Failure Detection and Classification for Industrial Robots

JÉRÉMY POUECH
Failure Detection and Classification for Industrial Robots

Master’s Thesis Report

JÉRÉMY POUECH
pouech@kth.se

Supervisor
John Folkesson
johnf@kth.se

Examiner
Danica Kragic
dani@kth.se

Stockholm 2015

Computer Science
School of Computer Science and Communication
Kungliga Tekniska Högskolan
Abstract

In industrial robotics, the detection of failures is a key part of the robots program to reach a robust assembly process. However, the setting up of such a detection is long, very specific to a robotic operation and involves programming by engineers. In response to this problematic, the thesis presented in this paper proposes an algorithm which makes it generic and semi-automatic the creation of a failures detection and a classification method. Based on machine learning, the algorithm is able to learn how to differentiate between a success and a failure scenario given a series of examples. Moreover, the proposed method makes the teaching of failure detection/classification accessible for any operator without any special programming skills.

After the programming of movements for a standard behavior, a training set of sensory acquisitions is recorded while the robot performs blindly operation cycles. Depending on sensors nature, the gathered signals can be binary inputs, images, sounds or other information measured by specific sensors (force, enlightening, temperature...). These signals contain specific patterns or signatures for success and failures. The set of training examples is then analyzed by the clustering algorithm OPTICS. The algorithm provides an intuitive representation based on similarities between signals which helps an operator to find the patterns to differentiate success and failure situations. The labels extracted from this analysis are thus taken into account to learn a classification function. This last function is able to classify efficiently a new signal between the success and failure cases encountered in the training period and then to provide a relevant feedback to the robot program. A recovery can then easily be defined by an operator to fix the flaw.
Acknowledgements

This research was supported by ABB Corporate Research in Västerås. I thank my colleagues who provided insight and expertise that greatly assisted the research.

I thank Ivan Lundberg and Shahbaz Khaber for assistance with machine learning, Daniel Sirkett and John Folkesson for their supervision, their ideas and support.

I would also like to show my gratitude to Carl Henrik Ek for sharing their pearls of wisdom with me during the course of this research.

Last but not least, I thank Anaïs Maroteaux for her help and support.
Contents

1 Introduction .......................................................... 1
   1.1 Background ....................................................... 1
   1.2 The search for robustness .................................... 2
   1.3 Goals and challenges .......................................... 3

2 Implementation framework ........................................ 5
   2.1 Formulation as a Machine Learning problem .......... 5
   2.2 Training set and Learning types ............................ 5
   2.3 Algorithm framework ........................................ 7

3 The clustering of the training data ............................. 9
   3.1 Related work on clustering ................................. 9
   3.2 Principle of OPTICS .......................................... 10
      3.2.1 Density-based clusters ................................ 10
      3.2.2 Density-based Ordering ............................... 12
      3.2.3 A clustering method with OPTICS ................. 20
   3.3 From hierarchical clustering to labeling .................. 22

4 The classification function .................................... 27
   4.1 Requirements .................................................. 27
   4.2 About semi-supervised multi-class classification .... 27
   4.3 Local Outlier Factor ......................................... 28
      4.3.1 The LOF score ......................................... 29
      4.3.2 Example in a 2d feature space ....................... 30

5 Application to a concrete robot operation .................... 35
   5.1 A robotic experiment ........................................ 35
      5.1.1 The YuMi robot ........................................ 35
      5.1.2 Operation ................................................ 36
      5.1.3 The different scenarios exploited ................. 37
      5.1.4 The experimental data ................................ 38
   5.2 Generic discrimination function for ................. 42
      multi-dimensional time-series ............................ 42
      5.2.1 Related work on time-series ......................... 42
5.2.2 Normalization ........................................ 44
5.2.3 The distance function ................................. 46
5.3 Clustering and training data labeling ............... 51
5.4 Performance of the learned classification function 53

6 Discussion and future work .............................. 59
  6.1 High-sensitivity to a change of shape ............... 59
    6.1.1 Observations ....................................... 59
    6.1.2 Simple trials of stabilization .................... 60
  6.2 Solutions proposals .................................... 61
  6.3 Similar subjects for further application in machine learning .. 62
Chapter 1

Introduction

The master’s thesis presented in this paper deals with issues encountered in industrial robotics. In this introduction, a first part concerns the general background about assembly line robots in the industry. Then some problematics linked to the installation and the programming of such robots are explained. Finally some goals are introduced so as to understand the current industrial interests in that field.

1.1 Background

Since their creation in the early 19th century, the assembly lines aim to increase the productivity in factories. To be as efficient as possible, the activity is divided in small and repetitive tasks performed by low-qualified workers. This way of organizing a factory has been proved to be effective and has allowed the beginning of massive productions lines. However, the gain of productivity came at the cost of bad working conditions. A job in a factory is limited to repetitive tasks that are often boring, tiring and stressful for a human. The job in factories has become degrading and unpleasant.

Nowadays, the newly-built factories tend to equip their assembly lines with machines and robotized agents. The robots are designed so as to help the workers in the most repetitive and tiresome actions. The future place of robots in assembly lines can be seen as a powerful tool for workers ([1, 2, 8]). Unlike the commonplace image of robots which can entirely substitute a human, the idea is to create intelligent agents able to co-operate hand in hand with human colleagues. Then, workers would become robot coaches. They would be able to teach operations without special education. The conditions of work in factories as well as the efficiency of the lines would be raised by this robotic revolution.

However, those objectives raise numerous technical problems. Currently,
it is difficult to communicate with an industrial robot. Most of them can only understand lines of codes written by programmers. Their integration in assembly line remains technical, takes a long time and is reserved for experts. The written program is often definitive and cannot be changed by an untrained laborer working next to it.

This thesis is part of a research work which aims to improve the usability of industrial robots by humans. The current researches have made attempts to automate the programmers’ work and to create simple interfaces for the future robot coaches. To ensure that every worker without special programming skills manages to teach a robot, the interfaces have to be simple and intuitive. Some recent works have shown it is possible to teach a robot by holding its arms and moving them as a puppet. The movements of the robotic arm are recorded and then are repeated in a loop in order to perform the taught operation ([2]). This robot teaching is an efficient method to automate one of the first parts of the work done by a robot programmer. Following this same idea, this report presents an efficient algorithm which makes an operator able, without writing any line of code, to modify the behavior of the robot in order to increase the robustness in the robot operations.

1.2 The search for robustness

A large part of the robot programmers’ job is to make the machine robust during its operations. The assembly task given to the robot has to be successful most of the time since the robot has to be as productive as possible. However, after the programming of a standard behavior, the robot is rarely fully successful in all the operations. Due to some irregularities of the pieces to assemble (different stiffness, previous malformation, etc.) or sensor measurement errors, the robot will have to deal with failures in the assembly. Those failures have to be detected and taken into account in the robot program. A flawed piece which continues its path in the assembly process can lead later to some bigger failures in the entire process, causing a significant efficiency loss in the assembly line afterwards.

To create a robust failure detection, the programmer has to study the robot operation and analyze the nature and the cause of the failures. Most of the time, he has to look deep down into the signals from the sensors integrated in the robot to find a way to produce a feedback for each assembly problem. Then, for each problem detected, the programmer can implement a recovery action. A recovery action can be either a new movement to fix a problem in the assembly or simply a rejection operation to eliminate the flawed object and give it to a more experienced human. As the tasks are most
of the time very simple, a basic movement can be programmed to correct the
failure. Then, the biggest part of this search for robustness is the detection
of a fault more than the programming of a recovery action.

1.3 Goals and challenges

To make the setting up easier and faster, the failure detection method and
the recovery actions have to be defined in a more simple way. The work done
by programmers has to be automated to allow non-programmers operators
to fully use and teach the robot. The robot teaching must be accessible for
an assembly line worker with little training: it has to be convenient and
intuitive for the user. This means the implemented algorithms must have as
few and understandable inputs as possible. The method has to be generic
and to work for every action possibly taught to the robot.

The goal of the presented work is to provide an automatic method to
define a failure detection. This feedback on the robot operation is based on
the information collected by the sensors during an operation cycle. Depending
on the nature of the sensor, the gathered signals can be binary inputs, images,
sounds or other information measured by specific sensors (force, enlightening,
temperature...). These stimuli contain specific patterns or signatures for
success and failure. Based on those patterns, the method has a double task:

• Failure detection
  The algorithm has to detect any deviation from standard successful
cases. A failure is detected if the sensor signal has an abnormal
behavior.

• Failure classification
  If a failure occurs, an associated recovery action has to be determined.
The recovery action is specific to the type of fault. Thus the algorithm
has to deal with a classification between the different cases of flaws.

