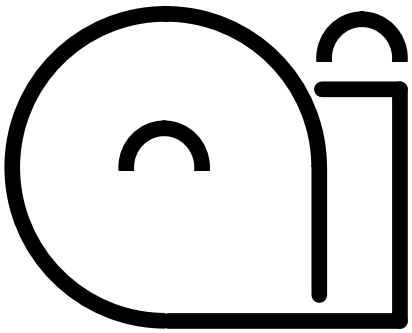


Österreichisches Forschungsinstitut für / Austrian Research Institute for / Artificial Intelligence

TR–2003–10

Patrick M. Pölz, Hörtnagl Erik, Prem Erich

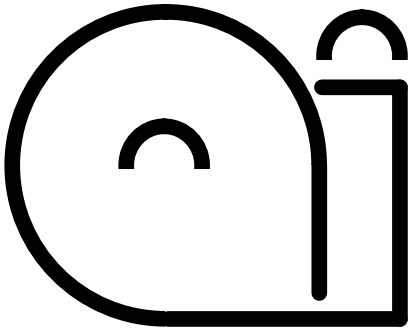
Processing and Clustering Time Series of Mobile Robot Sensory Data

• Schottengasse 3 • A-1010 Vienna • Austria •

• Phone: +43-1-5336112 •

• mailto:sec@oefai.at •

• http://www.oefai.at/oefai/ •



Österreichisches Forschungsinstitut für / Austrian Research Institute for / Artificial Intelligence

TR–2003–10

Patrick M. Pölz, Hörtnagl Erik, Prem Erich

Processing and Clustering Time Series of Mobile Robot Sensory Data

The Austrian Research Institute for Artificial Intelligence is supported by the Federal Ministry of Education, Science and Culture.

Citation: Pölz P., Erik H., Erich P.: Processing and Clustering Time Series of Mobile Robot Sensory Data. Technical Report, Österreichisches Forschungsinstitut für Artificial Intelligence, Wien, TR-2003-10, 2003

SIGNAL

IST-2000-29255

Systemic Intelligence for GrowiNg up Artefacts that Live

Processing and Clustering Time Series of Mobile Robot Sensory Data

Technical Report OFAI-TR-2003-10

Patrick M. Pölz, Erik Hörtnagl, Erich Prem

Version: 02.12.3.1

Date: 13.02.2003

Classification: Public

Project Coordinator: Div. of Neuroinformatics, Dept. of Computer Science, University of Bonn (Germany)

Partners: Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Genoa (Italy)

Austrian Research Institute for Artificial Intelligence (OFAI), Vienna (Austria)

School of Computing, Napier University, Edinburgh, Scotland (UK)

Project funded by the European Community under the "Information Scociety Technologies" Programme (1998-2002)

Processing and Clustering Time Series of Mobile Robot Sensory Data

Patrick Pölz, Erik Hörtnagl, Erich Prem

Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A1010 Vienna, Austria
{erik, Patrick, erich}@oefai.at

Abstract. This report describes work on clustering the time series produced by sensor readings of a mobile robot. Work reported here was driven by the necessity to distinguish different levels of processing. The first part of this report copes with experiences gained and mechanisms developed in our implementation of several algorithms in the different stages of processing the time series. This also includes event detection. The second part attends to clustering the pre-processed time series using a fixed window. We compare the technique of dynamic time warping versus using a Euclidian distance as measure for aligning series elements. We found the Euclidian distance technique to perform considerably faster than dynamic time warping while achieving results of comparable quality.

1 Introduction

For a mobile robot that builds categories of its surroundings based on sensor readings it is a necessary first step to abstract from raw sensor data towards a more reasonable and understandable representation of its environment and actions. The purpose of this paper is to report experiences gained in applying a number of techniques for pre-processing and for clustering windows of time-series data.

In our research, we use a wheeled mobile robot, Kurt-2. It is designed as a test platform for autonomous robotic behaviours and quite practicable for the type of experiments reported here. Kurt-2 is controlled by a notebook carried on the robot's back. The notebook runs our developing environment (cf. Poelz 2002). This allows us to implement the control architecture in a very comfortable way. The robot is equipped with a total of fourteen sensors: two ultrasonic and twelve infrared.

The sensors deliver a vector of these fourteen sensor values which are sampled at a frequency of 10Hz, thus resulting in a time-series. The digitised ultrasonic sensor readings range from a value of about 1024 (meaning nothing is detected) to 50 (meaning something is directly in front of the sensor, i.e. at a distance of 0 to 2 centimeters). The distance measure of the infrared sensors is inverted ranging from a value of 512 (something is about 10 centimeters ahead to the sensor) to about 100 when the sensor does not detect anything. Objects very near to infrared sensors let the

readings drop beyond a value of 100 again, but the data gathered within this interval is fluctuating in a way that makes it useless for further processing. Figure 1 depicts a time window of two seconds and displays all sensor readings without any preprocessing. The depicted situation shows the sensor signals as the robot was passing a little box.

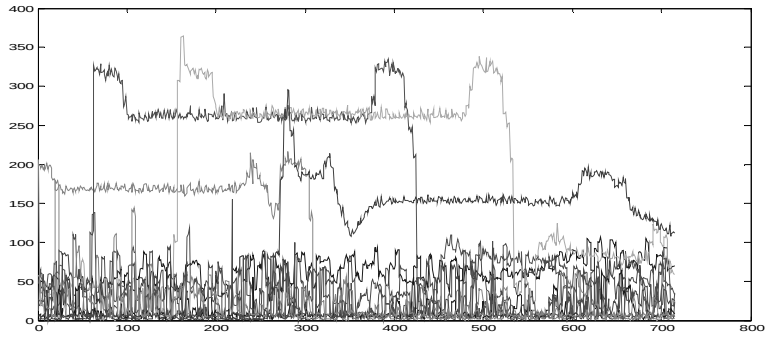


Fig. 1. Two Seconds of infrared sensor readings without any preprocessing.

There is a lot of noise in the data and its hard to make any sense of this two seconds. However, on a second glance, there is already some structure visible for the human eye: for the left side (the four graphs surmounting all others), where the box is passed, typical readings of the two left infrared sensors can be identified. But it is easy to see how hard it must be for any robot control software, that cannot take our global point of view, to find something meaningful in this data. Also, taking into account that the robots does not have the extra information that it just passed a box, the task becomes even more difficult.

2 Preprocessing

Preprocessing consists of several steps so as to render the sensory data smoother, reduce noise and transform the readings to be able to use the same mechanisms on the whole data vector of all sensors.

Roughly following [Keough & Pazzani, 1998] we may distinguish the six typical problems depicted in Figure 2 that make the categorization of data difficult. Figure 2 depicts varies forms of typical disturbances in the sensor readings:

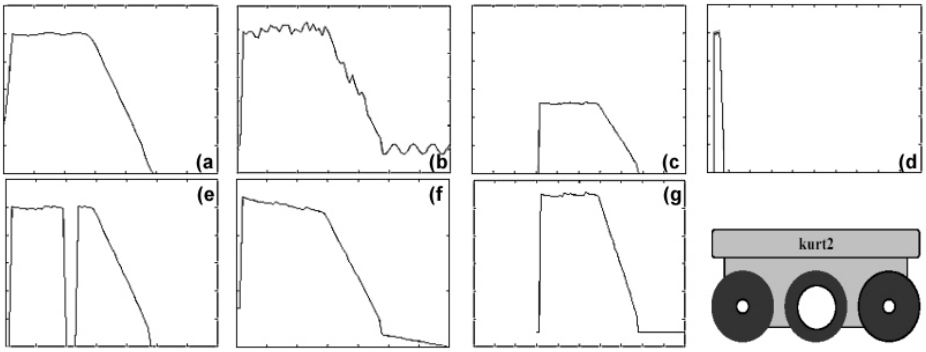


Fig. 2. (a) undisturbed signal, (b) noise, (c) amplitude scaling, (d) longitudinal scaling, (e) discontinuities, (f) linear drift, (g) offset translation

We will now discuss the mechanisms we use to overcome these deficiencies going through a number of data processing steps.

