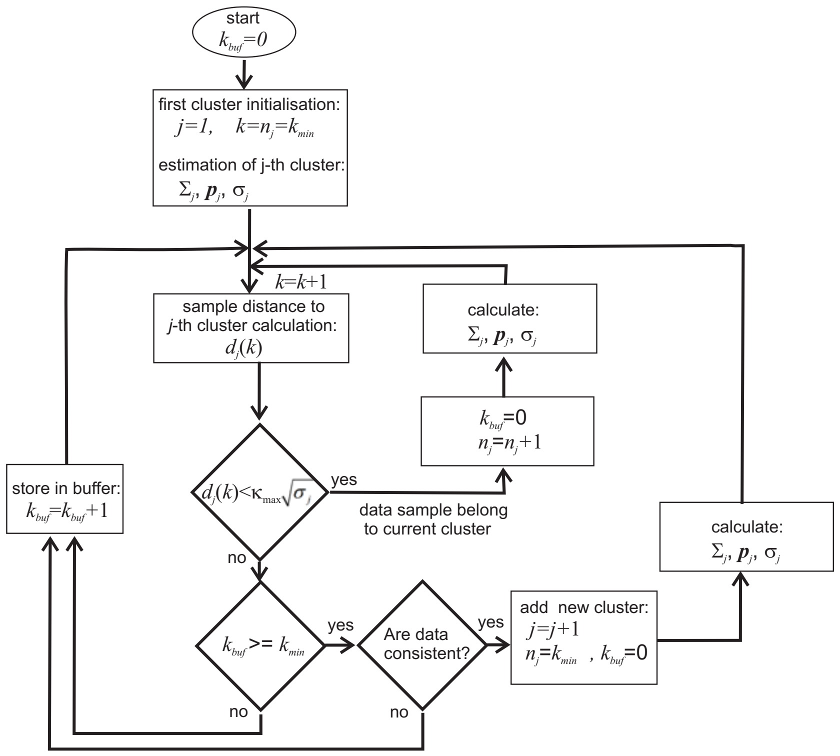

start
$k_{buf}=0$
first cluster initialisation:
$j=1, \quad k=n_j=k_{min}$
estimation of j-th cluster:
Σ_j, $\boldsymbol{p}_j$, σ_j
$k=k+1$
sample distance to
j-th cluster calculation:
$d_j(k)$
calculate:
Σ_j, $\boldsymbol{p}_j$, σ_j
$k_{buf}=0$
$n_j=n_j+1$
$d_j(k)<\kappa_{\max}\sqrt{\sigma_j}$
yes
data sample belong
to current cluster
no
store in buffer:
$k_{buf}=k_{buf}+1$
calculate:
Σ_j, $\boldsymbol{p}_j$, σ_j
$k_{buf}>=k_{min}$
yes
no
Are data
consistent?
yes
no
add new cluster:
$j=j+1$
$n_j=k_{min}$, $k_{buf}=0$

Figure 1: Evolving principal component clustering where the clusters are defined by linear prototypes.

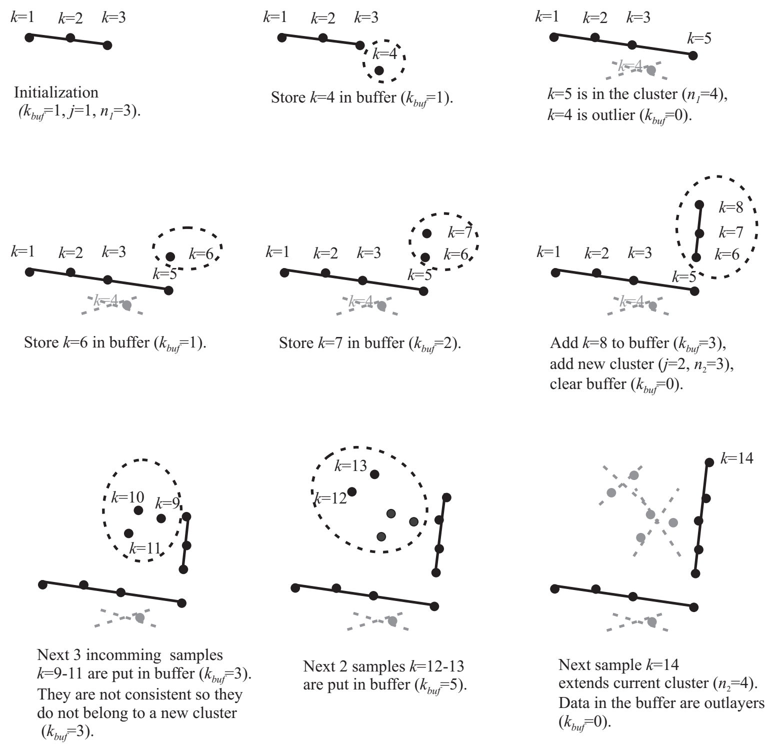

k=1 k=2 k=3
Initialization (kbuf=1, j=1, n1=3).
k=1 k=2 k=3 k=4
Store k=4 in buffer (kbuf=1).
k=1 k=2 k=3 k=5 k=4
k=5 is in the cluster (n1=4), k=4 is outlier (kbuf=0).
k=1 k=2 k=3 k=5 k=6 k=4
Store k=6 in buffer (kbuf=1).
k=1 k=2 k=3 k=5 k=7 k=6 k=4
Store k=7 in buffer (kbuf=2).
k=1 k=2 k=3 k=5 k=8 k=7 k=6 k=4
Add k=8 to buffer (kbuf=3), add new cluster (j=2, n2=3), clear buffer (kbuf=0).
k=10 k=9 k=11
Next 3 incomming samples k=9-11 are put in buffer (kbuf=3). They are not consistent so they do not belong to a new cluster (kbuf=3).
k=13 k=12
Next 2 samples k=12-13 are put in buffer (kbuf=5).
k=14
Next sample k=14 extends current cluster (n2=4). Data in the buffer are outlayers (kbuf=0).