The result of the failure detection and classification has to be released in
a short time after the end of the operation. As the process will be part of an
operation cycle, it must not slow down the frequency of the task too much.
The classification results have to be given after a negligible time compared
to the operation process or even during the process itself.
Chapter 2

Implementation framework

2.1 Formulation as a Machine Learning problem

The failure detection and classification problem defined in the introduction can be formulated as a machine learning problem. When a robot performs a cyclic operation, the recorded sensors observations are very repetitive over time. If a failure occurs, the sensors perception changes. One can think about an abnormal delivered motor torque, a different picture of the manufactured object, or an unexpected sound during the operation for instance. The general idea is then to learn the typical perceptions of the diverse situations.

From a set of sensor acquisitions examples, called training data, a machine learning algorithm can learn the differences between diverse scenarios of success and failed operation. To do so, it has to find typical patterns in the provided sensor feedbacks for the various scenarios. Using those patterns, a classification function can be generated to recognize a situation observed in the training set. Then, it would be possible to use the learned classification function on new observations in order to automatically detect which scenario occurred.

2.2 Training set and Learning types

To create the classification function, the training data has to be proceeded by a learning algorithm. In machine learning, there exist three different types of learning algorithm depending on the available knowledge about the training data.

- **Supervised learning**
  
  This case is the most popular in machine learning. With this type of learning, the training data set is labeled. It means that each object in the training set is given with a meaning, the desired class in output of the classification function. If the classification function has to make
the distinction between \( N \) different classes, the training data contains labeled examples for all the classes. Each new object presented to the classification function belongs to one of the classes represented in the training data.

This mode of learning would suppose that all the scenarios are known in advance and are shown in the training data. The resulting classification function would necessarily give as a result the most likely situation observed previously, even if the sensor signals do not look like any case of the training set. In our case, the training data will never represent exhaustively all the failures and all the possible situations. This kind of learning is thus not applicable.

- **Semi-supervised learning**
  The principle is the same as for the supervised learning. The training data is provided with labeled. The difference here is that the learning algorithm does not suppose that all the cases have been seen in the training data. The resulting classification function can differentiate between an object which has been represented in the training data and a new object which does not belong to any learned class. In that case, one talks about novelty detection.
  The semi-supervised learning is a good candidate for this application. The operator would have to assist the robot to discover the successful cases and maybe some failure scenarios. After the training, any novelty detected could be a new failure not discovered in the training data. It will have to be clarified by an operator or rejected by precaution. However, to have good results, the training data has to include a large amount of signals. If the operator wants to program a recovery action for a failure, the failure has to happen several times during the training period. In order to have the training data labeled, this implies that an operator would have to stay next to the robot when it is running and recording the training data. After each cycle, the operator would have to indicate the label of the scenario, that is to say if it was a success or a failure and the identification of the failure. This task would be long and very tedious for the operator. Moreover, with the tiredness or a lack of focus, an operator could make mistakes in the labeling. The training data could then contain some errors which would affect dramatically the efficiency of the classification function.

- **Unsupervised learning**
  In that case, the objects in the training data are given without any label. The algorithm has to find similarities between the objects to create a structure in the data set. One also talks about clustering. More
than defining a classification function, the goal of these algorithms is to find the recurrent patterns in the training data end to define groups of similar objects. This kind of learning can then be used to ease the acquisition of the labels. The operator does not have to assist the robot during the training period. The robot can run alone and collect data. Then, the clustering program identifies similarities between observations in the training data. A summary of the encountered situations can be shown to the operator. Using some understandable feedback representing the different identified clusters (for example sound/photo/video of a typical operation extracted from a cluster), the operator can label the data and analyze the different situations. These labels can then be used to learn a classification function with a semi-classification function as explained earlier.

2.3 Algorithm framework

The algorithm described further in this report combines both semi-supervised and unsupervised learning methods. The figure 2.1 illustrates the general chosen structure. The core of the algorithm is represented by the black dashes. The learning is then divided in two fully automatic phases represented by the red rectangles.

First, the training data is analyzed with a generic clustering method. The sensors acquisitions are sorted by similarity into clusters. After the automatic clustering, an operator has to assign a meaning to each cluster. This operation is illustrated by the purple rectangle. As the operator was not present during the training, a feedback can be done for example with a video recording to make the operator visualize the various clusters found. For each cluster, the operator can thus understand the failures in a few minutes. Then, he can associate each cluster to a meaning or a failure type which will later lead to a specific recovery action. After this phase, the training data is divided into several classes. The chapter 3 explains in detail the chosen implementation for the clustering algorithm.

So as to define a classification method between the different situations found during the clustering, a semi-supervised learning is used. This machine learning algorithm determines a classification function with both the training data and the associated labels. This classification function distinguishes between signals which have been encountered during the training phase but also any new type of signals. Moreover, unlike the clustering algorithm,
the result of this function has to be quick. Indeed, as explained in the introduction the classification of a new signal has to be performed at the end of each task. The robot must not be slowed down by this process. The implementation of the semi-supervised learning algorithm is detailed further in chapter 4.

Both clustering and learning phases rely on a discrimination method between the obtained signals. Depending on their nature, the sensor acquisition has to be compared by pair by a distance function. In the chapter 5 we will present a particular application of the algorithm for time sequences like the evolution of the torque in the robot joint over time. The design of a discrimination method to compare two time-signals is then explained. For other types of sensor acquisition, another discrimination function could be defined.
Chapter 3

The clustering of the training data

In order to ease the acquisition of training data labels by the operator, the research for similarities between sensor acquisitions can be automated. This operation is performed by a clustering algorithm which is presented in this chapter. In a first part, the literature on the subject is presented. Then the principle of the chosen algorithm is detailed. Finally we will explain how the hierarchical clustering can help an operator to label quickly the training data.

3.1 Related work on clustering

One of the most well-known clustering algorithms is called the k-mean ([17]). Its principle involves calculating k average signals which are the most representative of the data set. This calculation is done by an iterative method. Some variations of this algorithm exist such as the k-medoid ([13]) or the c fuzzy mean ([6]). Another algorithm named CURE ([12]) aims to find a hierarchical structure in the data. These algorithms can hardly be used in the presented application because one of their inputs is the number of desired clusters. In our case, the data is entirely unknown and the number of types of failure is difficult to predict.

Another approach to clustering is to look at the density of objects in a neighborhood of the k closest signals. Those methods similar to the KNN classification methods are explored in the literature as well. The principle is to create a cluster when the density of objects is high. For example, in a set of points in a 2d-space, a cluster would be created where the points are numerous in a delimited area. The most popular algorithm using this method is DBSCAN ([11]). This algorithm only needs two parameters to be tuned: $Minpts$ and $\epsilon$. It processes object by object the number of neigh-
boring objects separated by a distance lower than \( \epsilon \). If the neighborhood contains at least \( \text{Minpts} \) objects, the objects are part of the same cluster. The principle is simple however the algorithm is very sensitive to the choice of \( \epsilon \) and \( \text{Minpts} \). Changing the parameters can lead to totally different clusters. Then, they have to be determined very wisely depending on the data set. As the training data in our application is relatively small, the parameter for DBSCAN can hardly be tuned properly.

On the same idea of clustering by density, another method called OPTICS is very similar to DBSCAN (see [5]). It involves processing an order for the different objects in the training data so as to make a clustering operation easier. The article about OPTICS then proposes a clustering algorithm based on the preprocessed order. This last method has been chosen and implemented for our algorithm.

### 3.2 Principle of OPTICS

As mentioned before, the chosen algorithm is OPTICS and is described in the article [3]. The definition of clusters is similar to the one for DBSCAN but the implementation to find them is different. The algorithm provides an ordering of objects which allows a comprehensive representation of a complex set of data in 1D. The ordering makes it possible to find a hierarchical clustering, that is to say a complex structure containing several levels of clusters. A cluster can contain smaller clusters in its inside. The structure can be represented as a tree where each branch means a particular set of common points between the signals. The lower are the branches in the tree, the closer are the objects clustered.

#### 3.2.1 Density-based clusters

The methods OPTICS and DBSCAN are two density-based clustering algorithms. We present in this section necessary concepts to define mathematically a cluster with this clustering approach.

The general idea of this kind of clustering is to look at the distances between the objects of the data set. It supposes then that a discrimination method which gives a distance score between each pair of objects is available. Depending on the objects nature, this distance can be processed by various ways. If the object can be described with specific features, the distance function can be for example the Euclidean distance between the points representing the objects in the feature space. For other types of objects, it is easier to define a distance directly with the digital values of the object. The chapter [5.2] will give an example of research for an efficient distance function between time signals using the taken values over time. In order to illustrate
the concepts introduced in this chapter, each object will be represented with points in a 2-d space.

The density-based clustering methods are based on the number of neighboring objects of an object. One will remark that the notion of neighborhood depends on the chosen distance function. The concepts are based on two parameters, a positive real value $\epsilon$ and an integer $MinPts$. The $\epsilon$ neighborhood of an object $\omega$ is the set $N_{\epsilon}(\omega)$ which contains all the objects at a distance less than $\epsilon$ from $\omega$. Based on this first concept, one can define a core-object, the direct density-reachability, the density reachability, and then the density connectivity.