2.1 Filtering

The first step of preprocessing is to apply a simple filtering mechanism, taking out discontinuities like runaways and peaks. We just cut off all values above 950 and beyond 50 for ultrasonic respectively 500 and 100 for infrared sensors. Figure 3 displays the previously discussed situation of passing a box after this first step. Especially for the infrared sensors, the filtering turns out be an indispensable step, because of the high noise found in the data range below a value of 100, which otherwise would be confusing for any clustering algorithm (see also chapter 4.2.1).

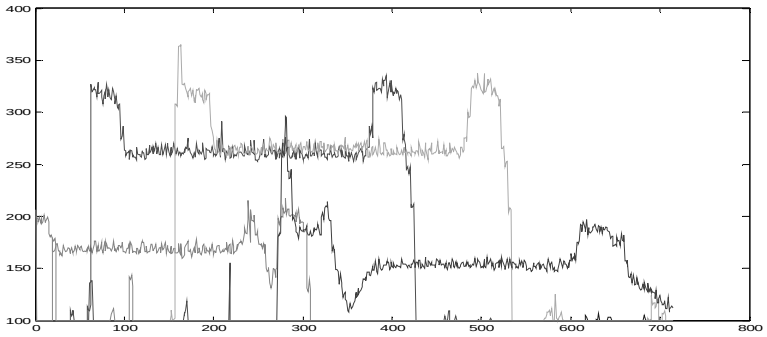


Fig. 3. . Two seconds of infrared sensor readings after filtering.

2.2 Smoothing

After filtering we smoothen the data. This is done by taking a specific amount of successive vectors and computing the mean of their row values. A reasonable number of vectors to use lies between four and ten, where ten means very strong smoothing of the data. Figure 4 compares the smoothing factors four (l) and ten (r) in respect to the example used before. This step removes most of the noise embedded in the data stream and also flattens, intensity depending on the factor, most remaining discontinuities.

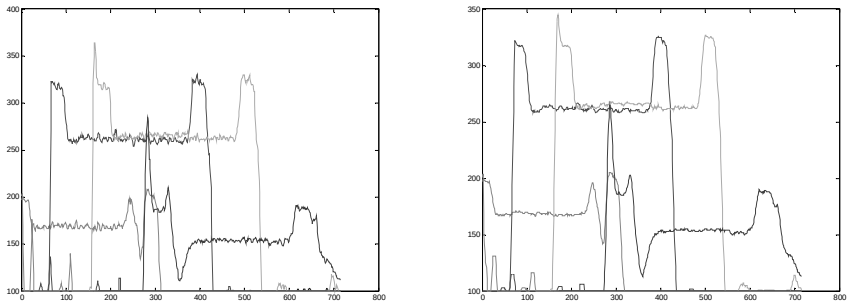


Fig. 4. Two seconds of infrared sensor readings. Smoothing factor four (l) and ten (r).

2.3 Standardization & Normalization

The last step in preprocessing performs standardization and normalization. First, all data is projected on the interval [0; 100] and the ultrasonic data is inverted. Thus a rising sensor reading means approaching an object, just like in the infrared domain. Figure 5 shows two seconds of real robot data, now concentrating on the front

ultrasonic sensor. Notice that there is not that much going on as in the infrared domain, which is a result of the location of the sonar sensors.

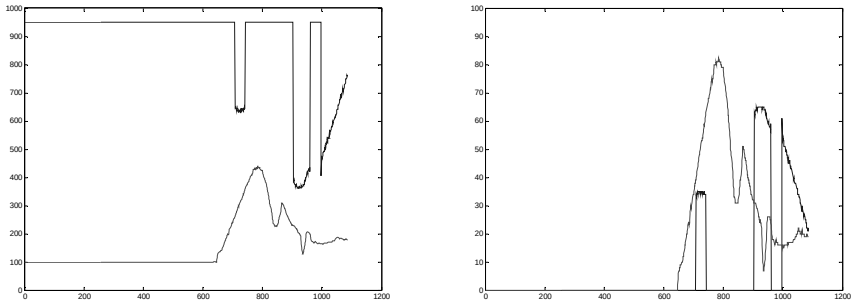


Fig. 5. Two seconds of ultrasonic data (front sensor) plus one infrared sensor in contrast. (l) Original data (after being filtered). (r) Standardized data. Notice the different scaling.

At last the data is normalized to deal with amplitude scaling (Figure 2.c). In this processing step distance information, which is irrelevant and more or less disturbing for most clustering algorithms, is removed from the data. This step - although it could be regarded as preprocessing - takes place after event detection (cf. chapter 3), because it must be applied to a time window of specified length. Figure 6 compares an ultrasonic signal before (l) and after (r) normalization.

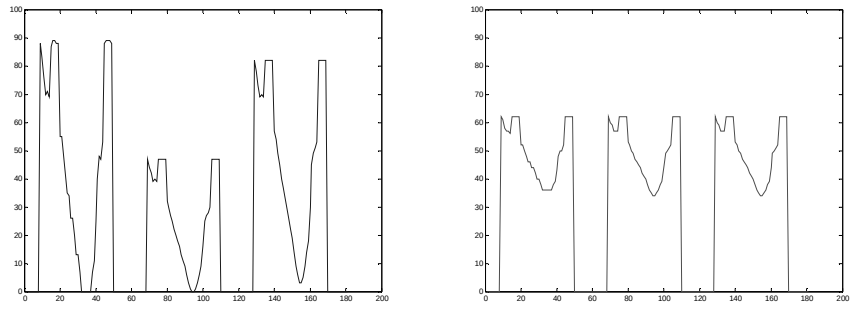


Fig. 6. Ultrasonic signal before normalization (l) and after all preprocessing steps (r).

3 Event Detection

The most important processing step towards categorization of sensor data is the division of the continuous stream of data into time windows of specific length that consist of something meaningful to the robot. Events could be defined as changes in the sensor readings that last for a specific time. They mark the beginning of a significant deviation to normal state, in respect to recognizing changes to the surroundings or the robot itself. While the previously discussed preprocessing steps are computed on the whole vector of data, we decided to process every sensor type independently for event detection, because of the different courses of their data streams.

Figure 7 shows a segment taken from ultrasonic sensor data recorded by our robot during a sequence of typical behaviors (i.e. passing through an edge, turning around, etc.). The recording consists of 300 sample data points recorded at a frequency of 100Hz. It is easy to separate the data into four to six events followed by some specific readings just looking on the falling and rising edges in the stream. That is exactly the way in which we try to make our robot detect these events, but as we will see in fact it is not that easy at all.