Figure 2: Illustrative example of evolving principal component clustering algorithm operation. In a sorted data stream the two-dimensional data ($r = 2$, $k_{min} = 3$) are clustered on-line to straight-lines prototypes. The first $k_{min} = 3$ samples ($k = 1, \cdots, 3$) initialize the first cluster. Next samples ($k > 4$) extend the current cluster if they are sufficiently close to the straight line or they are buffered until k_{min} collinear samples form a new cluster initialization.

rays that are sent in the environment from a starting angle and they increment to a final angle.

In general data streams (Algorithm 2) the data samples are arriving randomly, so consecutive points do not necessarily belong to only one prototype. For sorted data stream Algorithm 2 has the same performance as Algorithm 1, but the computational complexity of Algorithm 2 is greater.

An illustrative example of how each step of Algorithm 1 works is presented in Fig. 2.

Algorithm 1 Pseudo-code of the principal component clustering algorithm in on-line identification for a sorted data stream.

```
1: Definition of the clustering criteria parameter $\kappa_{max}$, minimum number of samples
   $k_{min}$ to add a new prototype and the buffer size $n_{buf} = 2k_{min}$.
2: Initialization of the first prototype with the first $k_{min}$ data samples ($k = 1, \cdots, k_{min}$)
   consistent with the prototype. Initialization of $j = 1$, $m = 1$, $k_{buf} = 0$, $n_j = k_{min}$.
   Estimation of the $j$-th cluster covariance $\mathbf{\Sigma}_j$, prototype parameters $\boldsymbol{p}_j$
   and distance variance $\sigma_j$.
3: for $k = k_{min} + 1 : n$ do
4:     Calculate sample $\boldsymbol{z}(k)$ distance to the last prototype $j = m$ by
           $$d_j(k) = |(\boldsymbol{z}(k) - \boldsymbol{\mu}_j)^T \cdot \boldsymbol{p}_j|$$
5:     if $d_j(k) < \kappa_{max}\sqrt{\sigma_j}$ then
6:         Add data sample $\boldsymbol{z}(k)$ to the current cluster $j = m$ and increment $j - th$
           cluster counter $n_j = n_j + 1$.
7:         Update $\mathbf{\Sigma}_j$, $\boldsymbol{p}_j$ and $\sigma_j$ of the $j$-th cluster by Eq. (2), Eq. (3) and Eq. (8).
8:         Delete previous data samples in the buffer (outliers) and set $k_{buf} = 0$.
9:     else
10:        Store data sample $\boldsymbol{z}(k)$ in a buffer and increment the buffer counter $k_{buf} =$
           $k_{buf} + 1$.
11:        if $k_{buf} \geq k_{min}$ then
12:            if $n_{good} \geq k_{min}$data in the buffer are consistent then
13:                Add a new prototype ($m = m + 1$, $j = m$), set the prototype counter
                   $n_j = n_{good}$ and clear the buffer ($k_{buf} = 0$).
14:                Estimate $\mathbf{\Sigma}_j$, $\boldsymbol{p}_j$ and $\sigma_j$.
15:            else
16:                If the buffer is full ($k_{buf} \geq n_{buf}$) remove its oldest datum sample.
17:            end if
18:        end if
19:    end if
20: end for
```

Algorithm 2 Pseudo-code of the principal component clustering algorithm in on-line identification for a general data stream.

```
1: Definition of clustering criteria parameter $\kappa_{max}$, minimum number of samples $k_{min}$
   to add a new prototype.
2: Initialization of the first prototype: store incoming data samples ($k = 1, \cdots, n_I$)
   until $k_{min}$ data samples consistent with the prototype are found. Initialization
   of $j = 1$, $m = 1$, $k_{buf} = n_I - k_{min}$, $n_j = k_{min}$. Estimation of the $j$-th cluster
   covariance $\boldsymbol{\Sigma}_j$, prototype parameters $\boldsymbol{p}_j$ and distance variance $\sigma_j$.
3: for $k = n_I + 1 : n$ do
4:   for $j = 1 : m$ do
5:     Calculate sample $\boldsymbol{z}(k)$ distance to the prototype $j$ by
           $d_j(k) = |(\boldsymbol{z}(k) - \boldsymbol{\mu}_j))^T \cdot \boldsymbol{p}_j|$
6:   end for
7:   if $\min_j \{d_j(k)\} < \kappa_{max}\sqrt{\sigma_j)}$ then
8:     Add data sample $\boldsymbol{z}(k)$ to a cluster $j$ and increment $j - th$ cluster counter
       $n_j = n_j + 1$.
9:     Update $\boldsymbol{\Sigma}_j$, $\boldsymbol{p}_j$ and $\sigma_j$ for the $j$-th cluster by Eq. (2), Eq. (3) and Eq. (8).
10:  else
11:    Store data sample $\boldsymbol{z}(k)$ in the buffer and increment the buffer counter $k_{buf} =$
       $k_{buf} + 1$.
12:    if $n_{good} \geq k_{min}$ data in the buffer are consistent then
13:      Add a new prototype ($m = m + 1$, $j = m$), set the prototype counter
         $n_j = n_{good}$ and remove the data from the buffer ($k_{buf} = k_{buf} - n_{good}$).
14:      Estimate $\boldsymbol{\Sigma}_j$, $\boldsymbol{p}_j$ and $\sigma_j$.
15:    end if
16:  end if
17: end for
```

3. Comparison with the split-and–merge algorithm

A comparison of the proposed clustering algorithm with the very popular split-and-merge clustering algorithm is made. The comparison is made for sorted data streams obtained from a 2D laser range finder.

First, the split-and-merge algorithm is explained, then the algorithmic complexity is evaluated, followed by experimental results obtained on a series of 2D scans from a SICK LMS200 [27] laser range finder.