**Core-object**

A core-object is an object which has at least $MinPts$ in its $\epsilon$ neighborhood.

\[
\omega \text{ is core-object } \Leftrightarrow \text{Card}(N_{\epsilon}(\omega)) \geq MinPts
\]  

**(3.1)**

**Direct density-reachability**

An object $\omega$ is directly density-reachable from an object $\nu$ if $\nu$ is a core-object and $\omega$ is in the $\epsilon$ neighborhood of $\nu$.

\[
\omega \text{ is directly density-reachable from } \nu \Leftrightarrow \begin{cases} 
\text{Card}(N_{\epsilon}(\nu)) \geq MinPts \\
\omega \in N_{\epsilon}(\nu)
\end{cases}
\]  

**(3.2)**

**Density-reachability**

An object $\omega$ is density-reachable from an object $\nu$ if it exists a chain of objects $\{\omega_i\}_{i=1}^{n}$ which fulfills the conditions:

\[
\begin{cases} 
\forall i \leq n - 1, \omega_{i+1} \text{ is directly density-reachable from } \omega_i \\
\omega_1 = \nu \\
\omega_n = \omega
\end{cases}
\]  

**(3.3)**

**Density-connectivity**

Two objects $\omega_1$ and $\omega_2$ are density connected if there exists at least one point $\nu$ such as $\omega_1$ and $\omega_2$ are both density-reachability from $\nu$.

These last definitions allow a practical definition for a cluster. A cluster $C$ is a set of objects for which all the pairs of objects are density-connected. $C$ fulfills as well a maximality property which says that if $\nu$ belongs to $C$ and $\omega$ is density-reachable from $\nu$, then $\omega$ is also part of $C$.

For a better representation of these concepts, the figure [3.1] shows the intuitive equivalent in a two dimensional feature space. Each point of the graph represents, in this case, an object. The generic distance used to
3.2.2 Density-based Ordering

Definitions of notions

The algorithm DBSCAN consists in finding the clusters as defined previously for the given parameters $\text{MinPts}$ and $\epsilon$. However, as mentioned before, the algorithm is very sensitive to a change of parameters, in particular for the parameter $\epsilon$. This parameter can be seen as the maximum tolerated distance between the objects in a cluster. A decrease of $\epsilon$ leads to smaller clusters in which the objects are closer to another and so have more similarities. On the contrary, if $\epsilon$ is increased then the clusters are wider and include further objects. The idea of the algorithm OPTICS is to find a representation of the objects so as to find all the levels of similarities corresponding to different values of $\epsilon$.

The sorting method OPTICS is based on two distances: the core-distance and the reachability-distance which are defined as follows:

Core-distance

The core-distance is calculated for each object and represents how far the neighboring objects are from this first object. Let $\omega$ be an object in the database $\Omega$. The core-distance $CD(\omega)$ is equal to the distance of the $\text{Minpts}^{th}$ nearest neighbors called as well $\text{Minpts}$-distance($\omega$):

$$CD(\omega) = \text{Minpts-distance}(\omega)$$  \hspace{1cm} (3.4)

This core-distance corresponds to the minimum value of $\epsilon$ such that an object can be considered as a core-object and thus generate a cluster.

Reachability-distance

The reachability-distance is an asymmetric distance between two objects. Let $d(\omega, \nu)$ the chosen distance to differentiate between $\omega$ and $\nu$. The
reachability-distance $RD(\omega)$ is defined as:

$$RD(\omega, \nu) = \max(CD(\nu), d(\omega, \nu))$$

(3.5)

This distance corresponds to the minimum value of $\epsilon$ for which $\omega$ is directly density-reachable from $\nu$. That is to say, if $\epsilon \geq RD(\omega, \nu)$, $\nu$ is a core-object and $\omega$ is in the $\epsilon$-neighborhood of $\nu$.

These definitions found in the article [5] can be restricted. The neighborhood query in large databases can be restrained to a maximum distance. This limitation is not very important in our use case because the data base used is relatively small and the execution time is not very limiting. The definition presented here is the extension of the definition in [5] when this maximum distance tends to $+\infty$.

The notion of core-distance and reachability-distance are illustrated in the figure 3.2 for a few objects in a 2d-space. In this graph, $MinPts$ is equal to 3. The Core-distance of the point $o$ is represented by the inner circle whose radius is equal to the distance of the third neighbor of $o$. The Reachability-distance of $p_1$ from $o$ is equal to the Core-distance of $o$ since $p_1$ is in the neighborhood of $o$. $p_2$ is not in the neighborhood of $o$, so the Reachability-distance of $p_2$ from $o$ is equal to the distance between the two points. The distance $M$ on the figure is the maximum distance that a point can have from $o$ to be in the neighborhood of $o$. In the following, we will fix this maximum value $M$ to infinity.

**The Ordering Algorithm**

The goal of the sorting is to create an order of the objects which represents their proximity and their capacity to create clusters according to the definition given earlier. During the ordering, the algorithm treats the objects one by one and places them in the ordered list by smallest reachability distance from previously processed points. The algorithm follows the steps as below:

**Initialization**

First, a temporary list is created containing all the objects to order. This list will be called $seeds\_list$. Aside, a value is associated to every object. This value will represent the minimum reachability distances $RD*$ of the point from all the points already placed in the sorted list. These values are initialized to infinity and updated during the ordering with smaller values encountered. Finally, an empty list $ordered\_list$ is created. This list will contain the sorted objects.
Iteration

The element corresponding to the smallest reachability distance $RD^*$ is removed from the seeds_list and placed in ordered_list. Note that for the first iteration, one element will be picked randomly. The reachability distances of all the elements in the seeds_list from the picked object are processed. The reachability-distance $RD^*$ of the objects still in the seeds_list is updated if the processed value is smaller.

End of the process

The algorithm ends when all the elements in the seeds_list have been processed.

The figures 3.3, 3.4, 3.5 and 3.6 illustrate different iterations of the algorithm for a simple example with 2d-points. At first the points are placed in the seeds_list randomly. The rectangles in the figures 3.3b, 3.4b, 3.5b and 3.6b represent the values of the reachability distances from previous points. The order of the points is the initial random order.

The results of the first iteration are illustrated on the figure 3.3. One point (represented by a circle) is picked in the seeds_list. Here, the first point has been picked. All the reachability distances are updated on the figure 3.3b. Previously infinity, the values are now finite except the first elements which will never be updated. A first blue point has been placed in the graph on the figure 3.3c.

After around ten iterations, the values of $RD^*$ decrease progressively as...
shown in the figure 3.4b. The black rectangles represent the values of \( RD^* \) for the elements already placed in the list. Those values will not change in the next iterations. The figure 3.4c shows the succession of \( RD^* \) values in the processed order. On the figure 3.4a, one can remark that two consecutive points in the processed order are not necessarily close. For this step, the current point and the next point are far from each other. However, the selected point is the nearest of all the processed points in terms of reachability distance.

After thirty steps, the algorithm has reached a zone with a lot of close points which could create a cluster. As the points are close, the reachability distances between each pair of points are small and so the points are all processed in a raw. One can see on the figure 3.5c that the values of \( RD^* \) decrease a lot and are stable around a low value.

Finally, the figure 3.6 shows the results of the ordering. One can see that the further points from the cluster have reachability distances \( RD^* \) greater than the values obtained in the cluster.
Figure 3.3: OPTICS: An example with 2D-points (1)

(a) Points set (black circle: current picked point; red plus: next chosen point; black x: points in \textit{ordered\_list}; black dot: points in \textit{seeds\_list}; orange lines: link between the point and its closest point in the \textit{ordered\_list} when the point was picked up)

(b) Reachability distance associated to each point. Initial point order. (black rectangle: processed point; red rectangle: next processed point; white rectangles: other point)

(c) Reachability distance associated to each processed point. Final point order.
Figure 3.4: OPTICS: An example with 2D-points (2)

(a) Points set (black circle: current picked point; red plus: next chosen point; black x: points in ordered_list; black dot: points in seeds_list; orange lines: link between the point and its closest point in the ordered_list when the point was picked up)

(b) Reachability distance associated to each points. Initial point order. (black rectangle: processed point; red rectangle: next processed point; white rectangles: other point)

(c) Reachability distance associated to each processed points. Final point order.
Figure 3.5: OPTICS: An example with 2D-points (3)

(a) Points set (black circle: current picked point; red plus: next chosen point; black x: points in ordered_list; black dot: points in seeds_list; orange lines: link between the point and its closest point in the ordered_list when the point was picked up)

(b) Reachability distance associated to each points. Initial point order. (black rectangle: processed point; red rectangle: next processed point; white rectangles: other point)

(c) Reachability distance associated to each processed points. Final point order.
Figure 3.6: OPTICS: An example with 2D-points (4)

(a) Points set (black circle: current picked point; red plus: next chosen point; black x: points in ordered_list; black dot: points in seeds_list; orange lines: link between the point and its closest point in the ordered_list when the point was picked up)

(b) Reachability distance associated to each points. Initial point order. (black rectangle: processed point; red rectangle: next processed point; white rectangles: other point)

(c) Reachability distance associated to each processed points. Final point order.

