Universität Bremen

Fachbereich 3: Mathematik und Informatik

Master's Thesis

Event-Aware Execution of Robotic Wiping Actions using Classification of Task-Specific Force Profiles

Simon Stelter

Matriculation No. 258 932 0

09. 03. 2017

Examiner: Prof. Michael Beetz, PhD
Supervisor: Prof. Dr.-Ing. Udo Frese
Advisor: Georg Bartels

Simon Stelter

Event-Aware Execution of Robotic Wiping Actions using Classification of Task-Specific Force Profiles

Master's Thesis, Fachbereich 3: Mathematik und Informatik

Universität Bremen, March 2017

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 09. 03. 2017

Simon Stelter

Acknowledgements

An dieser Stelle möchte ich Georg Bartels für sein hohes Engagement in der Rolle des Advisors meinen Dank aussprechen. Des Weiteren möchte ich Prof. Michael Beetz und der Arbeitsgruppe des IAI für die Bereitstellung des Roboters danken, was die Aufnahme von echten Sensormessungen ermöglichte.

Abstract

Current service robots lack the manipulation skills that nature has perfected in humans. Findings from cognitive psychology suggest that humans divide manipulation tasks into subgoals, for which it predicts and expects certain events, e.g., making or breaking of contacts. This thesis investigates how such contact events can be reliably detected and classified in force/torque sensor readings during robotic wiping tasks. An algorithm is presented that learns the task-specific force profiles of contact events using multidimensional time series shapelets. In contrast to comparable methods, this algorithm does not rely on the assumption that perfectly extracted patterns are available during training. Instead, the training dataset contained 520 robotic wiping episodes that are only labeled with a list of occurred events without temporal information. Nevertheless, contact events are detected and classified during wiping task execution online with a high precision and sensitivity.

Contents

Chapter 1

Introduction

1.1 Motivation

Current service robots lack the manipulation skills that nature has perfected in humans over the course of evolution. Cognitive psychology researchers have analyzed human manipulation. Their findings suggest that humans' high level of competence can be attributed to the way the nervous system organizes and controls manual tasks [10], [16], [23]. Apparently, the human brain divides manipulation tasks into subgoals, for which it predicts and expects certain events, e.g., making or breaking of contacts. If the perceived events differ from the predictions, the respective movements will be adjusted. As an example, consider a human trying to pick up an object. His action plan might be a sequence of three subactions, namely, reaching for the object, grasping it and then lifting it. For the first subaction he will put his hand near the object until it looks close or he touches it. Then he will start grasping, expecting a firm contact. Based on the sensed force in his fingers, he can change his grasping force, as well as his prediction for the subsequent lift off subaction. If the object is as heavy as predicted, the movement will look smooth. However, if the perceived events indicate a lighter or heavier object, he will pause and adjust his strength.

Supported by these findings from cognitive psychology, Winkler and Beetz [48] have proposed an architecture for a service robots that allows them to form expectations (i.e. build a decision tree) for action plans based on past executions, and to reason about task success. For many tasks, the most reliable source of information is camera vision, for example to search for objects or to plan trajectories around obstacles. Haptic perception systems are much less sophisticated. If present, they often come down to a simple distinction between light or strong sensor readings. This approach is less reliable in tasks where the robot can not rely on its vision system, which is the case in robotic wiping tasks. Here, camera vision is often impaired by the robots hand, arm or held tool. With *robotic wiping tasks* I refer to actions in which a robot moves a tool along a support surface, usually to manipulate (spread out, soak up, etc.) a third medium. These wiping tasks, however, make up about half of a future service robots todo-list, mainly in the context of cleaning, according to an analysis of household chore lists by Cakmak et al. [3].