3.1. Split–and–Merge algorithm

Split-and-merge is a very popular algorithm for line extraction [12, 6]. Its popularity is due to its simplicity, low computational complexity and good performance. It is an iterative algorithm, applicable to sorted data streams, such as the ones from a laser range finder. The algorithm first assigns all the data samples to one cluster and calculates a linear prototype of the cluster (line for two-dimensional data). The cluster is then iteratively split at the data sample whose distance to the prototype is the largest and higher than the distance threshold d_{split}. The choice of the splitting constant d_{split} should consider the expected noise of LRF sensor measurements (in distance and angle). d_{split} must be set higher than expected measurement error due to the noise (e.g. three standard deviations or more).

The linear prototype of each cluster j ($j = 1, \cdots, m$) can be expressed in the normal form as

$$[\boldsymbol{z}(k)^T, 1]\boldsymbol{\theta}_j = 0 \tag{9}$$

where $\boldsymbol{z}(k)$ is a datum sample lying on the prototype and $\boldsymbol{\theta}$ is the vector of the prototype parameters. The prototype parameters are estimated using singular value decomposition. From all the data samples $\boldsymbol{z}(k_j)$ in the cluster j ($k_j = 1, \cdots, n_j$) the regression matrix is written as

$$\boldsymbol{\psi} = \begin{bmatrix} \boldsymbol{z}(1)^T & 1 \\ \vdots & \vdots \\ \boldsymbol{z}(n_j)^T & 1 \end{bmatrix}$$

which defines a set of homogenous equations $\boldsymbol{\psi}\boldsymbol{\theta}_j = \mathbf{0}$ with unknown prototype parameters $\boldsymbol{\theta}_j$. The solution in the sense of a least-squares minimization is found if the eigenvector ($\boldsymbol{p}_r$) of the matrix $\boldsymbol{\psi}^T\boldsymbol{\psi}$ with the minimum eigenvalue is found. This can be calculated using singular value decomposition. The prototype parameters in a normal form are obtained by normalization of the eigenvector

$$\boldsymbol{\theta}_j = \|\boldsymbol{p}_r\|$$

The orthogonal distance of an arbitrary data sample $\boldsymbol{z}(k)$ to the linear prototype j is then obtained by

$$d_j(k) = \left| \; [\boldsymbol{z}(k)^T, 1]\boldsymbol{\theta}_j \; \right| \tag{10}$$

In the case of two-dimensional data the linear prototype can alternatively be estimated simply by connecting the first and the last data sample in the cluster. This lowers the computational complexity and ensures that the sample that defines the split does not appear at the first or the last data sample in the cluster.

The pseudo code of the split-and-merge algorithm is given in Algorithm 3.

3.2. Comparison of algorithmic complexity

The Complexity of the split-and-merge algorithm (Algorithm 3) is $O(n \log n)$ iterations [12] as the algorithm has two nested loops where n is the number of data samples. During each iteration one calculation of the data-sample distance (10) to the cluster prototype is made to find the sample with the worst fit to the prototype that has a complexity of $O(2r-1)$ arithmetic operations (r multiplications and $r-1$ additions). Additionally, for each cluster at least once the prototype is estimated using the batch least-squares method, so altogether approximately $m \log m$ times, where m is the number of identified clusters. The complexity of the least-squares is $O(r^2 n)$ [28] due to the singular value decomposition of the regression matrix $\boldsymbol{\psi}$, where r is the dimension of the data sample and $n \gg r$. In total the split-and-merge algorithm needs

$$O_{SM}(n \log n(2r-1) + m \log m \cdot r^2 n) \tag{11}$$

arithmetic operations.

The complexity of the evolving principal component clustering (Algorithm 1) is $O(n)$ algorithm iterations. During each iteration the distance of the current

Algorithm 3 Pseudo-code of the split-and-merge algorithm line identification for a sorted data stream.

```
1: Definition of split criteria $d_{split}$.
2: Start with a single cluster which contains all the data ($k = 1, \cdots, n$). Initialize
   current cluster index $j = 1$, number of samples in that cluster $n_j = n$ and number
   of clusters $m = 1$. Mark the cluster as non-final.
3: repeat
4:    for all non-final clusters $j = 1, \cdots, m$ do
5:       Fit the linear prototype to the data in cluster $j$.
6:       for all data $k_j = 1, \cdots, n_j$ in cluster $j$ do
7:          calculate distance of data sample $\boldsymbol{z}(k_j)$ to the cluster prototype $\boldsymbol{\theta}_j$.
                $d_j(k_j) = \left| \, [\boldsymbol{z}(k_j)^T, 1]\boldsymbol{\theta}_j \, \right|$
8:       end for
9:       Locate the sample $\boldsymbol{z}_{max}$ with the maximum distance $d_{max} = \max_{k_j}(d_j(k_j))$
10:      if $d_{max} > d_{split}$ then
11:         Split the cluster at the $\boldsymbol{z}_{max}$ into two clusters and mark them non-final.
12:      else
13:         Mark the cluster as final.
14:      end if
15:   end for
16: until all clusters are final
17: Merge collinear clusters. This step is optional and is usually not required in ordered
    data streams.
```

datum sample to the actual prototype is calculated and the cluster variance, mean and distance variance are updated for the actual cluster. An update of the distance (6) requires $O(2r-1)$ arithmetic operations, an update of the variance (2) takes $O(5r)$ (3 multiplications and 2 additions of r dimensional data) arithmetic operations, an update of the mean (1) takes $O(3r)$ (2 summations and 1 division of r dimensional data) and an update of the distance variance (8) requires $O(4r)$ arithmetic operations. In total, the evolving principal component clustering algorithm needs

$$O_{EPCC}(14nr - n) \tag{12}$$

arithmetic operations.