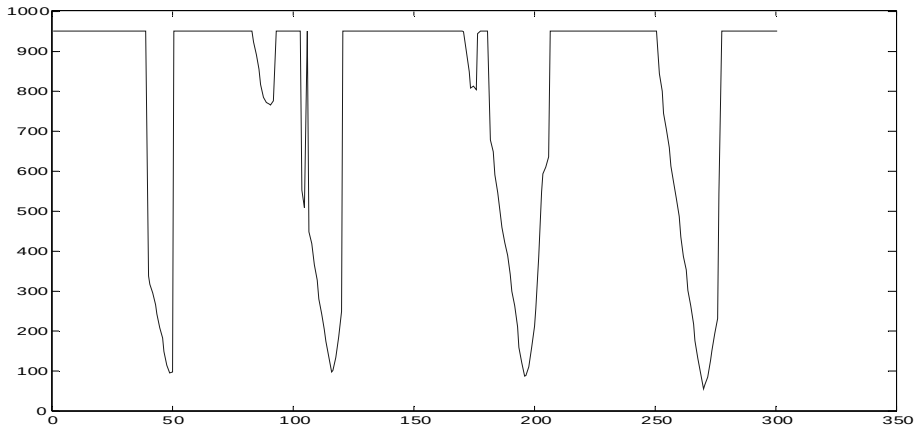


Fig. 7. Data segment of ultrasonic sensor data

3.1 First derivative

A very simple but effective approach (cf. Figure 8) for detecting the beginning (and even the end) of an event is to calculate the first derivative against a specific threshold:

$$\text{abs}\ (\ \text{data}\ (i+1,\ j) - \text{data}\ (i,\ j)\ ) > \text{threshold} \qquad (\mathbf{1})$$

Using different thresholds gives different sensitivity to the detection algorithm, where higher thresholds result in lower sensitivity. We found out that a fixed threshold in range of about 30% of the whole spectrum turns out to be reliable in the ultrasonic sensor domain, but this strongly depends on the sensor stream taken into account. Figure 8 shows the data stream of the previous picture, but this time includes event detection using the first derivative.

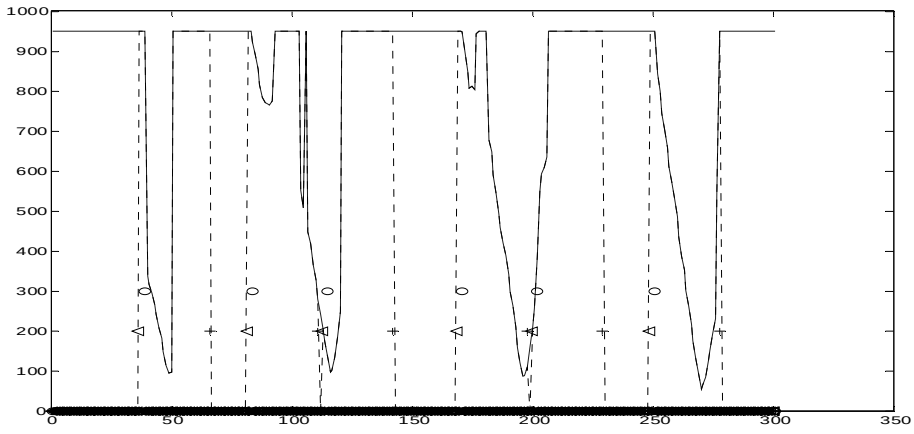


Fig. 8. Event detection of time series using first derivative method with a threshold of 30 and a fixed window size of 30 data points (Legend: **o** Event detected, ▷ starting point of event window, + end point of event window).

In this graph, the window size was chosen fixed at 30 points per event, which introduced some undesired artefacts: The first event includes many uninteresting points at the end, while the second was split into two parts (cf. Chapter 3.3 for further discussion of this problem).

3.2 Variance method

A problem of the first derivative method in conjunction with a fixed threshold value are smoothly changing sensor readings or events that happen at the horizon of sensor sensitivity, because they could be - and actually are – easily missed.

A possible solution would be a sliding threshold that adapts to the circumstances found in the surroundings. This solution is found in biological systems, where a comparable phenomenon would be the increasing production of rhodopsin in the rods within the retina of mammals in lower light conditions [Thiele, 2002] which results in better vision at night while also being able to see in bright sunlight.

We use a relation of the actual threshold to the variance of the previously seen time series. If much is happening the variance is high and vice versa, so the threshold gets lower when nothing happens within the actually set threshold and gets higher if too much events start confusing the robot. Thus the length of the data taken into account when computing the variance is a measure for the adaptivity of our threshold value. As we will discuss in the next chapter with regards to time window size, the threshold value and the variance inside the time windows also clearly depend on the surroundings. In our small test scenarios, the variance method gave no significant improvement to event detection.

3.3 Fixed Time Window Size

The major problem in event detection is not to find their beginning but to guess the duration of an event, i.e. to detect the corresponding end to an event. In literature mostly fixed event length, based on statistical experience is used and propagated to lie in the range of 5 seconds for a mobile robot's sensor readings (still depending strongly on the surroundings). However it is hard to say how long this fixed window length should be in a specific case like ours.

As depicted in Figures 9 to 11 a fixed window size can cause two major problems. First, the inclusion of uninteresting data at the end of a time window and second the unlucky splitting of an event into two meaningless parts. Because there is no perfect size for all events, one has to decide for the right length. This lies somewhere between having a time window of a size that fits virtually any possible event with the risk of having lots of noise at the end of an event or even getting into the next one (Figure 10), or using small time windows with many events split up into one or more subevents (Picture 11).