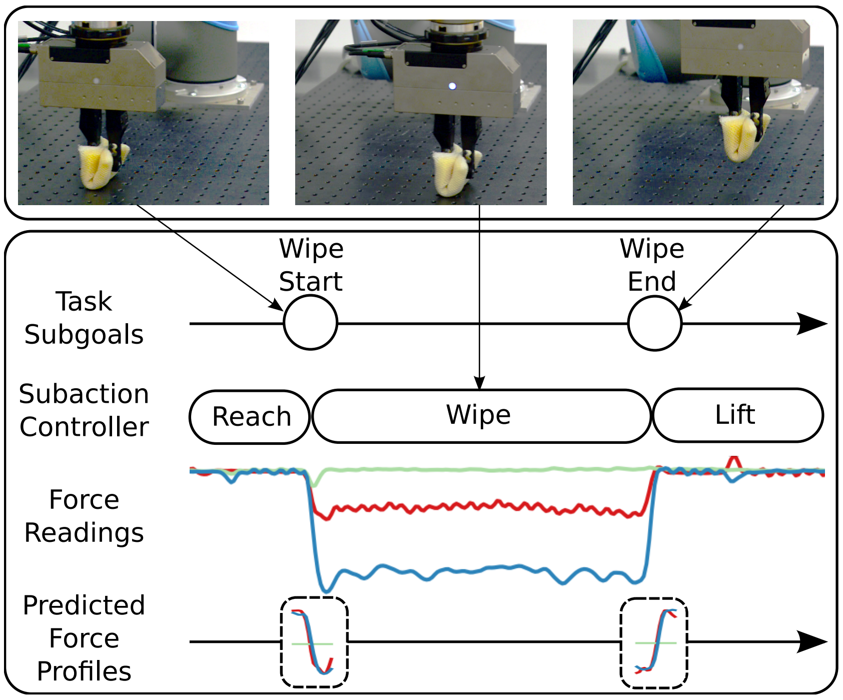


Figure 1.1 Representation of a table wiping action structured by contact events, inspired by cognitive psychology [10], [16], [23].

Motivated by this, I will contribute to the research on haptic perception in order to bring robot manipulation skills closer to the competence level of humans. As a model problem, I will investigate contact event detection in stereotypical wiping tasks, in which a robot wipes in a straight line over a surface or alongside an edge. Fig. 1.1 depicts the execution of a simple wiping task and the force sensor readings from a wrist-mounted sensor. The three phases, reach, wipe and lift, are difficult to detect because they are just straight lines at different force levels. However, the events between the phases contain additional shape information. I hypothesize that these force profiles can be used to reliably detect the contact events. Supportingly, other robotics researchers reported that it is easier to detect contact events than contact states from force measurements because the information content of the signals is higher directly before or after contact states [9], [17]. Such an event detection system can be integrated into the high-level planning architecture presented by Winkler and Beetz [48]. The high-level plan for the wiping task in Fig. 1.1 might identify two subgoals, wipe start and wipe end, between the subactions. A corresponding list of force profiles can then be generated using past experiences and compared to the sensor measurements to verify that a subgoal is accomplished. If the expectations do not match reality, the high-level plan is informed and can act according to its decision tree.

In this thesis, I investigate how these force profiles can be learned based on force/torque sensor recordings from past experiences, in order to reliably detect and classify their corresponding events in future trials. It is a topic for future research to enhance a high-level planning system with this ability.

The underlying problem is the classification of subsequences in streamed time series data. Time series classification in general suffers from the curse of dimensionality. In high-dimensional data such as the time series, most popular machine learning algorithms exhibit a high training or classification times or lose their ability to generalize altogether. On top of that, streamed data require a certain classification speed, especially in time-critical applications such as robotics, and pose new problems such as trivial matches (i.e., an event is classified in two successive subsequences).

This topic is also being investigated in other research areas, for example in gesture recognition [32], [21], [4], [24]. However, the presented solutions rely almost always on the assumption, that „copious amounts of perfectly aligned atomic patterns can be obtained" [20]. This assumption makes these algorithms unsuitable to achieve my goal because a robot can hardly identify and extract the event pattern in past recordings, if it has not learned them yet. Especially for unexpected events, not even an exact point in time can be assigned, let alone the length of the corresponding time series subsequence. The only information that can be given with high certainty is a list of events which have occurred somewhere in a recording. This is called weakly labeled data [20].