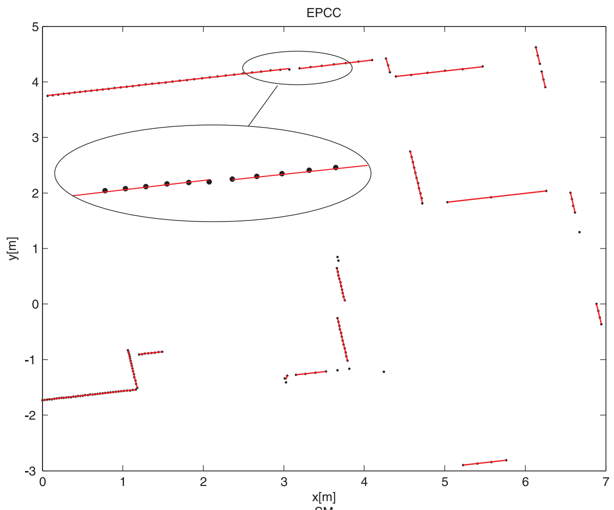
From a comparison of (11) and (12) it is evident that the complexity of the proposed algorithm is lower than the split-and-merge algorithm in the case of a large number of data n and decreases with the data dimension r and with the number of identified clusters m. The complexity of the proposed algorithm also does not depend on the number of identified clusters. The relative complexity O_{EPCC}/O_{SM} is shown in Table 1, where it can be seen that the split-and-merge algorithm is computationally more efficient only in the case of low-dimensional data and a small number of identified clusters (e.g., $r = 2$ and $m < 4$).

For general data streams Algorithm 2 needs to be employed. Its complexity is greater than the complexity of Algorithm 1, which is only valid for sorted data streams. The complexity of Algorithm 2 is $O(n \log n)$ and requires $O_{EPCCgen}\left(n \log n\,(14r - 1)\right)$ arithmetic operations. Note that the split-and-merge algorithm is a batch algorithm valid only for sorted data. This means that on-line identification is not possible. Moreover, its use for general data would require some modifications, which would increase its complexity like in the case of the proposed algorithm.

3.3. Experimental comparison

Both clustering algorithms were validated on data obtained from the SICK LMS200 laser range finder, where each scan contains $n = 180$ two-dimensional points.

The results of the clustering for both algorithms are shown in Fig. 3. Both algorithms produce clustering results of similar quality if the clustering parameters are properly set. In the split-and-merge (SM) algorithm the threshold distance of the cluster splitting was $d_{split} = 0.06$ m. The value of this parameter depends on the clustering data noise and scale, and therefore needs to be adjusted for a particular clustering problem. In evolving principal component



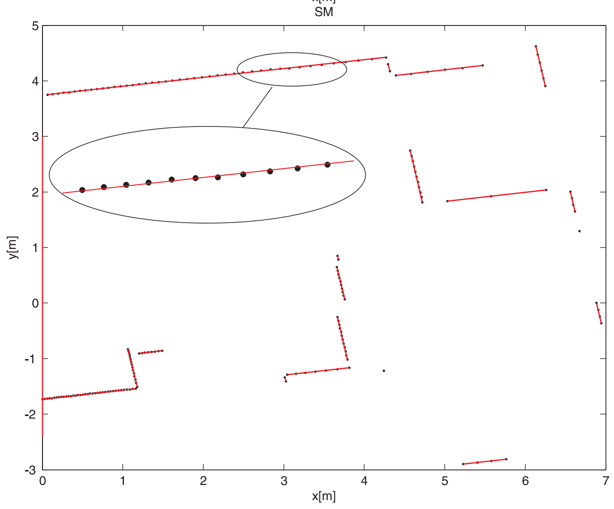


Figure 3: Clustering results of Evolving principal component clustering (EPCC) algorithm and Split–and–Merge Algorithm (SM). Results are obtained on data from SICK LMS200 laser range finder.

Table 1: Relative complexity comparison (O_{EPCC}/O_{SM}) of evolving principal component clustering (EPCC) algorithm and split–and–merge Algorithm (SM) where n is the number of r dimensional data and m is the number of clusters. For values less than 1 the EPCC is more efficient than the SM.

Relative complexity for $r = 2$

n \ m	2	4	6	8	10	12	14
10^2	3.21	1.72	1.09	0.77	0.58	0.46	0.38
10^3	2.36	1.44	0.97	0.71	0.55	0.44	0.36
10^4	1.87	1.24	0.88	0.66	0.51	0.42	0.35
10^5	1.55	1.09	0.80	0.61	0.49	0.40	0.34
10^6	1.32	0.97	0.73	0.57	0.46	0.38	0.32

Relative complexity for $r = 3$

n \ m	2	4	6	8	10	12	14
10^2	2.65	1.29	0.78	0.54	0.41	0.32	0.26
10^3	2.00	1.11	0.71	0.51	0.39	0.31	0.25
10^4	1.61	0.98	0.66	0.48	0.37	0.30	0.24
10^5	1.34	0.87	0.61	0.45	0.35	0.28	0.24
10^6	1.15	0.79	0.56	0.43	0.34	0.27	0.23

Relative complexity for $r = 4$

n \ m	2	4	6	8	10	12	14
10^2	2.32	1.04	0.62	0.42	0.31	0.24	0.20
10^3	1.79	0.92	0.57	0.40	0.30	0.24	0.19
10^4	1.46	0.82	0.53	0.38	0.29	0.23	0.19
10^5	1.23	0.74	0.50	0.36	0.28	0.22	0.18
10^6	1.06	0.68	0.47	0.34	0.27	0.22	0.18

clustering (EPCC) the choice of initialization parameters was $\kappa_{max} = 7$ and $k_{min} = 3$. Those two parameters are independent of the clustering problem. The parameter k_{min} only defines the minimum number of samples needed to form a new cluster, while κ_{max} is used to validate whether the current sample distance to the actual cluster prototype is less than the distance variance for all the samples in the cluster (see clustering criteria in line 5 of Algorithm 1). The clustering criteria therefore considers the cluster statistic and adapts automatically; there is no need for manual tuning for different data noise or scaling.