19
3.2.3 A clustering method with OPTICS

Intuitively, when a sequence of $RD*$ values are locally small, the corresponding objects are close to each other and can be part of the same cluster. A rising slope means a higher distance between all the previous points and the next ones and so the end of a cluster. A decreasing slope shows the beginning of a new cluster. In the article [5], a method based on the values of $RD*$ and $CD$ is presented in order to find a DBSCAN equivalent clustering. The parameter $\epsilon$ in DBSCAN can be seen as a threshold for the $RD*$ curves. A segment of consecutive indexes which have $RD*$ values lower than $\epsilon$ belongs to the same cluster. The first element of a cluster in the ordered list does not have necessarily a low value $RD*$. Indeed, as the first element of a cluster is reached by a further point. So, the beginning of a cluster is detected by a Core-Distance $CD$ lower than the threshold $\epsilon$. By this method the obtained clusters are almost identical to those found for DBSCAN. Only some border points of the clusters would be missed.

In order to find a hierarchical structure in the database, one can think
about a multi-thresholding method. By this way, it is possible to divide the data in a succession of clusters which correspond to different values of $\epsilon$. The figure 3.8 shows all the thresholds which lead to all the possible division of the data into clusters. Each black segment of the figure represent one cluster. The height of the segment is the applied threshold to find the cluster. The segment starts at $x = x_0$ and stops at $x = x_1$. The corresponding cluster gathers then all the signals which have an index between $x_0$ and $x_1$. One can notice that the clusters corresponding to a threshold $\epsilon$ are fully included in the clusters gathering at least one common object and with a threshold $\epsilon' > \epsilon$. The representation by a tree is then perfectly justified.

The figure 3.9 represents the tree built with the method described earlier. Each horizontal rectangle represents a cluster. Each small edge corresponds to a limit of the cluster. The consecutive indexes between those two limits are part of the same cluster. One can observe the cluster on the level 0 is divided in two distinct clusters on the level $-1$. The signals from the indexes between 1 and 10 belong to one cluster and the signals with the indexes between 11 and 50 to another cluster.
3.3 From hierarchical clustering to labeling

The main goal of the clustering algorithm is to help an operator to gather labels for the training data of the classification method. After the clustering, the relevant data has to be selected. When the operator comes after the training period to define the recovery behaviors, it will have to give a meaning to each of the clusters. Actually, only a few branches of the tree created earlier correspond to real observed scenarios. A lot of branches correspond only to small variations in the acquisition of the same scenario. To get then the largest training data for the classification function, we need then to pick up the biggest clusters gathering each only one scenario.

From the representation of the data created by OPTICS, it is possible to build an intelligent software to select the relevant training data with the help of the operator. The software can show the different situations corresponding to local minima with video feedback. The goal of this software would be to find the highest cluster in the tree which represents the same scenario. By starting from the leaves, the software can show to the operator the images corresponding to the signals with the highest core-distances which correspond to the most critical cases of the cluster. If all the clusters corresponding to the same subtree represent the same scenario the upper cluster can be tested. If the upper cluster corresponds again to the same cluster, then all the lower
branches can be removed from the tree. The operation can be repeated with all the leaves until no more simplification is possible. The different scenarios found will be represented by the clusters on the leaves of the simplified tree structure.

The reader has to note this software is a proposal for a future development. The implementation of such a feedback loop requires important settings and is specific to the application and to the nature of the signal used. We will assume in this report the operator was supported by such a software and was able to make the best selection of training data.

The figure 3.10 shows a typical selection of 2 clusters for our example case. The threshold for the first cluster is bigger than the one for the second cluster. That means that the points are closer in the second cluster than in the first one. One can observe this effect in the figure 3.11. The remaining points which does not belong to one of the clusters are removed from the training data. With some real data, those points would correspond to noisy data or acquisition of a new kind of error not enough represented in the training data. This type of failure would be detected in the following but not classified.
Figure 3.10: Labels selected by the operator assisted with a software (Blue for $RD^*$, red for $CD$, black for segment of indexes belonging to the same cluster). The clusters can be compared to the scenario represented by dots of different colors.
Figure 3.11: Points set with clusters
(red+: first cluster; blue stars: second cluster; black dots: unselected points
orange lines: link between the point and its closest point in the
ordered_list when the point was picked up)
Chapter 4

The classification function

After the acquisition of labels by the clustering algorithm and the interaction of the operator, a classification function can be processed by the algorithm to automatically detect and classify the failures without extra intervention from the operator. This part explains then the classification and the learning method used for the failure detection and classification algorithm.

4.1 Requirements

The proposed classification is created with the data provided after the clustering and the acquisition of labels. Only the data corresponding to the selected clusters is kept. All the signals considered as noise and too far from one of the scenarios found are removed. However, even if a set of failures might have been recorded, a new type of failure could still occur during the process. The algorithm has to deal with novelty. Moreover, the algorithm can have to classify between more than two classes of signal. The problem given here is thus a semi-supervised multi-class machine learning problem.

The detection of a failure for a new signal has to be online. The detection is part of the real-time process. The response of the algorithm has to be quick enough to be integrated in the behavior loop. The classification has to be as accurate as possible in order to avoid some misbehaviors. Finally, the amount of training data obtained after the clustering is rather small since the training period must be quite short.

4.2 About semi-supervised multi-class classification

As explained in the article [4], some algorithms used for a two-class classification can be extended for a multi-class classification. These methods include among others neural networks, naive Bayes, K-Nearest Neighbor,
and support vector machines. However, these algorithms can only classify between the known classes and cannot detect a novelty.

Another solution is to decompose the problem of multiclass classification into several binary classification tasks. It exists then two main strategies:

- **Strategy One-vs-One**
  The principle is to define classification functions which compare pairs of classes. Those classification functions assign a winner class for each pair. A final vote is done taking into account the winner of each dual. The class attributed is the one which corresponds to the highest number of votes.

- **Strategy One-vs-All**
  A classification function is created for each class. For each class, the classification functions answer 1 if the new object belong to their class. To avoid several classes winning, a score is assigned to each class. The final classification is done by comparing the scores obtained for each class and determining the class with the best score.

The strategy one-vs-one is not possible in our case because it is not possible to detect a possible new failure in a signal. The best strategy is the one-vs-all because this novelty detection can simply be done by a thresholding on the score. A too poor score would thus correspond to an outlier. The recorded signal does not correspond to any known class.

The most standard solution for classification of several classes with the one-vs-all strategy is the one-class SVM. This solution is however based on features and representation of the signals in a multi-dimensional space. As explained in the article [13], the algorithm finds a border which defines the best the class using Support Vectors implementation. However, the SVM methods requires a feature description of the objects. One of the most frequent use case of the algorithm will have to deal with time-series. The extraction of features being difficult for time-series, this particular method and all method using feature extraction did not seem suitable for the study case.

### 4.3 Local Outlier Factor

The Local Outlier Factor method of (L.O.F) is based on the same principle as the clustering algorithm used previously. The algorithm uses the density of the points in the data base to calculate a score (see [7]). Only a distance function is thus needed to discriminate and classify between the cases. The
score calculated for each cluster indicates if the new signal is close to the other point and have the same density around it. A new signal is classified as an outlier for one cluster if the score value obtained is too low.

4.3.1 The LOF score

The LOF score is also calculated with the reachability distance between objects as defined earlier. However, the notion of neighborhood is changed. The $MinPts$-neighborhood of an object $\omega$ is the set of the $MinPts$ nearest neighbors. The main idea is to compare the reachability distances of the new object from its neighbors with the reachability distance of the neighbors themselves from the neighbors of their neighbors. In other words, the distances between the new object and the object in the cluster has to be equivalent to the distances of the objects inside the cluster in order to consider the new object belongs to the cluster.

To express this idea mathematically, let $\rho$ be the neighborhood density of the object $\omega$ defined as:

$$\rho(\omega) = \frac{MinPts}{\sum_{\nu \in N(\omega)} RD(\omega, \nu)}$$

This value represents the proximity of the points close to $\omega$. The LOF gives a comparison between proximity of a new point and proximity of the neighbors of the new point. Mathematically, it can be expressed as:

$$L(\omega) = \frac{\sum_{\nu \in N(\omega)} \rho(\nu)}{\rho(\omega) |N(\omega)|}$$

There are three possible situations given the value of the local outlier factor.

- If the score is less than 1, the new point is literally inside the cluster. Its neighbors are closer to this new point than to their respective neighbors.