To remove the requirement for a perfect training set, I will present a machine learning algorithm that exploits the time series shapelet discovery algorithm [51] to learn models for the unique force profiles of events. Shapelets are short shape snippets that are good at separating classes of time series by searching for the presence or absence of that shape. Weakly labeled data can then be divided into experiments that contain event X and experiments who do not. The single best shapelet to separate these two classes should be the sequence with the shape of X. This shapelet can then be used to search for similar shape snippets in the time series streamed during task execution. Therefore I only rely on the assumptions, that the training examples can be divided into subsets with and without event X and that these subsets are not the same for any two events classes.

To evaluate my algorithm, I have recorded 520 wiping episodes of a table mounted robotic manipulator with a wrist mounted force/torque sensor, which resulted in 106 minutes of force/torque sensor readings. Each episode is paired with a list of up to ten event labels, that I have manually identified. A shapelet based classifier is trained for each event and tested for its ability to detect and classify it.

1.2 Thesis Contribution

This thesis contributes to the state of the art in robotic manipulation and time series stream classification. In particular:

- I will demonstrate that contact events during robotic wiping tasks cause force/torque measurements with distinct shape, which can be used to detect and classify them.
- I will show that multidimensional time series shapelets (*MTS*) can capture these events and detect them online, if the individual dimensions are time synchronous.
- I present an algorithm to discover such *MTS* in weakly labeled training data.
- I prove mathematically that shapelets satisfy a constraint, that can be exploited to simplify parameter tuning, and that enables a simple learning technique, which significantly reduces the training time without a huge classification performance trade-off.

1.3 Structure of the Document

The remainder of this Thesis is structured as follows. In chapter 2 I will give an overview of related work in the field of robotic wiping tasks, and event detection and classification in time series streams. Thereafter, I will give an intuition and reasoning behind my *MTS* discovery algorithm in section 3.1, followed by a detailed description in section 3.4. In chapter 4 the algorithm will be evaluated and the last chapter 5 contains the conclusion.

Chapter 2

Related Work

2.1 Robotic Wiping Tasks

In the context of service robotics, wiping actions are an important topic, because they make up most of their tasks. In the analysis of household chore lists conducted by Cakmak et al. [3] about half of the listed tasks are wiping related cleaning tasks, e.g., vacuum cleaning, doing the dishes or cleaning windows.

Leidner et al. have therefore paid much attention to this topic. In a series of papers, they have first presented a taxonomy of compliant manipulation tasks, including wiping tasks, which are described as an external manipulation in which a soft object is guided along a rigid object or vice versa [27]. Usually a medium is involved, which is spread out, soaked up, collected, etc. In [28] they have used a humanoid robot which executed whole-body motions to clean a window, to scrub a mug and to collect shards with a broom. However, the robot had no internal model of the desired effect nor could he reason about the performance of its actions. The first problem was addressed in [26], by representing the medium in wiping tasks as a generic particle distribution and planning Cartesian motions to alter this distribution. It was combined with a path following technique, that takes into account the free DOF of the used tool. In their latest paper [25], a method was added to infer the effect of wiping motions based on haptic perception, in order to reason about the performance of the action. Hess et al. [15] have also calculated efficient motions to vacuum-clean planar surfaces. But instead of haptic feedback, they utilized color segmentation in camera images to detect areas, which were still dirty.

In order to reliably perform wiping actions, Schindlbeck et al. [43] have proposed a combination of force and impedance control to safely react to unexpected contact loss in a polishing task. Ortenzi et al. [35] took advantage of geometric constraints, such as movements along a surface, presented by the environment to decouple force and motion control thereby reducing joint torques.

Findings in cognitive psychology suggest, that humans divide movements into contact event subgoals and react depending on whether or not the predictions were met [10], [16], [23]. However, most research in the context of wiping tasks focused on either control systems, that allow for a smooth and save execution, or path planning. The latter took visual and haptic perception only into account after the movement has ended, to determine previously poorly cleaned areas.