In Fig. 3 the EPCC finds 19 clusters and the SM 18 clusters, which is due to the different clustering criteria. This additional cluster (in the EPCC case) is zoomed in Fig. 3. It appears because the distance variance σ of the cluster data is very small and therefore the splitting condition (7) becomes more selective. If d_{split} were to be lowered or κ_{max} set higher then the same number of clusters could be obtained, but the clusters would not be identical due to the different clustering algorithm. Both algorithms were implemented in the same environment (Matlab) and on the same computer with a similar programming style. For the data in Fig. 3 the EPCC took 0.0098 s and the SM took 0.0262 s. The computation time for the EPCC is less than half of the computation time needed for the SM, which corresponds to the complexity comparison given in Table 1.

Besides the lower computational complexity, the EPCC's advantage over the SM is a simpler implementation on some embedded computers, because only simple mathematical operations are required, as opposed to the SM, where least-squares or singular value decomposition needs to be coded. When splitting clusters in SM it could happen that the sample with the worst fit (with the largest distance from the cluster prototype) appears at the beginning or at the end of the strait line segment where line splitting does not make sense. In this case the cluster prototype can be re-estimated by simply connecting those two edge points of the cluster which guarantee the splitting sample to be somewhere in the middle. So obtained prototype then does not fit the cluster data optimally in the least-squares sense, therefore it should only be used in the mentioned splitting problem.

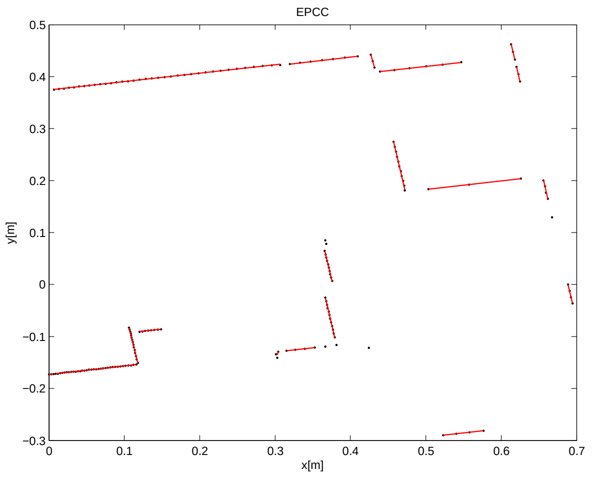
An additional advantage of the EPCC is the adaptive nature of the clustering parameters, which do not need to be fine tuned for each problem separately (different noise and data scaling). The latter advantage is also important when performing clustering on the same data, where data belonging to one cluster can have more noise than the data belonging to the other cluster (e.g., in the case that the laser range finder rays are close to being parallel to the object's borders). The SM is a batch clustering algorithm, so it requires all the data samples to perform clustering, while the EPCC can be used online for data coming sequentially from a process. The former can be an advantage when online clustering is required, or a disadvantage in the case that batch data are available because the clustering algorithm can perform better if it has the whole data set information available.

Nevertheless, the quality of clustering is very similar for both approaches.

3.4. Algorithm tuning effort and robustness to data scale and noise

Both algorithms have only one tuning parameter and are very simple to adjust to a particular clustering application. In the SM this parameter is d_{split} and in the EPCC it is κ_{max}. The clustering criterion (7) in the EPCC considers data variance, which causes the clustering to adapt to the current data. The former is especially convenient in the case of different data scaling or different noise in the data. The EPCC therefore needs less or no tuning compared to the SM, an example is given in Fig. 4. Data from Fig. 3 are scaled to 10 % of the original scale (divided by 10), while the algorithm's parameters remain the same. As seen from Fig. 4, the clustering result remains unchanged in the EPPC, while in the SM the clusters are wrong.

The example in Fig. 3 includes real laser-range-finder data with noise. In order to demonstrate the robustness to different noise influences, additional noise with a normal distribution is added to the real data. The obtained clustering results are given in Fig. 5 where, the EPCC identifies 16 clusters and the SM 36



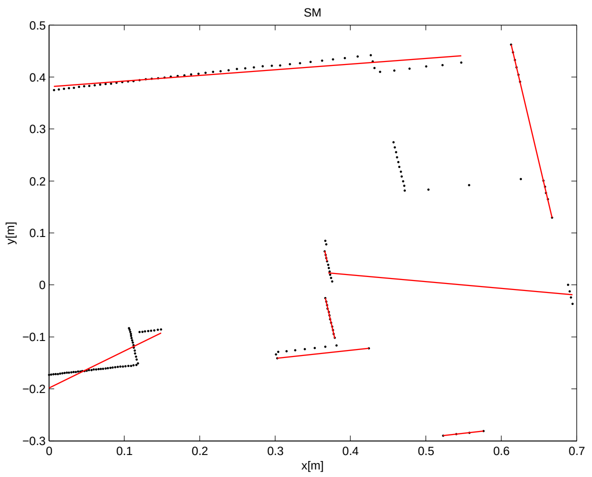


Figure 4: Change of data scaling to 10 % of scaling in Fig. 3 with no change of clustering algorithm parameters.

clusters. More noise causes the clustering criterion (7) to become less sensitive, so a smaller number of clusters are identified (three less than in Fig. 3). This behavior is desirable. However, in the SM the number of clusters is increased (twice as many as in Fig. 3) because the splitting criterion (line 10 in algorithm 3) is triggered by the noise and the obtained clusters are not reliable. To obtain more reliable results in the SM, the parameter d_{split} must be increased.