- If the score is close to 1, it means the new point is near the cluster, probably on a border but at a comparable distance to the neighbors between them.

- Otherwise, if the score is greater than 1, the new point is far from the other points of the cluster and cannot be associated to the same cluster.

This classifier is a one-class classifier. The maximum of the LOF score for each cluster is calculated. The cluster with the highest score is the best
candidate. By using a threshold $\theta$ on the score, the best candidate can be eliminated if the score has a value that is too low (value under $\theta$). The new signal is then associated to noise or unknown.

The algorithm has then two parameters, $MinPts$ and the threshold $\theta$. The first parameter $MinPts$ represents the number of neighbors around the objects. It can be fixed empirically by default as previously. The other parameter $\theta$ has a bigger impact on the results of the algorithm. However, it does not depend on the data classified since it is a ratio of distances. Its value can be fixed to a default value. Moreover, this threshold can as well be modified by the operator. It corresponds directly to the sensitivity of the classification function to a new type of error.

### 4.3.2 Example in a 2d feature space

To understand better the LOF classification method, the clusters found in the previous chapter were taken and used to learn a classification function in a 2d-space. The learning phase consists then to learn the density of each of the points in the cluster. These densities $\rho$ will be used during the classification of new points. Their pre-processing is useful to increase the rapidity of the results.

For a new point, the classification finds the $MinPts$ closest points for each cluster. Then it processes the density of points from one cluster around the new point if it was belonging to the same cluster. Finally, it compares the new point density and its neighbors’ density calculating the LOF score for each cluster. The new point is classified similar to the cluster with the lowest score. If the lowest score is greater than $\theta$, the point is classified as unknown.

The figures 4.1, 4.2 and 4.3 represent the results of the classification function for a grid of points in the space for $MinPts = 3$ different values of $\theta$. One can notice that a change of $\theta$ modifies only the boundary between known and unknown.
Figure 4.1: Results of the LOF classification function on a grid of elements
\((\theta = 1.5)\)
(red+: first cluster; blue stars: second cluster;
black dots: points classified as unknown;
red x: points classified as similar to the first cluster;
blue lozenges: points classified as similar to the second cluster)
Figure 4.2: Results of the LOF classification function on a grid of elements 
\((\theta = 2)\)

(red+: first cluster; blue stars: second cluster;
black dots: points classified as unknown;
red x: points classified as similar to the first cluster;
blue lozenges: points classified as similar to the second cluster)
Figure 4.3: Results of the LOF classification function on a grid of elements ($\theta = 3$)
(red+: first cluster; blue stars: second cluster;
black dots: points classified as unknown;
red x: points classified as similar to the first cluster;
blue lozenges: points classified as similar to the second cluster)
Chapter 5

Application to a concrete robot operation

5.1 A robotic experiment

For evaluation purposes, the algorithm has been tested with realistic experiments on a robot. This section is dedicated to the description of the tools and manipulations used to collect the so-called experimental training data.

5.1.1 The YuMi robot

The experiments have been performed on the Yumi robot (see picture on figure 5.1). This ABB product is a robotic chest which is fixed on a support like a table and equipped with two automated arms. Each arm has seven degrees of freedom. A gripper is assembled at the end of each arm in order to take and manipulate manufactured objects. The robot is human size and is designed so as to collaborate with a human co-worker. In this study, for simplicity reasons, only one of the two arms has been used.

The robotic arm is equipped with a minimum set of sensors for economic reasons. The idea behind the concept is to deduce a maximum amount of information from a minimum price invested in integration of sensors. The position of each arm is available from resolvers mounted on the motors. Then the velocity can easily be deduced by derivation. The currents running through each motor are measured by cheap and easily integrated sensors. From those intensity signals, a software force sensor has been implemented. Using models, it calculates the torque applied by each motor in real time.
5.1.2 Operation

To test the algorithm, the robot has been programmed to perform a simple operation inspired by one of the Yumi’s demonstration programs. The demo operation consists in assembling two plastic pieces used for an emergency stop press button. The figure 5.2 illustrates the task of the robot during the demo. The robot picks the two pieces, one white and one grey, originally placed in feeders around the robot (on the left and right of the robot on the figure 5.1). It uses both its hands to insert the two legs of the white part into holes in the grey part as shown in the picture 5.2b. Finally, holding the grey part in its first hand, it pushes the white into the grey part so as to click it in. The picture 5.2c shows this last state. As the final operation requires an important force for the robot, the arm takes a run-up and punches the white plastic piece.

For simplification purposes, this assembly has been simplified so as to require only one arm of the robot. A 3D-printed support has been designed in order to hold the grey piece in a vertical position as shown in the picture 5.3. The support is fixed on the robot table. The simplified operation focuses more on the punching phase. The white and grey parts are settled manually in the support in the situation of the state 2 like in the picture 5.2b. Then the robot only has to come above and make a tap on the white piece to finish the assembly. The program of the robot is really simple since it is only
movements from a position $A$ to a position $B$ and could be done by hand with the interface provided by ABB.

5.1.3 The different scenarios exploited

The assembly of emergency buttons is a well-known demo operation for Yumi. Many works have been done to improve the robustness. A set of particular failures has been reported and some recovery behaviors have been programmed to reach now a good efficiency in the shows. The simplified operation described just before focuses on the most critical part of the robot program. The punching phase is the moment where most of the failures happen or where it is the easiest to detect a previous anomaly.

The program used in the experiment does not take into account the improvements developed for the demo. The goal here is to see if the algorithm is able to detect and recognize some failure situations by itself without a specific code supplied. That is why the experimental data was generated so as to include success and failures cases. However, as the task was simplified and involved only one arm, the amount of failures in the experimental operation was very small. To increase the number of failures in the experimental database, the failure signals have been generated by changing some oper-
ating conditions but still trying to imitate failures encountered during the programming of the demonstration.

As shown in the figure 5.4, the experimental dataset gathers four different scenarios:

- A success case (figure 5.4a), the two parts have correctly been assembled.

- A first failure case (figure 5.4b) where the two parts cannot be assembled because something is blocking. This failure happened during demo. Often it is due to another white piece which has been gripped and has come astride the two legs of the first one. This can happen as well when the white piece is very stiff. The force applied on it was not strong enough to finish the assembly. This failure type has been generated by setting a thin object between the two plastic pieces.

- A second failure case (figure 5.4c) where either the grasping of the white part has failed or the white part dropped during a robot movement. The white part is not in place before the punch. The robot moves then in the air without touching anything.

- And a last failure case (figure 5.4d). Here, only one of the two legs of the white piece has been well placed in the hole of darker part. After the robot performs the punch, the final object is misassembled.

5.1.4 The experimental data

The figure 5.5 and 5.6 shows some samples of signals extracted during the process. The data has been labeled by hand so as to evaluate the performances. For display reasons only 4 out of 7 axes are shown. The four colors are used to separate the different scenarios as explained in the caption. The subfigure 5.5 shows the signals evolutions for the whole cycle of operation whereas the subfigure 5.6 zooms on the critical area which allows to discriminate between the different cases.
One can observe on these curves that during a free movement of the robot (between $t = 0.2s$ and $t = 0.6s$ and between $t = 1s$ and $t = 1.6s$) the curves are close to each other and repetitive, in every scenario. A significant deviation can be observed between $t = 0.6s$ and $t = 1s$ according to the scenario. This window of time corresponds to the punch. In this window, one can see that a shape of signal is associated to a case and can easily differentiate between the situations. A punch in thin air and a success case have two clearly different patterns for example. However, when the robot is idle between $t = 0s$ and $t = 0.1s$, we can notice a big variance between the signals for any scenario. This variance is due to imperfections of the friction force estimation for low speed. These aspects will be developed further in this chapter.

The data obtained by the experiment is divided in two sets. The first one hundred signals will be considered as training data. This data is used to learn the different scenarios encountered. The remaining of the data will be used further to evaluate the performance of the classification function learned. As the data is labeled by hand, the answers of the function and the true scenario can be compared.
Figure 5.5: Torque signal obtained for the axis 2, 3, 5 and 7
Entire operation, Labels extracted by hand
Blue: success, Green: blocking, Cyan: beating the air, Red: one leg out
Figure 5.6: Torque signal obtained for the axis 2, 3, 5 and 7. Zoom on the punching phase ($0.6 < t < 1$), Labels extracted by hand: Blue: success, Green: blocking, Cyan: beating the air, Red: one leg out.
5.2 Generic discrimination function for multi-dimensional time-series

The objects compared by the algorithm are sequences of values mathematically called time-series. Each sensor gives a temporal signal which is windowed during a critical operation. The information consists of several synchronous sequences of values from the different sensors. This constitutes a multidimensional time-series. In order to find a structure or patterns in the data bases, several techniques are used to distinguish different objects. This chapter is dedicated to the research of an efficient and generic discrimination method for time-series and multidimensional time-series.