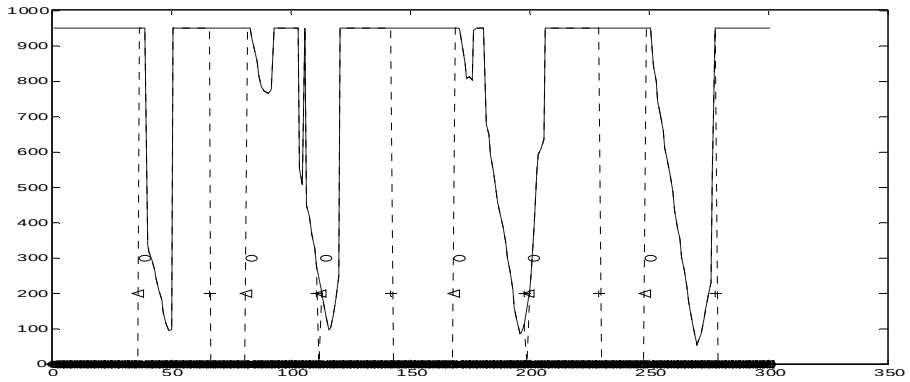


Fig. 9. Time window size = 30

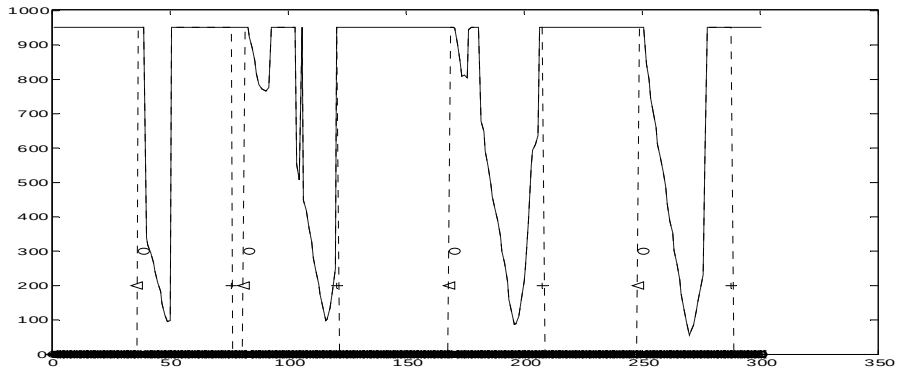


Fig. 10. Time window size = 40

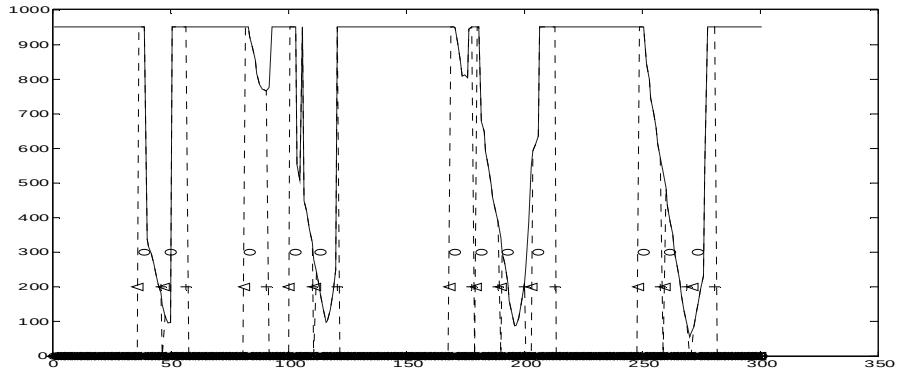


Fig. 11. Time window size = 10

Legend for Figures 9–11. Time series is sketched as full line, the detected time window is bordered by dotted lines. **o** Event detected, ▷ starting point of event window, **+** end point of event window.

For further work and comparison tasks we used a fixed time window size of 30 sample points and 30 as a fixed threshold for event detection.

3.4 Variable Time Window Size

Taking into consideration the results of Section 3.3 the best behaviour for a event detection mechanism would be to catch virtually all events as whole no matter what size they are. The realization would result in the need for a variable window size, producing different windows for all events (in many cases even if they are identical).

The first difference to the algorithm discussed so far would be a mechanism to also detect the endpoints of occurring events. This can be easily accomplished by just

inverting the algorithm we described in 3.1. Figure 12 shows the event windows which were found by this method. It detects the events starting and ending points by the first derivative threshold algorithm.

This way another problem arises. To speed up computation and allow for easy programming structures, etc. we decided that our clustering algorithms are always provided with fixed length time windows. Thus two mechanisms are needed, one to stretch a time window if it is shorter than needed and another that compresses them.

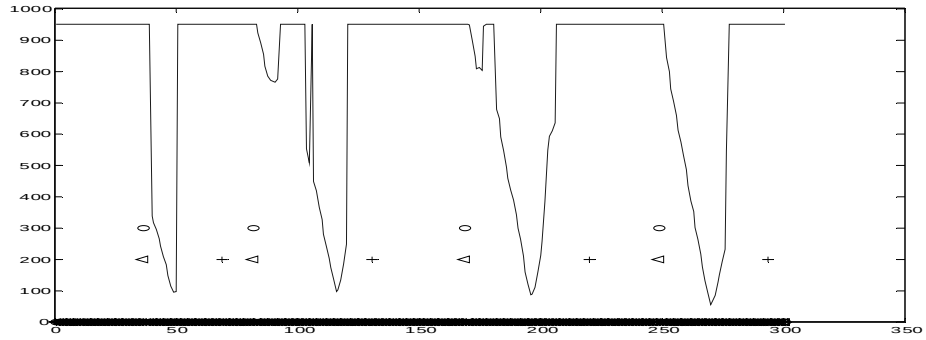


Fig. 12. First derivative event detection of time series generating variable window size

3.3.1 Stretching Time Windows

If the length of the time window of a detected event is shorter than the fixed window size needed for clustering, then the window has to be stretched. This means we have to add new points that do not change the event at all, or at least as little as possible.

A simple algorithm that is sufficiently speedy and results in most of the events lasting longer as our predefined window size, is the following (cf. Figure 13):

1. Calculate the number *n* of data points missing.

2. In our sample find the *n* pairs of data points that are closest to each other.

3. For every pair of these *2n* points insert one more data point in between using the average on both axes.

4. If there are still points missing go to step 1 again taking into account also the newly gained points.

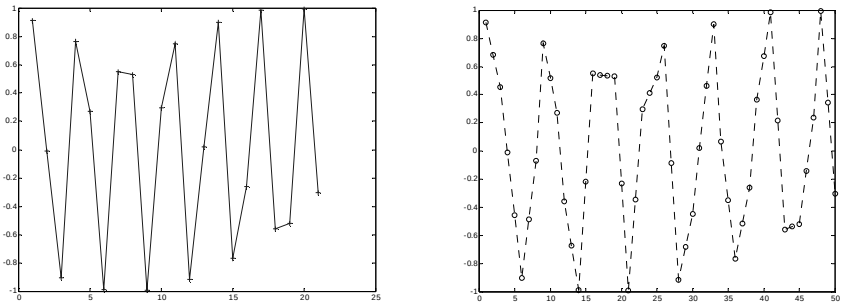


Fig. 13. Stretching a time window consisting of about 20 data points (l) to a size of 50 (r).

3.3.2 Compressing Time Windows

Shrinking event time windows turns out to be a little more complicated, because of the need to remove data points while preserving the information of the data sample at best. The challenge here is to detect which points are of main interest and which ones could be deleted safely. [Fu *et al.*, 2001] propose an algorithm in which the *Perceptually Important Points* can be found easily and fast. We adapted this algorithm to fit our needs and will now briefly explain how it works.

The first two points taken are starting point and end point of the time series (Figure 14 (l)). The next point is chosen to be the one with maximum distance on behalf of the points already in our list (Figure 14(r)). The distance is computed using the following metric:

$$\text{distance} = \text{abs}\left(y_1 + (y_2 - y_1) \,.\, \left((x_3 - x_1) / (x_2 - x_1) \right) - x_3 \right) \qquad (2)$$

where: $x_1, y_1; x_2, y_2$ … Starting and end-point coordinates
x_3, y_3 … Point actually compared

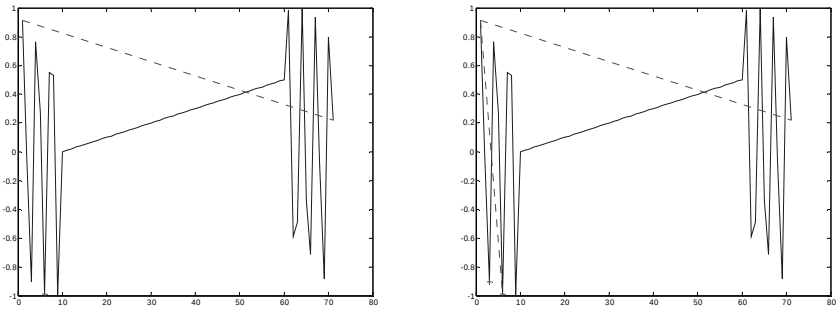


Fig. 14. PIP-Algorithm first two steps. + has maximum distance.

In the next step the new point is added to the list of perceptually important ones and used as new end respectively starting point for the determination of two more points. The algorithm is repeated until the wanted number of points is reached. Figure 16 depicts the time series after being processed by the PIP and a stretching algorithm in contrast to Figure 15 that shows the original time series.