In order to achieve event-awareness during wiping tasks, event detection and classification in force/torque time series streams seems to be the most promising approach, because camera vision, which is often used in other context, is often impaired by the arm, hand or tool. I will therefore review techniques on time series streams in general in the next section.

2.2 Event Detection and Classification in Time Series Streams

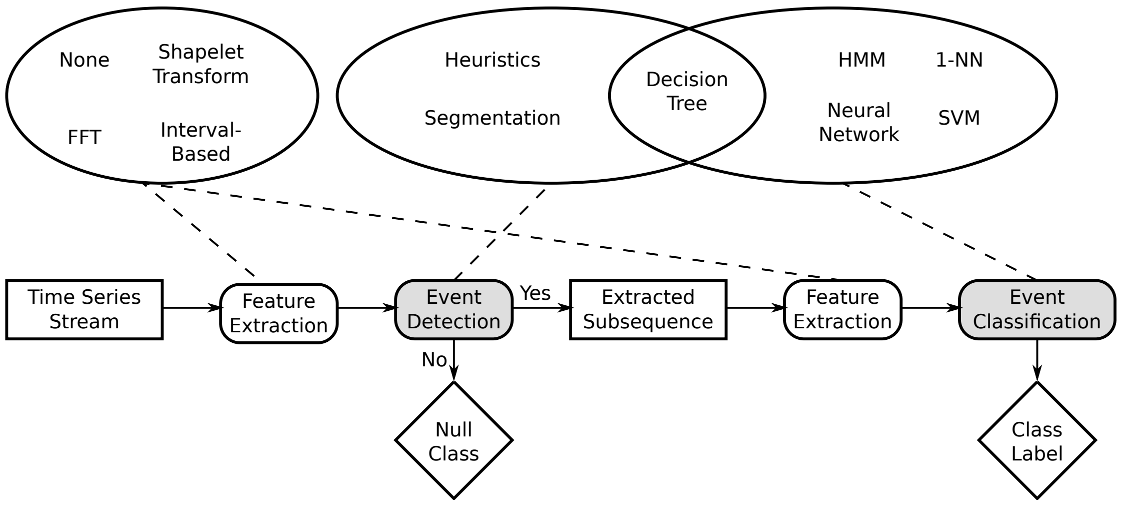


Figure 2.1 General event detection and classification pipeline.

In this section I give an overview of different approaches that detect and classify events in time series streams, for various application scenarios. In a series of publications, Hovland and McCarragher have tried to detect events in force/torque sensor readings during a robotic manipulation task [17], [18], [19]. Other papers focus on event detection in electricity data [36], [52], traffic data [33], [14] or pattern recognition in ECG time series [45]. Activity detection in wearable sensor data also received a lot of attention [32], [21], [4], [24].

Fig. 2.1 depicts a general pipeline for event detection and classification. It consists of two major phases, event detection and event classification, each with an optional preprocessing step for feature extraction step.

The main purpose of separating these phases is to improve the reaction time of the system. It is generally easier to differentiate between stream sections with and without activity, than it is to classify an event. Hence, a computationally inexpensive event detector can quickly dismiss large portions of the input stream. The most popular means for this are time series segmentation algorithms. In the event classification phase ordinary algorithms for complete time series can be used on promising stream segments provided by the event detection phase. However, a separate event detection phase is not a necessity, one could just start the classification process at every incoming data point.

2.2.1 Feature Extraction

There are three popular approaches to extract features from time series data: fast fourier transformation (FFT) [5], shapelets [31] and interval-based transformations. If no feature extraction is used, the time series is directly used as a feature vector.

The first one was used in [17], [18], [19] to transform the input from the time domain into the frequency domain. This can be advantages, if the events can be identified using specific frequencies. For example, a door bell can easily be detected in audio data because it always produces the same frequency waves.

The basic idea behind shapelets is that time series can more easily be classified be searching for the presence or absence of shape snippets (the shapelets), instead of considering the complete shape. They are usually short subsequences from training examples that are good at separating two classes. A new feature vector for an input time series is generated by calculating the best match distance for every shapelet to the time series. Shapelets will be explained in more detail in section 3.1. This technique was utilized in [36], [32], [52].