5.2.1 Related work on time-series

Feature extraction

Many works about machine learning in the literature use feature extraction to find patterns. A feature is commonly a numerical value which represents a property of an object. Usually, a feature extraction function can be found to transform the object into the feature space. Let \( \omega \) be an object in a set of objects \( \Omega \). One can write the feature extraction function \( g \) as:

\[
\Omega \rightarrow \mathbb{R}^d
\]

\[
g : \omega \mapsto x
\]  

(5.1)

where \( x \) is called the feature vector associated to the object \( \omega \).

The use of features allow a vectorial representation of an object in a d-space, d standing here for the number of features describing the object. One can thus define borders to create areas in the space corresponding to one class of object. Intuitively, the learning algorithm creates the borders in the d-space. Then, the classification function retrieves the domain in which is the new object is represented by a point.

A large number of studies about pattern recognition for time series have as well exploited simple features. A common idea is to transform the time series in the frequency domain by calculating the Discrete Fourier Transform or to use the Discrete Wavelet Transform. The articles [3] and [18] use this idea to extract features and classify signals. However to keep the essence of the information given in the sequence, a lot of frequencies must be kept. The main problem of such feature extractions is the curse of dimensionality. As the number of dimensions used to describe an object is high, a large training set is required to learn efficiently the complex data structure. Some methods like PCA or other features selection can thus be used to reduce the
dimensionality of the problem in some situations (see [20]). However, those techniques can hardly be used for novelty detection. By choosing a mixture of features, a novelty in neglected features could be missed.

Some techniques for feature extraction developed in [9] among others, consist in finding an ARMA model which fits the behavior of the signal. The coefficient of the model can be learned and compared. Those features would describe the signal in a space with far fewer dimensions. Nevertheless, this approach is more adapted to learn a behavior than a shape of signal. In our application we are more interested in a temporal change than a general behavior.

The thesis [19] gives a generic method to extract features from time signals which describe shapes. Given a set of signals, it divides the time into segments so that each segment can be represented by one simple behavior. It can be linear, exponential, triangle, and other shapes. Each feature represents one parameter of the behavior in a segment. This approach could have been explored. However, the procedure to calculate the feature seems heavy and relies on a labeled dataset.

**Similarity join**

For some kinds of objects like the time-series, it is difficult to define proper features and to represent them into a d-space. However, it is sometimes easier to create a similarity function. This function retrieves a positive number, classically a distance, which evaluates the similarity between two objects. Let \( \omega_1 \) and \( \omega_2 \) be two objects in \( \Omega \). A similarity or distance function \( f \) can be expressed as:

\[
\Omega \times \Omega \rightarrow \mathbb{R}^+
\]

\[ f : \omega_1, \omega_2 \mapsto s \quad (5.2) \]

where \( s \) is a score evaluating the similarity between \( \omega_1 \) and \( \omega_2 \). This similarity function is reflexive and positive.

Using the distances between the training data objects and the new unclassified object, the classification functions find which class of objects the new object is the most similar to.

A lot of methods used to classify time-series are based on similarity measurement. Two ubiquitous methods are the Euclidian distance and the Dynamic Time Warping (or DTW). It consists in evaluating how far the signals values are in the time-domain. These methods have been explored and compared in the articles [15, 10, 13] among others. Those distances allow a comparison of shape between signals by simple mathematical operation.
They are defined for unidimensional time series but it is possible to transpose the method for multidimensional time-series using operations on vectors at each time instant.

The popular DTW method aims to calculate distances between non-synchronized signals. It finds similarities of shape even if the signals are shifted in time. It matches the time instants by minimizing the sum of the distances between each match. This matching can be useful when the sequences given are not synchronized or do not have the same length. For our application, the operations are periodic. The signals extracted from the sensor can be synchronized with the position control signals which are perfectly repeatable. Moreover, the matching method would erase possible temporal shifts in the signals that could need to be detected.

In this thesis, the similarity query has been exploited. However, the Euclidian distance had a too poor discrimination power to be used. To find the better distance function, a comparison has been made with several candidates. (see section 5.2.3)

5.2.2 Normalization

Scaling normalization

As explained above, the discrimination method is based on distances between the values of the signals. However, in the use case, seven time-series are concerned which have different scales. This effect can happen in every use case of the algorithm. Sometimes even the unit between the signals could change. One can think about comparing two signals of different nature like a force and a pressure for example. Two robot operations could be considered as similar if the shapes of the sequences are similar. A change of shape in any of the signal has to have the same impact regardless to the scale of the signals. Each sequence must have the same weight on the similarity measurement.

To balance each signal of one multidimensional time-series, the sensor signals can be normalized. In this study, it is assumed that the signals are varying and have repetitive waves. The experimental signals show that for each axis the torque calculated has a significant amplitude. The method adopted here is normalizing the amplitude of each signal. To do that, the amplitude is calculated with the variance over the time of the signal as written in the equation \(5.3\) where \(x_k^l\) is the signal extracted from the sensor \(k\) at the time \(l\). One can normalize the signal by dividing them by their amplitude as in the equation \(5.4\).

\[
\forall k, \text{Amp}(k) = \text{var}(x_k^l) \\
\text{Equation 5.3}
\]

\[
\forall k, x_k^l / \text{Amp}(k) \\
\text{Equation 5.4}
\]
\[ \forall k, \forall l, \hat{x}_l^k = \frac{x_l^k}{Amp(k)} \] (5.4)

**Speed pruning normalization**

As one could observe in the figure 5.5, the torque measurement has a poor resolution when the speed is low. This lack of accuracy is due to an error in the estimation of the force. To decrease the impact of this effect, the similarity query must be applied to portions of the signals where the speed is significant. In addition, one can suppose the failures would happen when the robot does not move.

To address this problem, the difference of the signals can be weighted according to the speed of each arm. Consider the Euclidean difference. For each axis, the values of the differences between torques corresponding to a low speed has to be pruned so as to reduce the measurement error impact in the similarity measurement. For the following, the notation \[ x_l \] means the multi-dimensional time-series value for the instant \( l \). This value is a column vector:

\[
x_l = \begin{pmatrix} x_1^l \\ x_2^l \\ \vdots \\ x_K^l \end{pmatrix}
\] (5.5)

Let \( \mathbf{v}_k^l \) the speed of the motor on axis \( k \) at the instant \( l \) corresponding to the torque time-series \( x \). Mathematically, the pruning operation corresponds then to the equation (5.6) where \( w(\mathbf{v}_k^l) \) is the pruning factor for the instant \( l \), axis \( k \) of the time-series \( x \)

\[
d_{Eucl} = \sum_{l=1}^{L} \sqrt{\sum_{k=1}^{K} (w(\mathbf{v}_k^l) \times x_l^k - w(\mathbf{v}_k^l) \times \mathbf{v}_k^l)^2} \] (5.6)

Let \( W^x \) be the diagonal matrix \( K \times K \) composed with the values of \( w(\mathbf{v}_k^l) \). The equation (5.6) can easily be reformulated as in the equation (5.7)

\[
d_{Eucl} = \sum_{l=1}^{L} \| W^x_l x_l^k - W^y_l \|_E \] (5.7)

All the difference functions proposed below in the section 5.2.3 are also based on a transformation of the vectorial Euclidian norm of differences between two signals at each instant. Let \( \tau \) be a transformation. The distances can be written as in the following equation (5.8)

\[
d_\tau = \tau(\| W^x_l x_l^k - W^y_l \|_E) \] (5.8)
Instead of including the pruning process in the distance function, a normalization can be done to produce the same effects as showed in 5.8. The second step of normalization can be written as in the equation 5.9:

\[
\tilde{x}_l = W^x_l \hat{x}_l
\]

where

\[
W^x_l = \text{diag}(w(x_{v_1}^l), w(x_{v_2}^l), \ldots, w(x_{v_K}^l))
\]

(5.9)

The pruning function chosen is a sigmoid function. Its expression is written in the equation 5.10 and the figure 5.7 represents its shape. The value of the function is almost zero when the speed is low. When the speed exceeds a threshold, the function increases and reaches 1. The parameters have been chosen empirically trying to remove the values corresponding to the lowest speed and keeping the general shape of the signals \((v_{th} = 2\) and \(\lambda = 10\)).

\[
\forall v \in \mathbb{R}, w(v) = \frac{1}{1 + \exp(-\lambda(|v| - v_{th}))}
\]

(5.10)

### 5.2.3 The distance function

To select the most adapted distance function, a comparison between a few candidates has been made. The distance functions proposed were designed so as to be simple and adapted for a quick processing. The choice of the distance function is crucial since it is used to find similarities between the signals.
The candidate distances

Four distance functions have been compared in this study. In the following, \( x \) and \( y \) are two multi-dimensional time-signals which have already been normalized as explained in the previous section. The figure 5.8 represents the distances between two one-dimensional signals for a better visualization.