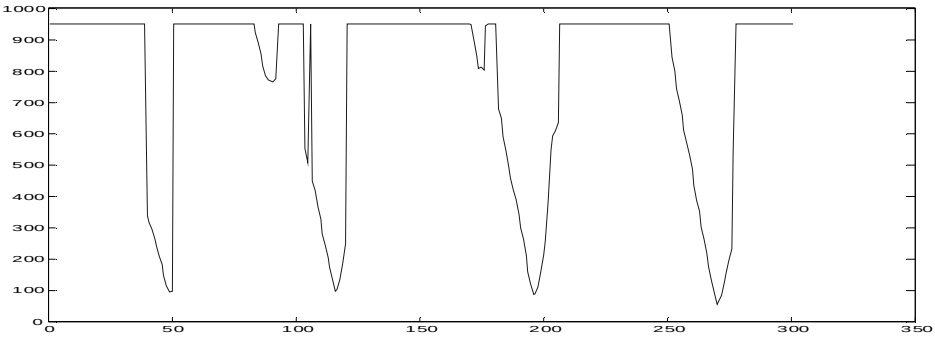


Fig. 15. Original time series before processing

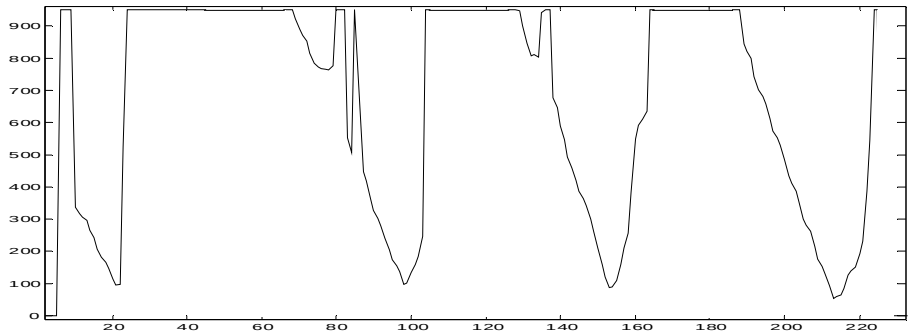


Fig. 16. Events detected with a variable time window size method (namely PIP and a self developed stretching algorithm). The occurring events have been stretched and compressed to fit into a time window of width 40. Notice that the first event has been stretched while all others were compressed (be aware of the different scaling).

Now all events have the same length and can be transferred to the clustering algorithms, but there remains one problem:

Compression and stretching are longitudinal scaling operations (see Figure 2.d). Thus the time series of detected events looses some of its temporal information after this processing step. But again this could be solved easily by just saving some extra information like the time an event lasts.

On the other hand, we are able to use this as an advantage as we will see later when comparing the simple Euclidian clustering to the complex dynamic time warping algorithm.

3.4 Event detection – A closer look

Let us now take a closer look at event detection for the infrared sensors. Here we have to distinguish the different sensors according to their direction and location on the robot. We can form three groups:

Their are 4 diagonal sensors in each corner of the robot:

Location	Sensor
Left front corner	ir4
Right front corner	ir6
Left back corner	ir1
Right back corner	ir9

There are 2 sensors on both sides of the robot

Location	Sensor
Left side front	ir2
Left side back	ir3
Right side front	ir7
Right side back	ir8

Their are 2 sensors observing the front and 2 sensors observing the back side of the robot:

Location	Sensor
Front left	irA
Front right	irB
Back left	irC
Back right	irD

The behaviour of the different sensors and the events they detect are tied strongly to the location in which they are located. This makes individual event detection for each of these sensor groups necessary.

3.4.1 Corner infrared sensors

Figure 17 depicts the frontal infrared sensors while the robot moves around in our tested. You may notice that the two sides differ a lot. This difference emerges of the surroundings that in conjunction with the simple behaviour of our robot make it follow a wall when it found one. Since there is no need to turn around, most observations are made on the right side and furthermore there is almost always sensor deflection on the right side.

For event detection we have chosen a high smoothing factor of ten (see chapter 2.2) a low threshold of ten percent for the event detection algorithm. This takes out most of the noise disturbing event detection while the low threshold still allows sensitive recognition of the ongoing events. Figure 18 shows the two sensor data streams after pre-processing and event detection.

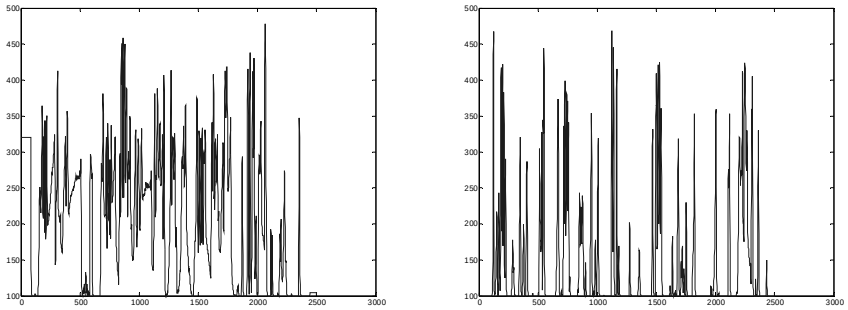


Fig. 17. Left and right diagonal front corner infrared sensors. Raw data.

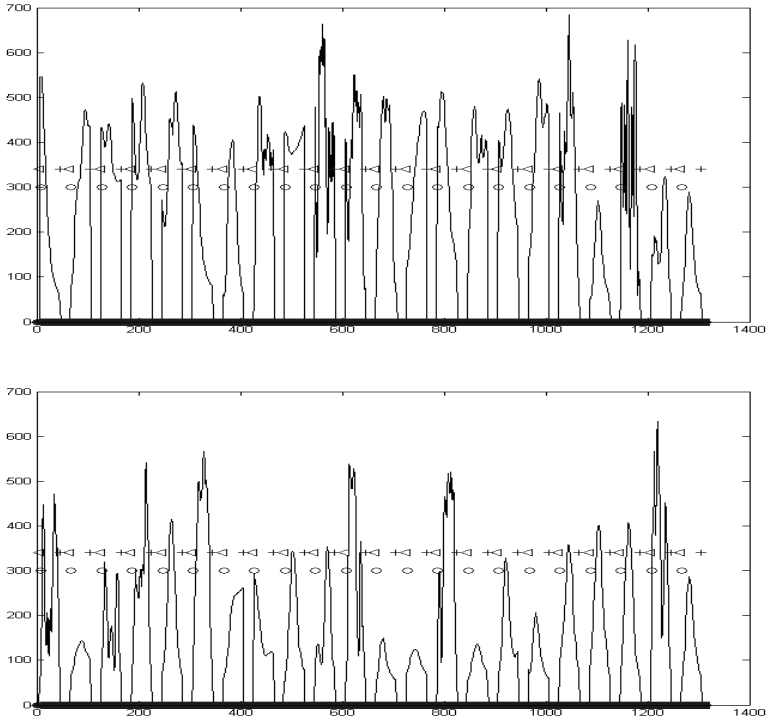


Fig. 18. Left and right diagonal front corner infrared sensors. After Preprocessing and event detection.

3.4.2 Side infrared sensors

In this sensor group, in the beginning it was virtually impossible to detect anything meaningful. But with some fine tuning in the parameters we managed to achieve a more or less satisfying behaviour of our detection algorithm.