Interval-based approaches extract features from the input series, by sliding a window over it and calculating features for each interval. Popular ones are mean, root mean square, standard deviation or the wavelet transform [45], [14], [4], [21], [45].

Most of these techniques were only applied for the classification step.

2.2.2 Event Detection in Time Series Streams

A major challenge for online processing of continuous data streams is that events can start and end at any time points. Therefore, a first goal is to quickly dismiss subsequences that do not contain events. Segmentation algorithms are the most popular approaches for this. However, as noted by Rakthanmanon et al. [41], this phrase is unfortunately overloaded.

For instance, it can refer to the approximation of segments with polynomials, e.g., piecewise linear approximation. An often cited publication [13] on this topic interpreted events as points in between such segments.

A second interpretation refers to the division of the time series into semantically meaningful segments. Typically, approaches try to differentiate between activity and no activity. This can be achieved by calculating the standard deviation of subsequences [32]. Alternatively Chambers et al. [4] dismissed segments, if sensor readings are likely to be caused by gravity alone, thus indicating no activity. This segmentation method was also recommended by Ko et al. [24] as an optional event detection phase to improve the reaction time of their system.

A combination of both segmentation interpretations was used by Junker et al. [21]. First they partitioned the series into linear segments and then merged them to motion segments.

As for non-segmentation approaches, a landmark detection system was developed by Miao et al. [33]. In a nutshell, classes are represented by patterns which have a certain sequence of landmarks, e.g., points with high gradients or local extrema. The input stream is then searched for the same sequence of landmarks.

Simple [18], [19], [17] or complex hand crafted [14] heuristics were also used for the detection and the whole phase was skipped in [45], [24], [52].

Patri et al. [36] have trained a separate shapelet-decision tree classifiers for both phases. In this case the division was only made to enhance the classification performance of the system.

2.2.3 Event Classification in Time Series Streams

In the event classification phase, every or just the pre-selected subsequences from an input stream are classified. This view reduces the problem to the classification of complete time series and therefore algorithms from this field are applicable.

A good comparison of recently proposed time series classification methods can be found in [1]. Its main problem is the curse of dimensionality. An input series of length n can be viewed as a point in n-dimensional feature space. Hence, for a time series of length 100, a training set containing a trillion examples only covers a fraction of about 10^{-18} of the whole possible input space. Furthermore, in a high dimensional space it gets increasingly likely that examples are similar in some dimensions [8]. Imagine two binary feature vectors, that have a low similarity. If we now add more feature which are all 0 or random they become more similar. Even if the new features correlate with the old one, the similarity increases because of noise.

Fortunately, time series tend to not be uniformly distributed in their feature space. Which is why the standard benchmark time series classifier, 1-nearest neighbour (1-NN) using dynamic time warping (DTW) [42] or euclidean distance, performs well on many datasets. More sophisticated approaches rarely outperform 1-NN significantly on many dataset [1]. With a low training set size, DTW beats euclidean distance in terms of accuracy, however this difference vanishes as the training set size increases [7]. As euclidean distance is per definition faster than DTW (at least if DTW uses euclidean distance internally, which is most often the case), it is preferable on a large training set size. The major disadvantage of 1-NN is that the classification is very costly, since the distances between the input and every training example have to be calculated. There is an anytime version [47], but it exhibits a clear speed-accuracy trade-off. For that reason 1-NN is typically not a good choice for online classification.

However, most other supervised classification algorithms such as support vector machine (SVM) show very high training times, especially for training examples with high dimensionality. Therefore other time series classification techniques first start by transforming the series into a lower dimensional feature space, while hopefully retaining the important information, before applying such algorithms. These include the previously mentioned feature extraction methods. Worth mentioning are also dictionary based transformations such as Symbolic Aggregate approXimation (SAX) [30] that transform the series into a discrete symbolic representation, enabling the usage of text classification algorithms.