**Euclidean distance**

The Euclidean distance consists of calculating the sum of the norm of the difference between the signals at each time instant \( n \) as written in the equation 5.11. This distance is given as reference in the literature.

\[
d_E(x, y) = \sum_{l} ||x_l - y_l||_E
\]  

**Maximum distance**

This distance is a common distance for vectors used in our case for time-series. The principle is simple: the distance between two signals is the maximum the distances at each time instant (see equation 5.12).

\[
d_M(x, y) = \max_{l} ||x_l - y_l||_E
\]

**Sum of maximum distance**

This is an extension of the maximum distance. It involves summing the \( w \) biggest distances among the record (see equation 5.13). This distance requires as a parameter \( w \) the number of values to sum up.

\[
d_S(x, y) = \frac{1}{w} \max_{\{t_i\}_{i=1}^w, \text{with } l_i \neq l_j} \sum_{i=1}^{w} ||x_{l_i} - y_{l_i}||_E
\]

**Maximum window distance**

This distance involves finding the window of time for which the Euclidean distance between the two subsequence is the maximum (see equation 5.14). It takes in parameter the larger of the window \( w \).

\[
d_W(x, y) = \frac{1}{w} \max_{i} \sum_{i}^{i+w-1} ||x_l - y_l||_E
\]
Figure 5.8: Visualization of the candidate distances applied on two time-series in one dimension
Performances

The four distances have been tested with the data gathered during the experiment and labeled by hand. The data is divided in four subsets each representing a scenario. The goal here is to see how well the distances are able to differentiate between the scenarios by themselves.

Method for performance evaluation

To evaluate the performance, for each signal, the mean of the studied distances $d_*$ to the 5 closest signals from the same label (S.L.) and from a different label (D.L.) have been calculated and compared. By this method, one can see if the distance function is efficient for differentiating between signals from the same category and signal from other categories. To make a concrete number on this separation power, the quotient $R$ between those two values has been calculated for each input signal. With this ratio, one can estimate how many times farther the nearest signals from other labels are compared to the nearest signals of the same label. Let $|N_{S\ell}(\omega)|$ and $|N_{D\ell}(\omega)|$ the set of the 5 nearest signals from $\omega$ respectively belonging to the same label and to a different label. One can define mathematically the means distances and the quotient $R$ as in the equation 5.15.

$$
\begin{align*}
\mu_{S\ell}(\omega) &= \frac{1}{5} \sum_{\nu \in |N_{S\ell}(\omega)|} d_*(\omega, \nu) \\
\mu_{D\ell}(\omega) &= \frac{1}{5} \sum_{\nu \in |N_{D\ell}(\omega)|} d_*(\omega, \nu) \\
R(\omega) &= \frac{\mu_{D\ell}(\omega)}{\mu_{S\ell}(\omega)}
\end{align*}
$$

Reading of the graphics

The figures 5.9 illustrates the results obtained for the four distances with the method explained above. The figure 5.9a is a plot of $\mu_{D\ell}$ as a function of $\mu_{S\ell}$ for each distance function. The plot is separated in two parts. The upper part is where $\mu_{S\ell}$ is smaller than $\mu_{D\ell}$ which means that the algorithm is able to differentiate between the signals from the same label compared to the others. In the lower part, $\mu_{S\ell}$ is bigger than $\mu_{D\ell}$ so the signals will probably be misclassified during a classification steps using the KNN method. In other words, the further the points of the graph are from the first diagonal of the plan in the upper corner, the more efficient is the distance function in classifying between the different classes. The axes $\mu_{S\ell} = d_{D\ell}$ is displayed for the purpose of clarification. The curves in the figures 5.9b shows the distribution of the ratio $R$ calculated for all the signals. The values of $R$ have been ordered by increasing values. The higher is the curve, the better the distance function can discriminate between cases.
Interpretation

One can immediately see that the Euclidean distance function has poor results compared to the other functions. The points on the figure 5.9a are close to the first diagonal and the corresponding curve on the figure 5.9b is under all the others stabilizing at around 2 for the best signals. The integration of all the small variations among the entire signal blurs the distinction between the cases. The distance between two signals from the same scenario is already important due to this integration of noise. On the contrary the maximum distance has remarkably good results. The distances between similar signals is in most of the cases greater than 3 times more the distance to a signal with another label. The maximum distance is more efficient because it can more easily detect a pick of divergence. On the one hand, if two signals differ of a local variation, the maximum distance focuses directly on the highest difference. On the other, the difference between two similar signals is very low for every time instant. The ratio $R$ is thus raised by a slight increase in $\mu_{DL}$ but mostly a large decrease in $\mu_{DL}$. However, this distance relies only on one time value of the signals and can be disturbed by a pick of noise.

The two next distances have been designed in order to address this last issue raised. The maximum search of distance allows a better resolution but can easily be disturbed by a noise measurement. The idea is then to sum up several maxima to decrease the impact of the small variations by dividing by $w$. The sum of the $w$ maximum differences uses then the same principle as for the maximum function. Its results are better than the Euclidean distance but reduces the performances slightly compared to the maximum function. In fact, this operation mostly helped to make a better difference between two signals from different labels. However, when two signals come from the same labels, their differences at all time instances are relatively small and probably spread over the operation time. Calculating the sum of maximum values that are spread over the time is not particularly relevant. Instead, the last candidate function compares portions of signals and finds the maximum Euclidean distance for all the windows of time. One can see on the figure 5.9a that this last operation is slightly better at reducing the distances between signals of the same set. Moreover, the maximum window difference has even better results than the maximum difference in terms of ratio $R$. On the figure 5.9b, the curve corresponding to the maximum window distance is greater than the three other distance function for most of the signals.

Nevertheless, the size of the window $w$ for the distance function has to be defined wisely. The function is perfectly equivalent to the maximum distance if the window is one-element large and perfectly equivalent to the Euclidean function for a window equal to the size of the signals. Then if $w$ is small, the efficiency towards the discrimination of picks is high but can be disturbed
by noise. On the contrary, if the window is too large, the picks of difference are drowned among small variations of the signals.

The chosen distance function for our study is then the maximum window difference. It easily differentiates between the different example cases proposed. The parameter $w$ has been fixed to 4 which corresponds in the time domain to around $10\,\text{ms}$. The difference is thus generic and provides a good and reliable detection of divergence picks reducing the impact of variation in the measurements.

5.3 Clustering and training data labeling

As described in the chapter 3, the training data is clustered so as to shows the structure of the data with an intuitive way to the user. The example treated in the chapter 3 deals with classification of objects represented by a point in a 2d feature space. As explained in the previous section, the feature extraction for time-series is difficult and not suitable to compare shapes between two signals. However, as the notion of neighborhood is directly linked to the notion of distance, the ordering and clustering method OPTICS only require a distance function between two elements. The maximum window distance function was then used to discriminate between several multidimensional time-series.

The first one hundred signals of the data labeled by hand were used as training data for the algorithm. The algorithm OPTICS provided an order of the points and the curves of $RD*$ and $CD$ values shown in the figure 5.10. Then the tree representation of the data is processed by the multi-thresholding method. The figure 5.11 illustrates all the different thresholds for each possible cluster. The corresponding tree is shown in the figure 5.12. The last figure 5.13 indicates the best clusters that an operator can select.

On the figure 5.13, the line of colored dots represents the different scenario reported by hand during the experiment. One can notice that the ordering method has gathered all the signals from the same scenario into segments of successive indexes. The best possible selection is the selection of the biggest segments representing each one unique scenario. On the figure 5.12, the clusters gathering signals from only one scenario has been colored. Another way to represent the best cluster selection is to find the highest branches of the tree representing one scenario.

To perform the cluster selection, the operator is assisted by a software. The software displays a video feedback of the typical cases gathered in the cluster. The operator can easily visualize if an operation is a success or if
Figure 5.9: Comparison between the distance functions

(a) $\mu_{DL}$ VS $\mu_{SL}$

(b) Croissant sorted values of $R$ for the different distances
a failure has happened on the video. By finding similarities between the clusters, most of the branches can be simplified as explained in the section 3.3.

With the training data available, the signals gathered for the scenario 4 are too far from each other to detect a cluster. The creating of a cluster could have been possible if the signals were closer or if the failure had happened more often during the training phase.

5.4 Performance of the learned classification function

After the training phase and the association of each case to a success or failure, a classification function is learned with the labeled data gathered in the clustering phase. As explained in the chapter 4, the classification learned is based on the LOF factor and is able to differentiate between a scenario observed in the training phase and a scenario unidentified.