Figure 19 shows the sensory stream of the side infrared sensors. The above two figures show the right, the lower on the left side. Having a closer look on the two streams of the infrared sensors on the right side, it can be seen that they are strongly correlated. The main difference is some time delay. Of course the same characteristic holds for the sensors on the left side. Again, beside other fine tuning we use a smoothing factor of ten in pre-processing and a threshold of ten for the detection algorithm.

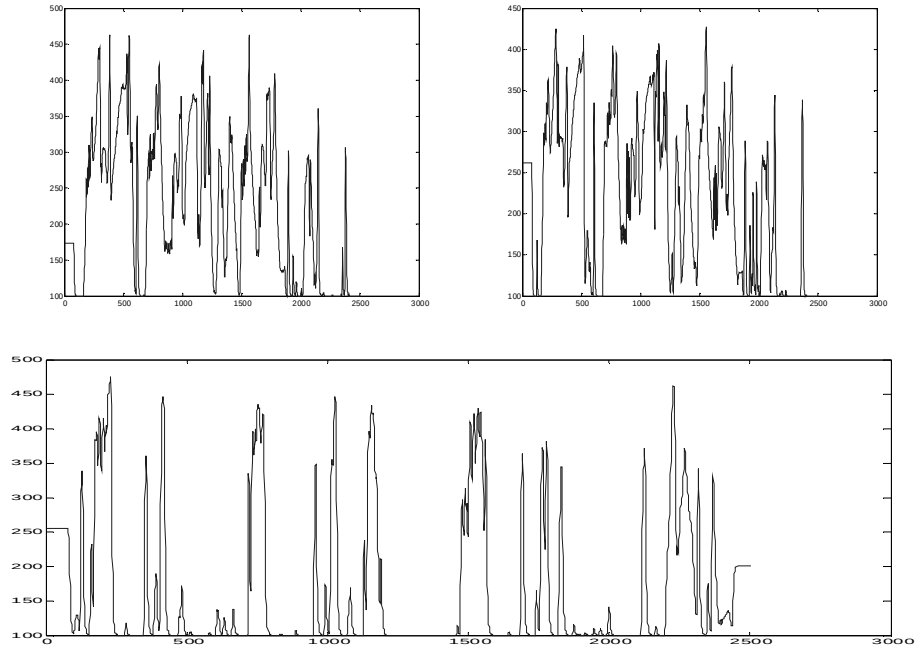


Fig. 19. Sensory stream of the side infrared sensors. Above: sensor stream of the right side front sensor (r), right side back sensor stream (l). Beneath: left side front sensor (back sensor omitted because of strong correlation – see text).

3.4.3 Front infrared sensors

Generally, front infrared sensors can be added to the group of the ultrasonic sensors with respect to their detection parameters. They are used for detecting the same objects and show similar behaviour, especially after pre-processing.

4 Clustering of events[1]

Cluster analysis is a collection of statistical methods used to group data points. Group members will share certain properties and it is hoped that the resultant group structure will be useful for the task at hand. The classification has the effect of reducing the dimensionality of a data table by reducing the number of cases. Put differently, clustering quantizes high-dimensional data into a number of discrete cluster centres (also called code book).

In the domain of mobile robot research, particular interest lies on clustering methods that work on-line, i.e. assign novel inputs to clusters as data is collected and change the clusters accordingly. Similar events should be grouped and the clusters be assigned an alphabet (a code book), such that similar events are assigned the same symbol, signature or code book vector.

After clustering, a prototype or a best example for each cluster will be available. Normally either a member that minimizes the distance to all other members of its cluster, i.e. the average of all members in the cluster is taken (as, for instance, happens in the well-known k-means clustering algorithm). Alternatively, we may also choose to remember all the members of a specific cluster. In our case, data clusters should be made from time series of similar events perceived by the sensors of our robot.

Every clustering algorithm requires a measure of instance similarity, or dissimilarity to guide its decisions about cluster membership. Finding a measure of similarity is difficult, because experiences that are qualitatively the same may be different in length (different speed stretches the experiences), time progress may be nonlinear (velocity of the robot through speed variances), or there may be noise and discontinuities (see chapter 2.2).

4.1 Clustering algorithms

Among various algorithms in literature we have chosen two and evaluated their performance in our problem domain. We have created several sets of test data and in the following we will discuss our experiences with dynamic time warping and k-means-clustering using Euclidian distance measure in clustering the time series produced by our mobile robot.

4.1.1 Dynamic Time Warping

Dynamic time warping or short DTW is a technique for defining similarities between time series independently of their individual time courses. Unfortunately, it is a relatively costly method making it difficult to apply in robotic online learning. It can also be used for (non real time) event-detection. DTW calculates the similarity of experience E1 and experience E2 by finding the warping in the time dimension of E1, which minimizes the difference between the two experiences E1 and E2, and then determining the area between the two curves. Warping in this context means

[1] Parts of the following paragraphs are taken from [Prem *et al.*, 2002]

stretching and shrinking the time axis. DTW uses dynamic programming to find the optimal warping, that is a low order polynomial of the lengths of E1 and E2 (O(|E1|*|E2|)). Distortions (e.g. shrinkage in one part of the time series and expansion in another) are possible, too.

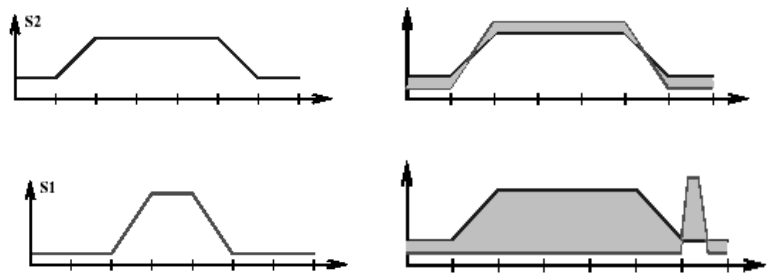


Fig. 20. Experiences S1 and S2; on the right: area between the two curves and warping of S1 which minimizes the difference between the two experiences and the area between the two curves, cf. [Oates *et al.*, 2000].

In [Oates *et al.*, 2000] an algorithm for clustering with DTW is introduced: a complete pair wise distance matrix is calculated by invoking DTW. Next, a standard, hierarchical, agglomerative clustering algorithm is applied that starts with one cluster for each experience and merges pairs with the minimum of average inter-cluster distance. If the mean inter-cluster distances of two clusters are significantly different, then they are not merged. After clustering, a prototype for each cluster is selected.

Note, however, that through DTW important information might be lost - namely the information about the exact time course of the event. Difference in speed, for instance, can matter and should not be disregarded. Therefore, it is wise to keep information about the original time series where relevant - e.g. through separate processing of event shape and speed, similar to what happens in an animal visual system, where different modules process aspects like motion, shape or colour.

For evaluation purposes instead of entirely developing a DTW clustering method we decided to take an off the shelf library implementation provided free for scientific purposes by Intel Corporation [Intel, 1998]. After having solved problems like understanding how to integrate the components into our existing architecture (which turned out to be a hard job, because of the lack of good documentation), we managed to do a lot of tests against our self programmed k-means-clustering algorithm, which will discuss in 4.2.

4.1.2 k-means-clustering

We compared our DTW method to k-means-clustering being one of the simplest clustering algorithms to implement which also provides for high performance

processing of on-line time series.