The apparently best time series classifier in terms of accuracy is the recently proposed collective of transformation based ensembles (COTE) [2]. As its name suggests, this classifier trains 35 different kinds of other time series classifiers together with respective weights. This leads to a high training time and classification time.

For the event classification phase, the most popular classifiers used were hidden markov models (HMM) [19], [17], [21], [4] and shapelet decision trees [36], [32], [52]. SVMs [14] and artificial neural networks (ANN) [45], [18] have also been used.

The advantage of HMMs is their ability to deal with temporal variations, for which reason they are often used in speech recognition [39]. Their disadvantage is that they are hard to visually inspect, just like ANN. In contrast, decision trees in combination with shapelets learn models that are easy to visualize. Shapelets are also z-normalized to zero mean and a standard deviation of 1. Hence, they are invariant to changes in offset and scale.

Another interpretable approach is presented by Miao et al. [33]. Domain experts create templates for the time series classes that allow for limited stretching in time, scale and offset.

Last but not least, 1-NN was used by Ko et al. [24]. To speed up the classification time only the distance to one representative per class is considered. By utilizing DTW, they are able to capture temporal differences.

2.2.4 Discussion of the Reviewed Literature

All of the previously cited literature makes the assumption, that the start and end point to the events are given during training time. In some of these papers it is more or less explicitly stated, that even during evaluation, such perfectly extracted subsequences are used [36], [52]. Labeling every class instance in a time series dataset collection from a stream is very labor intensive and can introduce human bias. Sometimes the only information available is that an event has occurred, but not when it happened or how the it looks like. If a human interprets the data to find out this information, then he is in fact performing the most difficult learning part. This observation was already made in Hu et al. [20].

Furthermore, most of the cited papers lack a clear true positive (TP) and false positive (FP) definition. Although TP are sometimes defined as the number of correctly predicted classes, this is not trivial when handling streaming data. Fig. 2.2 depicts several problematic situations. In

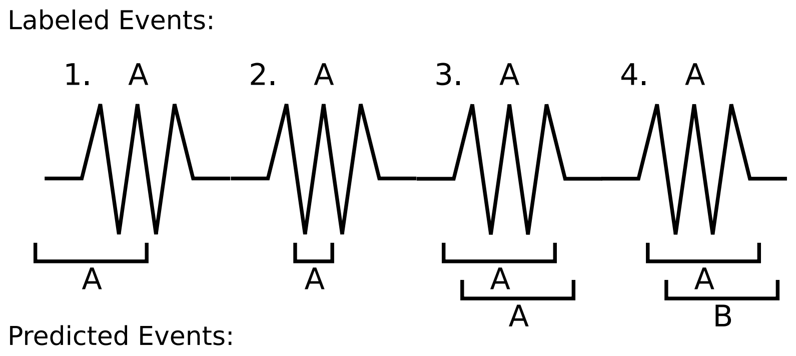


Figure 2.2 When to count a predicted event as TP or FP during the evaluation of event classification in streamed time series data?

the first example you can see a predicted event, that does not perfectly align with the labeled event. Especially in time series streams with a high sampling rate, a perfect match should not be required, but at how much of a difference does the detected event stop being a TP and becomes a FP? A similar problem arises, if the classified subsequence does not have the same length as the labeled event.

The third example is called a trivial match, which has been discussed in a different context in [22]. If an event was predicted in a subsequence starting at t, it will likely be predicted in $t + 1$ as well. Does the second, better match, turn the previous one from a TP into a FP?

In the last situation, the classifier predicted event A, which would be a TP on its own. But if an additional incorrect prediction was made, that is even closer to the labeled event, should the event A detection still be considered a TP?

To the best of my knowledge, there is no standard guideline for assigning TPs and FPs in this context.

However, two papers have addressed this problem to some extend. Sternickel [45] detected P wave patterns in ECG data. They have counted a TP, if a P wave was detected at least once in a fixed window before a QRS complex (an easily detectable pattern in ECG data), thus presenting a clear TP definition. Ko et al. [24] have acknowledged the problems in the first two examples, by reporting the time difference between the detected and real events. Furthermore, the problematic cases of the third and fourth picture were avoided by using recall $\left(\frac{TP}{TP+FN}\right)$ as the only performance measure, thereby ignoring FPs.