For the purpose of clarification, we have made manually correspond the numbers of the experimental scenarios and the cluster identification numbers.
Figure 5.11: Multi-thresholding method
(Red curve: $RD^*$; Blue curve: $CD$;
Black line: segment corresponding to a cluster obtained for threshold $y = \theta$)

Figure 5.12: Tree representation of the similarities in the data
The colored clusters represent only one scenario
Blue: Scenario 1; Green: Scenario 2; Cyan: Scenario 3; Red: Scenario 4)
Figure 5.13: Clusters Selection
(Red curve: $RD^*$; Blue curve: $CD$;
Black line: segment corresponding to a cluster obtained for threshold $y = \theta$;
Colored dots: representation of the scenarios reported by hand;
Blue: Scenario 1; Green: Scenario 2; Cyan: Scenario 3; Red: Scenario 4)
In the results displayed later, the cluster $i$ has been learned from example signals extracted from the scenario $i$. One can remark that the scenario 4 was not identified during the clustering phase. It does not exist any cluster to represent this scenario in the labeled training data for the classification function. The signals extracted from this scenario have to be classified as unknown.

The figure 5.14 shows the results of the classification function. The plot is separated in four lines. Each line corresponds to a scenario reported during the experiment. Each tested signal corresponding to one of the scenarios is represented by a point on the respective line. The decision of the classification function for a signal is shown by the color of its representative point. In other words, a blue point on the line 2 would represent a signal extracted from the scenario 2 which has been associated to the cluster 1 supposed to represent the scenario 1. This example point would then be an outlier. A black dot relates a signal classified as outlier. The signal has a singular shape and is too far from the training data examples.

One can see that each scenario is never mixed with one other. As it has not been discovered in the training period, the fourth type of failure is excluded and each signal from this scenario is considered as unknown. These results displayed on the figure 5.14 were obtained with a sensitivity threshold $\theta = 2$.

The figure 5.15 shows the values of the best score obtained with each signal. The color of the curve corresponds to one scenario. One can see that for each known scenario in the training data, the obtained score is very low compared to the ones for the scenario 4. The horizontal line for
$y = 2$ represents the threshold in the scores. One can see that only the point extracted from the scenario 4 are rejected. Fixing the sensitivity threshold to an upper value would lead to a wrong association of a signal from the scenario 4 to one of the scenarios discovered in the training data. On the contrary, if the threshold is too low, the sensitivity to new signals would increase. Some signals from known scenarios would be classified as noise even if the best score found correspond to the right scenario.

To have an idea of the efficiency of the classification function, one can also look at the mean score of each clusters for one scenario. These mean values are shown in the table 5.1. One can see on this table that the mean scores between a scenario and its corresponding cluster is very close to 1 as foreseen. Moreover, a misclassification between known classes is not very likely in our application since the other mean scores are much higher. The scores for the fourth scenario are relatively high for each cluster since their values are far above 1. To avoid the association of a scenrio-4 signal with an
existing cluster, the sensitivity has to be tuned for a value much lower than those three averages.

Table 5.1: Mean LOF Scores for each scenario and each cluster

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>1.09</td>
<td>10.6</td>
<td>2.82</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>8.03</td>
<td>1.05</td>
<td>6.61</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>4.97</td>
<td>10.6</td>
<td>0.85</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>3.27</td>
<td>8.24</td>
<td>3.57</td>
</tr>
</tbody>
</table>

To conclude, the results of this algorithm are satisfying. For a good tuning of the sensitivity parameter, the classification function does not make any mistake. As the training data has a relatively small size (around 100 signals), the computation of the results is quick (less than 50ms on Matlab for each signal) which is far beyond the requirements. However, a consequent quantity of data is required to reach these performances. All the signals of the training have to be stored together with the labels and the distances between each signal for a quicker implementation. The setting of a correct sensitivity threshold is an understandable parameter for a worker. It can be either defined by default like in this study or tuned by the robot coach himself.
Chapter 6

Discussion and future work

The system proposed in this thesis gives good results in terms of accuracy, rapidity and usability for a novice user. However, the principle of this algorithm suffers from a main inherent drawback: the algorithm is very sensitive to a general change of signal. This section presents the different observations made during experiments which highlight this phenomenon. Then some potential improvements are proposed and finally, a presentation of other application domains are presented.

6.1 High-sensitivity to a change of shape

6.1.1 Observations

The inherent problem of the algorithm is that the performance is significantly decreased by a change of signal over time. The signals recorded during the training period create a reference set which is used for the entire application. However, if the shape of the signals changes over time, the classification function will automatically classify the signals as a new behavior. The performance can then be drastically affected.

A special experiment has been done with the robot in a longer time to see how much the signal can vary over time. Without changing the program, the robot ran during 40 minutes in a loop without touching any object. The results of this experiment are given in figure 6.1. One can see that progressively the signals shift over the time and in particular during the first minutes. This shifting is due to a variation of the friction forces in the joints of the robot. After a certain amount of operations, the temperature of the motor and the articulation rises. This produces a change in the efficiency of the motor and reduces the viscous friction in the gear box.

The designed algorithm has been trained with the first 100 signals collected at the beginning experiment. As the robot was moving the whole time in the air, only one scenario is repeated here for each iteration. The goal of this experiment is thus to see if the algorithm is able to recognize the same
scenario even after a long period of time and some shifting of signals. The figure 6.2 shows the evolution of the classification function scores obtained over time. One can see that the score progressively increases. After almost 300 operations, the score finally reaches the sensitivity threshold and classify a signal as outlier.

By the same effect, we could have shown that the algorithm used with the estimated motor torques is as well extremely sensitive to the position and to the speed of operation. The robot has to repeat the exact same movements after the training. Otherwise another training would be required to adapt to the new speed or trajectory.

6.1.2 Simple trials of stabilization

To address the problem of shifting signals in the estimated motor torques application, some simple fixes have been applied to make the algorithm more robust.

One first idea was to create a normalization which compensates the variation of the friction force. The amplitude normalization is efficient to partially erase the shift but one can still see some divergence in the signals. For better performances, the normalization would have to entirely remove
the friction force for each motor. However, an accurate estimation of the torque is difficult to calculate. A too simple estimations of these forces was really noisy and inaccurate. It was not possible to make a proper training data for the learning algorithm.

Another solution tried is to extend the training period. The training data gathered is composed of the signals during the shift. The results of the classification algorithm are thus more accurate in the time. However, depending on the machine and the application, this heating period could take a long time. Consequently, the training data might be huge and some problem of memory and rapidity can be raised.

6.2 Solutions proposals

The algorithm proposed is sensitive to any change of signal shape compared to the training data. One simple solution is then to choose some signals which are not supposed to change over time. Some other sensors could be used for this failure detection. One idea could be to use sensors to detect only the force applied on an external object. For example, if the robot had sensors on its hand to measure how much force is applied on the object the robot is touching, the signals obtained would not shift over time and
would have similar patterns attesting the occurrence of a failure. Other types of sensor can also be used. The operation used in the demo generates a particular 'click' sound once the two parts are correctly assembled. This sound could be recorded and analyzed in failure detection algorithm. The sound of the operation is probably stable and would not change over time.

Another way to solve this shifting problem is to improve the machine learning algorithm to take into account the signal variations. The clustering algorithm OPTICS has an incremental update version derived in the article [16]. After the first determination of the success and failures in the RD plot provided by OPTICS, the algorithm could continuously record and store new signals and associate them to one of the existing cases. As the shift occurs in small increments, it could be absorbed in the self-growing training data. In this manner the classification function could learn the shift. Then the scenarios could be recognized even after several hours. Nevertheless, this automatic learning cannot store every single signal for trivial memory issues. The algorithm would have to choose automatically the most relevant signals to store or forget some useless signals. Some other issues could as well appear. For example, two clusters corresponding to different scenarios could be close to each other and eventually merge together. Such situations would have to be handled in order to ensure a good general behavior.

One recurrent problem involves the memory required to store the signals. Indeed the method presented here needs a big quantity of memory to save the signals encountered in the training period. To decrease the memory requirements, one can think about a compression of the data which would decrease the space required. Instead of using the entire signal, only the trend or the general shape could be used and stored. The signals could be cut for example in small parts which could be described by a standard behavior such as linear, constant, exponential... This idea has been explored in the article [19] to create a complex feature extraction method.

6.3 Similar subjects for further application in machine learning

Leaving the failure detection aside, this machine learning algorithm could be used for other applications. It could be used for example to detect an intrusion in the workspace of the robot: Using proximity sensors placed on the wrist of the robot, the distance to the object next to the robot would be recorded during its operation cycle. When someone enters the robot workspace, the sensors will record a different signal that could be automatically detected. By the same principle, an impact detection could be implemented. A collision would generate force signals which diverge
significantly relatively to the variance in the training data.

To increase the robustness of the robots, one also can think about the same kind of algorithm but using images instead of time-series signals. With the exact same principle, the algorithm would compare images of the assembled products and create clusters of the different results. An operator could then define which of the cases are success ones and which are failures ones. The robot would be able afterwards to determine if the assembly was correct or wrong and then correct its mistakes.
Bibliography