K-means-clustering is a non-hierarchical clustering method initially taking the number *k* of clusters it is allowed to create. For any new time series the minimum distance to each existing cluster is computed and the new component is assigned to the nearest cluster. As distance measure mostly (and like in our case) the Euclidian distance is used. The cluster's centre is recomputed every time a new component is added, taking their special characteristics into account.

If the distance to all existing clusters is higher than a predefined threshold, a new cluster is generated. It is easy to see that this predefined threshold value in combination with the number of clusters are crucial parameters in respect of how well the k-means-clustering algorithm will work.

4.2 Evaluation of clustering algorithms

4.2.1 Test data sets

For evaluation of the two clustering algorithms we created three different data sets, each one with specific classification attributes that we will discuss now shortly.

4.2.1.1 Four Clusters

There are four main events (depicted in 21), each one slightly diversified five times, resulting in twenty time series that should be put into four different clusters. Figure 22 graphically shows the resulting time series.

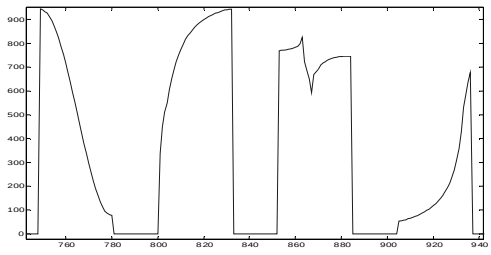


Fig. 21. Four easy distinguishable events

We used this data set in conjunction with the third one for tuning the parameters of the algorithms in respect to setting their threshold values for creating a new cluster, in a way that the twenty time series of set one were just about distinguished the right way and the ones of set three are all put into the same cluster. This was very easy to manage for our k-means-clustering, but turned out to be much more complicated for the DTW libraries, because of hardly no information about the scaling of the distance metric that is used "inside" them.

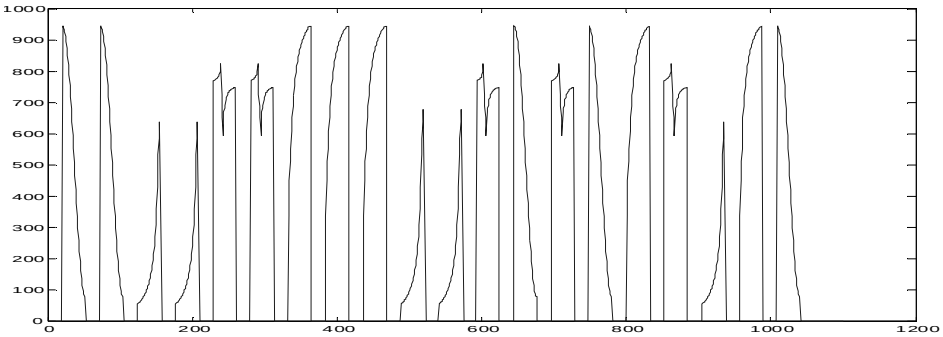


Fig. 22. Time series created by variation of four specific clusters.

4.2.1.2 One cluster

The next test data set consists of six very similar events taken form a test-drive recording of our mobile robot. The events are all slightly different, but as they represent the same event detected by the front sonar sensor, they clearly should be associated to one cluster. Figure 23 again shows a graphical representation of the time series. The two additional events at the end of the data stream are artificial and have to be assigned to separate clusters.

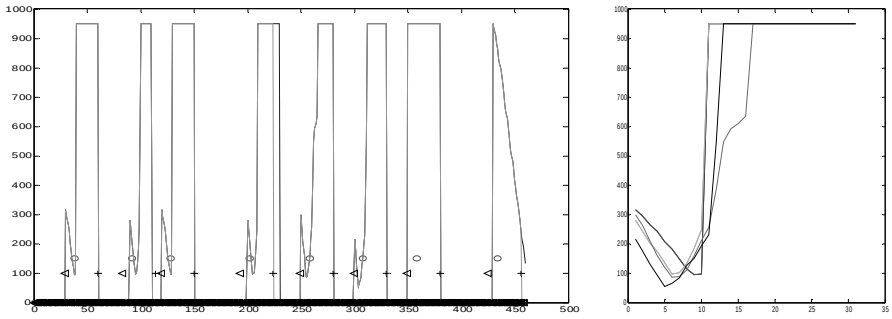


Fig. 23. (l) Graphical representation of six similar robot perceptions plus two artificial events (r) Another representation of the same date highlighting the similarity of the data sets.

4.2.1.3 Disturbed data

The third and last dataset was created for testing an clustering algorithm in terms of how it is able to handle disturbed data. The six different cases used were created inspired by [Keough & Pazzani, 1998] (see chapter 2.2 for detail). Figures 24(1-6)

show these test data sets using the representation we already introduced in 23(r) depicting the similarities of the different sets very clearly. For creation we took one real world data sample and disturbed it several times in the mentioned six different ways, always using increasing strength.

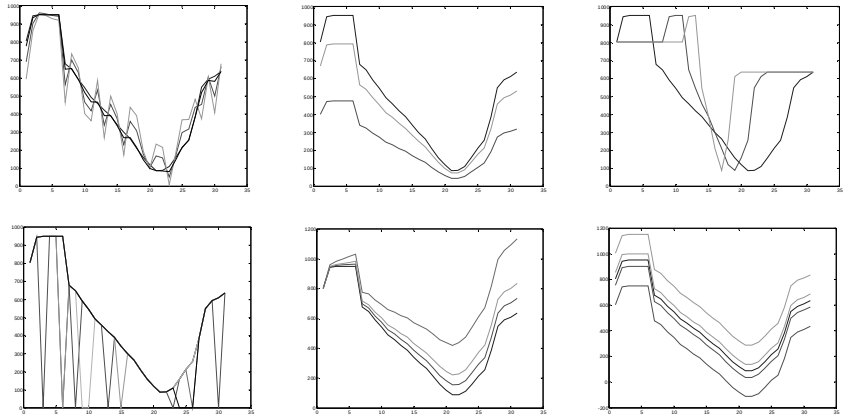


Fig. 24. (from left to right and top down) (1) noise, (2) amplitude scaling, (3) longitudinal scaling, (4) discontinuities, (5) linear drift, (6) offset translation.

Figure 25 shows the graphical representation of the resulting data stream, if the data sets in 24 are added together.

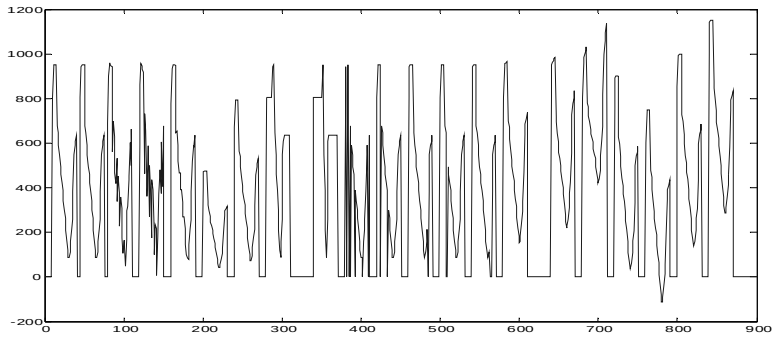


Fig. 25. Data stream composed out of the data sets shown in Figure 24.