3.5. Robustness to outliers

The original laser-range-finder data from Fig. 3 are additionally corrupted by salt-and-pepper noise. To the original sorted data stream the uniform random noise sample (outlier) is included so the final data set has 360 samples (180 original samples and 180 outlier samples). The results of the clustering using EPCC are given in Fig. 6, where 16 clusters are identified. On the same data set the basic SM algorithm fails to produce useful clustering results due to the inserted outliers. Therefore, the Hough transform is applied, which can reliably estimate the clusters in the presence of outliers.

The basic Hough transform (HT) is implemented where the straight-line parameters α and d are defined by the linear prototype (9) where $\boldsymbol{\theta}_j = [\cos\alpha,\ \sin\alpha,\ d]$. The normal line parameters range $-\pi < \alpha \leq \pi$ and $d_{min} < d \leq d_{max}$ are presented in the accumulator by a 720 row and 720 column array. HT therefore requires some a-priori knowledge to properly select quantization of the parameter space and the threshold value to locate maximums in the accumulator. The obtained accumulator is shown in Fig. 7. The HT can locate 8 straight lines that describe 13 data clusters of the data, because the HT cannot separate collinear line segments, as seen from Fig. 7. Other straight lines appear randomly on collinear outlier samples. The HT is a robust algorithm for sorted or general batch data, or also for data streams if online implementations are used. In contrast, the EPCC algorithm is a well-suited algorithm for sorted stream or batch data and can therefore reliably eliminate outlier data that appear during a sequence of good data samples belonging to one cluster. Also, the distance of the consequent data samples can easily be considered in the clustering cri-

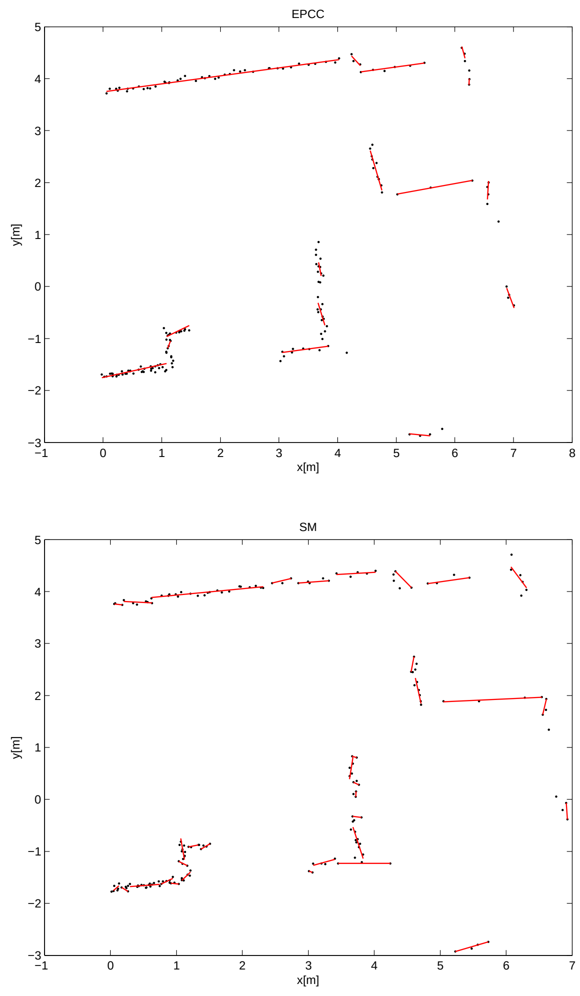


Figure 5: Clustering results on real data from Fig. 3 with enhanced noise and with no change of clustering algorithm parameters.

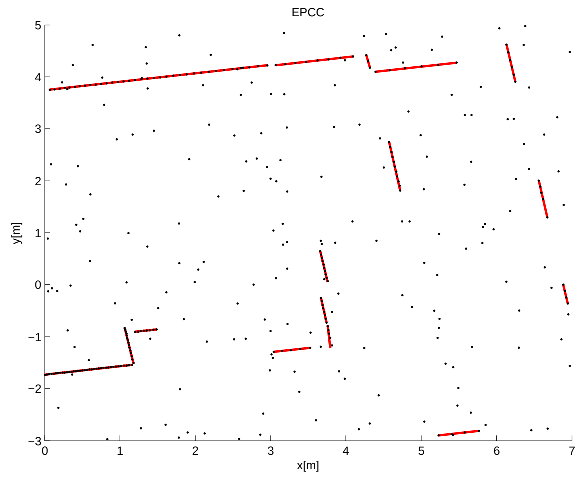


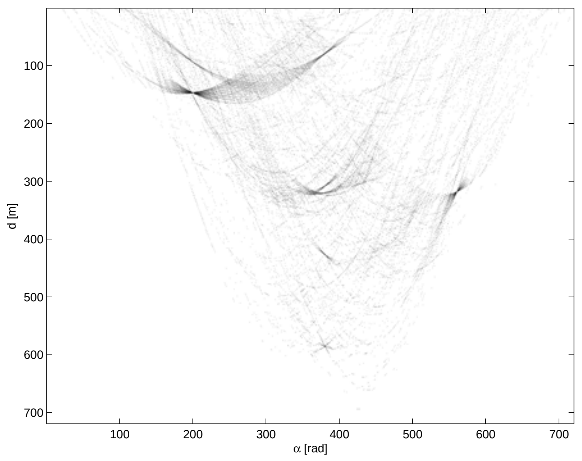
Figure 6: Clustering results of Evolving principal component clustering (EPCC) algorithm on real data with salt-and-pepper noise.

teria. If this distance is too long, then the current sample does not belong to the current prototype. Additional advantages of the EPCC are its simplicity of implementation, low parameter-tuning effort and fast operation. The EPCC is much faster than the classic HT implementation (in the example in Fig. 6 the EPCC took 0.03 s and the HT took 1.1 s). In the HT the result of clustering is dependent on the proper selection of the parameter-space quantization and on the parameter settings for the maximums search in the accumulator.