Processing this stream a clustering algorithm should create more or less one cluster, depending on how sensitive it reacts on strongly disturbed data. It is hard to set a specific limit (a maximum distance to the cluster's centre respectively our threshold) determining when a data set is disturbed that much it has to be put into a new cluster. The best way to find out is by having a look at the clustering algorithm while some real world test-drives.

4.2.2 Comparisons & conclusions

For both of our algorithms we now made performance tests trying to evaluate their usability and processing speed. We used the following test scenario:

The platform consists of a notebook running a mobile Pentium III 1000MHz processor and providing 256MB of random access memory. The whole robot's control architecture is developed in C++, the DTW libraries are integrated into the existing architecture and provided the same interfaces as our self implemented k-means-clustering algorithm.

We now made measurements of the time required by each algorithm to process the test data sets we introduced in 4.2.1. After every event, each of which has a length of forty data points, we let a pause[2] of 500ms and determined the time needed for processing the whole data stream.

	test set 1	**test set 2**	**test set 3**
dynamic time warping	5,9 (15,9) sec	3,0 (7,0) sec	4,3 (14,8) sec
k-means-clustering	6,2 (16,2) sec	3,2 (7,2) sec	4,5 (15,0) sec

Table 1. Comparison of two clustering algorithms (values in brackets are brutto time).

Looking at table 1 we can see that both algorithms provide nearly the same processing speeds, DTW being a little faster. So for further development we are giving preference to k-means-clustering for being quite as fast as DTW but outperforming it in usability.

An important thing to mention is that most of the pre-processing levels are unnecessary for DTW, while being mandatory for k-means-clustering to provide the same level of accuracy. But even when omitting these the DTW algorithm does not show a significant performance leap against k-means-clustering.

[2] The pause should substitute the time needed to gather the data (in real world conditions we plan to use a sampling frequency of 10Hz, being quite lower than in our simulation – 1000/500 . 40 = 80Hz).

5 Further work

Most algorithms for clustering time series work on univariate series, i.e. on signals provided by a single sensor (this is the level this report copes with). Usually robots have more than one sensor, a fact that gives need to some kind of multivariate clustering method.

Sensors in mobile robots mostly have different units and different ranges of values. An appropriate clustering algorithm should be able to handle this problem, or otherwise (and like in our case) the data needs standardization beforehand. Also not all sensors may be equally important in characterizing an specific event. E.g. when bumping head-first into a wall, all the sensor readings in the rear of the robot may not be of great importance. Therefore multivariate clustering should process only important sensor data and ignore erroneous and unnecessary sensor one.

The following two approaches may be distinguished:

5.1 Multivariate signatures

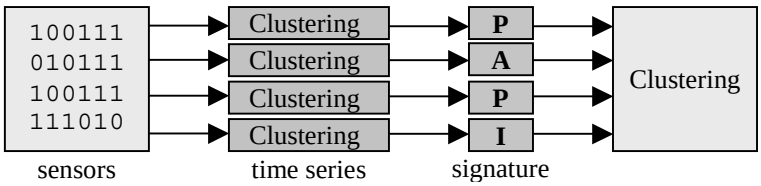


Fig. 26. Multivariate signatures – schematic illustration.

One solution is to cluster individual sensor data and to create multivariate signatures (prototypes). For each sensor a separate alphabet (code book) is created. This way, multivariate time series are transformed into a multivariate clustering problem. Now any static multivariate clustering method can be applied.

Only the signature of actually interesting sensors needs to be taken into account. In clustering signatures, weights can be used to compare only the signature of the sensors involved in the ongoing event. The weights are initiated by supervision (by hand) or by giving high weights to sensors which triggered the event, but more complicated methods to set these values are conceivable.

5.2 Multidimensional scaling

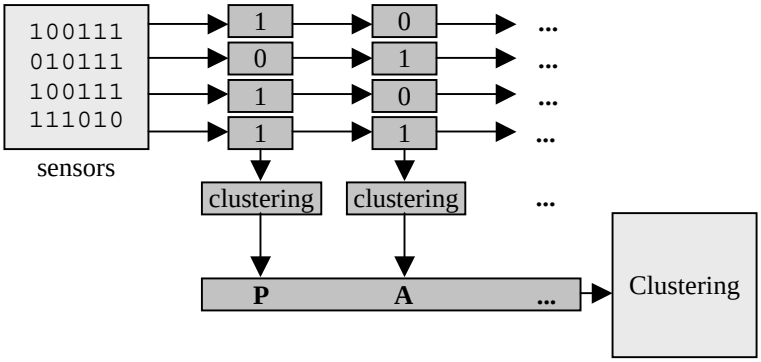


Fig. 27. Multidimensional scaling – schematic illustration.

The idea in this approach is to perform clustering on the whole set of signal samples at each point in time, leading to a time series of signatures (or cluster prototypes). Coincidentally, this approach shares many features with identifying time-variant event-related potentials in ongoing EEG, as exemplified by [Flexer *et al.*, 2001]. Like there, events the robot is to recognize can be viewed as trajectories of particular sensor configurations (akin to cognitive topologies in EEG). Clustering using k-means or Gaussian mixtures (which also permit the inclusion of particular noise models) appear especially appropriate. The signature string can then be used as the time series for the temporal clustering method. Interesting approaches for further investigation into this domain could be methods such as isomapping and, in particular, local linear embedding, which, after a first look, promise to be statistically sound approaches to reducing robot sensor data dimensionality.

Acknowledgements

This research is supported by the European Commission as IST project SIGNAL. Partners in this project are University of Bonn, Napier University, National Research Council Genova, and the Austrian Research Institute for Artificial Intelligence, which is also supported by the Austrian Federal Ministry for Education, Science, and Culture.

References

[Flexer *et al*., 2001] Flexer A., Bauer H., Lamm C., Dorffner G. – *Single Trial Estimation of Evoked Potentials Using Gaussian Mixture Models with Integrated Noise Component*. In Dorffner G., et al.(eds.), Artificial Neural Networks - ICANN 2001, International Conference, Vienna, Austria, Lecture Notes In Computer Science 2130, Springer, pp. 609-616, 2001.

[Intel, 1998] Intel® Recognition Primitives Library V4.0 http://www.intel.com/support/performancetools/libraries/rpl/index.htm

[Fu *et al.*, 2001] Fu, T., Chung, F., Ng, V. and Luk, R. – *Pattern Discovery from Stock Time Series Using Self-Organizing Maps*. Workshop Notes of KDD2001 Workshop on Temporal Data Mining, 26-29 Aug., San Francisco, pp.27-37, 2001.

[Keogh & Pazzani, 1998] Keogh E. J. and Pazzani M. J. – *An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback*. Department of Information and Computer Science. University of California, Irvine, California 92697, USA.

[Oates *et al.*, 2000] Oates T., Schmill M. & Cohen P. – *A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgments*. Computer Science Building, University of Massachusetts, Box 34610, Amherst, MA 01003-4610, USA.

[Poelz 2002] Poelz P. – *Eine Kontrollarchitektur fuer verhaltensbasierte Telerobotik*. Institut fuer Med.Kybernetik u. AI, Universitaet Wien, Diplomarbeit, 2002.

[Prem *et al.*, 2002] Prem E., Hörtnagl E., Dorffner G. – *Growing Event Memories for Autonomous Robots*. In Proceedings of the Workshop On Growing Artifacts That Live, Seventh International Conference on Simulation of Adaptive Behavior, Edinburgh, Scotland.

[Thiele, 2002] http://home.t-online.de/home/03742246535-0001/twiss.htm