3.6. Statistical results of clustering

A detailed comparison of commonly used algorithms for straight-lines extraction from laser-range-finder data is given in [12]. To evaluate the proposed evolving principal component clustering (EPCC) algorithm a similar comparison is made by considering the split-and-merge algorithm (SM) and the Hough transform (HT).

A laboratory room consisting of 60 scans obtained from different locations using a mobile robot with a SICK LMS200 laser range finder is shown in Fig.



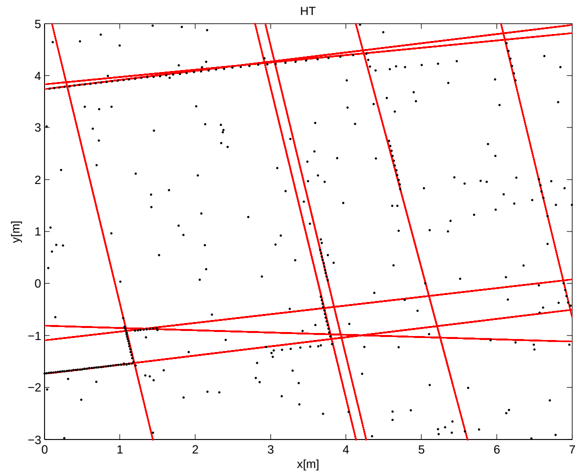


Figure 7: Accumulator of the Hough transform (upper) for data with salt-and-pepper noise and identified straight lines (lower). Maximums in the accumulator (720 row and 720 column array) were identified using the Houghpeaks function in Matlab, where the threshold is set to 5 and the suppression neighborhood to [13, 29].

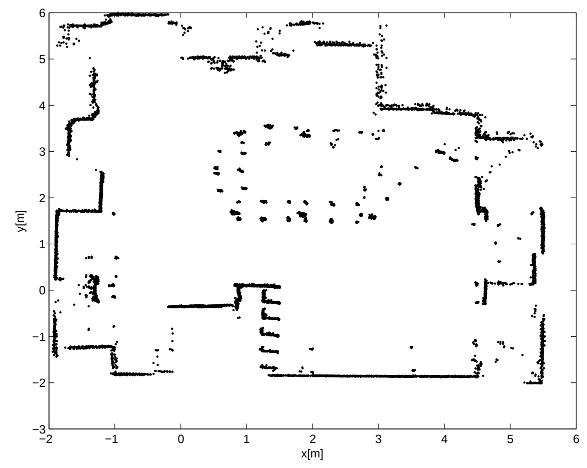


Figure 8: The map of LMSV laboratory obtained from 60 laser range finder scans containing 10800 points.

8. Each scan contains 180 points, so in total Fig. 8 consists of 10800 reflection points. Each scan is evaluated separately by the number of estimated line clusters relative to the number of true clusters. A true line contains at least 4 sample points, where the distance between consecutive points is less than 0.2 m. A total of 561 straight lines can be found on all 60 scans or approximately 9 per scan. In Fig. 8 a cumulative scan is shown where local scans are merged together using the SLAM algorithm, which is also the reason for some additional noise in the observed groups of points that belong to a straight line (error in the robot-pose estimate). In separate scans the error due to the SLAM is not present; however, there are many outliers or smaller groups of points belonging to curved objects, chair legs, humans and the like.

In Table 2 the obtained clustering results are given by average processing time (Matlab implementation on 2.6-GHz personal computer), by the percentage of correctly estimated line segments (according to the number of true straight lines in the scan) and by the percentage of wrong estimates (estimated lines without a match in the true line-set relative to the number of estimated straight

Table 2: A comparison of line-extraction algorithm performance on the laser-range-finder data shown in Fig. 8.

algorithm	comp. time [ms]	true est. [%]	false est. [%]
EPCC	23	90.2	10.3
SM	44	87.9	13.0
HT	774	76.3	17.5

lines).

The clustering parameters were chosen as follows: $\kappa_{max} = 7$, $k_{min} = 3$ for EPCC, $d_{split} = 0.06$ m for SM and quantization of the normal line parameters and the threshold (for locating maximums in the accumulator) in the Hough transform were $0.5°$, 1 cm and 5, respectively. In all the algorithms only the estimated lines containing at least 4 points are considered. From the comparison it can be concluded that for sorted data streams the EPCC's performance is at least similar to or better than, the performance of the SM or HT, while the computational time of the EPCC is much shorter. The classic HT algorithm performs the worst; however, its performance can be improved by additionally considering the information from successive data, which is available in sorted (batch or stream) data where a new cluster can be formed only from the successive samples if the distance between the neighboring data is sufficiently small.

The clusters in Figs. 3-8 are crisp with no or very little overlapping because the LRF sensor can only measure visible reflection points (i.e., it cannot measure reflection points behind the wall corner). The proposed algorithm is applicable also to the overlapped clusters data (the linear prototypes are not collinear and cannot be described by a single linear model). If the data with overlapped clusters are not sorted data stream then the Algorithm 2 must be used.

The proposed algorithm can also be applied to 3D LRF where $r = 3$ and the linear prototype is plane. If the obtained 3D LRF data stream is sorted then Algorithm 1 can be applied otherwise if due to scanning pattern data stream is not sorted then the Algorithm 2 need to be used.

4. Conclusion

A novel evolving clustering algorithm has been developed to identify clusters defined by linear models from particular process data. The algorithm can be used for online or batch clustering. For ordered data streams such as the one from a laser range finder the EPCC produces similar results to the SM, but requires less than half of the computational effort of the SM. For general data streams the EPCC algorithm requires a double iteration for the loop, so the algorithm complexity increases: however, the basic SM algorithm cannot be used in this case, as it requires ordered data.

A practical advantage of the EPCC is that it does not need least-squares fitting methods to estimate the linear models. It only requires a recursive estimation of the data variance, which is much simpler to implement on embedded systems with lower performance. The EPCC algorithm is easy to tune to various processes as it only has one parameter. The clustering criterion depends on the currently estimated distance variance from the cluster prototype. The clustering performance therefore adapts automatically to the process noise, so less tuning effort is required and a higher clustering robustness is achieved. The EPCC algorithm is therefore a good choice for linear prototype extraction in sorted data streams due to its simplicity of implementation, low computational complexity and good estimation accuracy. It is therefore applicable to SLAM related problems where on-line localization is done using estimated strait lines from the laser range finder data.

References

[1] S. Thrun, Robotic mapping: A survey, in: G. Lakemeyer, B. Nebel (Eds.), Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann, San Francisco, 2003, pp. 1-35.

[2] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, M. Csorba, A solution to the simultaneous localization and map building

(SLAM) Problem, IEEE Transactions on Robotics and Automation 17 (3) (2001) 229–241.

[3] H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping: Part i, IEEE Robotics & Automation Magazine 13 (2) (2006) 99–110.

[4] N. Musavi, J. Keighobadi, Adaptive fuzzy neuro-observer applied to low cost INS/GPS, Applied Soft Computing 29 (2015) 82–94.

[5] U. Larsson, J. Forsberg, A. Wernersson, Mobile robot localization: Integrating measurements from a time-of-flight laser, IEEE Transactions on Industrial Electronics 43 (3) (1996) 422–431.

[6] L. Teslić, I. Škrjanc, G. Klančar, EKF-based localization of a wheeled mobile robot in structured environments, Journal of Intelligent and Robotic Systems 62 (2) (2011) 187–203.

[7] X. Zhang, A. B. Rad, Y.-K. Wong, A robust regression model for simultaneous localization and mapping in autonomous mobile robot, Journal of Intelligent and Robotic Systems 53 (2) (2008) 183–202.

[8] M. Begum, G. K. I. Mann, R. G. Gosine, Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots, Applied Soft Computing 8 (2008) 150–165.

[9] G. A. Borges, M.-J. Aldon, Line extraction in 2D range images for mobile robotics, Journal of Intelligent and Robotic Systems 40 (3) (2004) 267–297.

[10] D. Borrmanna, A. Nüchtera, M. Dukalović, I. Maurović, I. Petrović, D. Osmanković, J. Velagić, A mobile robot based system for fully automated thermal 3D mapping, Advanced Engineering Informatics 28 (4) (2014) 425-440.

[11] D. H. Ballard, Generalizing the hough transform to detect arbitrary shapes, Pattern Recognition 13 (2) (1981) 111–122.

[12] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, R. Siegwart, A comparison of line extraction algorithms using 2D range data for indoor mobile robotics, Autonomous Robots 23 (2) (2007) 97–111.

[13] L. Xu, E. Oja, Randomized hough transform (RHT): Basic mechanisms, algorithms, and computational complexities, CVGIP: Image Understanding 57 (2) (1993) 131–154.

[14] J. Basak, A. Das, Hough transform network: Learning conoidal structures in a connectionist framework, IEEE Transactions on Neural Networks 13 (2) (2002) 381–392.

[15] T. Pavlidis, S. L. Horowitz, Segmentation of plane curves, IEEE Transactions on Computers 23 (8) (1974) 860–870.

[16] T. Kenesei, J. Abonyi, Hinging hyperplane based regression tree identified by fuzzy clustering and its application, Applied Soft Computing 13 (2013) 782–792.

[17] L. Zhu, F.-L. Chung, S. Wang, Generalized fuzzy C-means clustering algorithm with improved fuzzy partitions, IEEE Transactions on Systems Man and Cybernetics Part B (Cybernetics) 39 (3) (2009) 587–591.

[18] D. Dovžan, I. Škrjanc, Recursive clustering based on a Gustafson-Kessel algorithm, Evolving systems 2 (1) (2011) 15–24.

[19] R.-E. Precup, M.-C. Sabau, E. M. Petriu, Nature-inspired optimal tuning of input membership functions of Takagi-Sugeno-Kang fuzzy models for anti-lock braking systems, Applied Soft Computing 27 (2015) 575–589.

[20] P. Angelov, An approach for fuzzy rule-base adaptation using on-line clustering, International Journal of Approximate Reasoning 35 (3) (2004) 275–289.

[21] M. A. Fischler, R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM 24 (6) (1981) 381–395.

[22] X. Bai, W. Yao, J. E. Boyer, Robust fitting of mixture regression models, Computational Statistics and Data Analysis 56 (7) (2012) 2347–2359.

[23] A. Cerioli, D. Perrotta, Robust clustering around regression lines with high density regions, Advances in Data Analysis and Classification 8 (1) (2014) 5–26.

[24] W. Li, H. H. Yue, S. Valle-Cervantes, S. J. Qin, Recursive PCA for adaptive process monitoring, Journal of Process Control 10 (5) (2000) 471–486.

[25] A. Bellas, C. Bouveyron, M. Cottrell, J. Lacaille, Model-based clustering of high-dimensional data streams with online mixture of probabilistic PCA, Data Analysis and Classification 7 (3) (2013) 281–300.

[26] E. Anderson, Z. Bai, C. Bischof, S. J. Blackford, J. Demmel, J. Dongarra, J. Du. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK User's Guide. Third Edition, SIAM, Philadelphia, 1999.

[27] SICK, Reute, Germany, LMS 200 / LMS 211 / LMS 220 / LMS 221 / LMS 291 Laser Measurement Systems (2002).

[28] L. Li, A new complexity bound for the least-squares problem, Computers & Mathematics with Applications 31 (12) (1996) 15–16.

[29] S. Barchinezhad, M. Eftekhari, A New Fuzzy and Correlation Based Feature Selection Method for Multiclass Problems, International Journal of Artificial Intelligence 12 (2014) 24–41.