A similar problem comes up, when defining the negatives (N), because theoretically every point in the incoming stream starts a new time series subsequence. Should the negatives therefore be defined as the length of the tested input stream minus the amount of positives (P)? This problem renders the popular performance measure accuracy $\left(\frac{TP+TN}{N+P}\right)$ very problematic, as it results in a huge amount of Ns. Nevertheless it was the only reported performance measure in [52]. The rest of the reviewed literature has additionally and sometimes excursively reported precision $\left(\frac{TP}{TP+FP}\right)$ and/or recall $\left(\frac{TP}{TP+FN}\right)$. These are more meaningful, because they ignore true negatives (TN).

2.3 Time Series Clustering

In this section, I will give a short overview of time series clustering algorithms, because it is a popular subroutine in many algorithms. The main problem is again the curse of dimensionality. As a countermeasure, dimensionality reduction techniques have been proposed, for example those described in 2.2.1, as well as distance measures that try to exploit the nature of time series, such as DTW. A good overview on time series clustering can be found in [29].

However, there is the special case of subsequence time series (STS) clustering, where subsequences are extracted from a time series with a sliding window and then clustered. Keogh et al. [22] claim that this produces meaningless results. They have proven this empirically by generating 100 random walk dataset and finding the same rules in them, as Das et al. [6] (one of the most influential papers on this topic) did in the stock market, using STS clustering. The reason for this is that for any dataset, the average of k clusters, weighted by membership, must sum up to a straight line, the global mean of the original time series. As a consequence STS clustering is only useful, if someone really wants cluster centers that satisfy this constraint. In a later publication Rakthanmanona and Keogh et al. [41] have shown that this can only be achieved by ignoring some of the extracted subsequences.

Chapter 3

Methodology

3.1 Intuition

I will open this chapter by giving an intuition for how my algorithm works. To recap, the goal of the algorithm is to learn distinct force profiles from a weakly labeled dataset. A weakly labeled dataset only contains a list of event labels that have occurred somewhere in the training example.

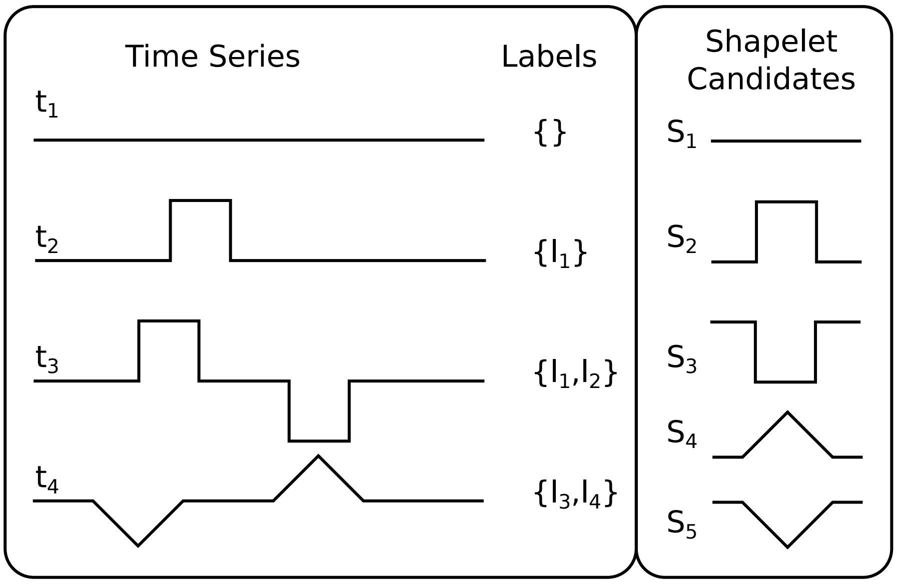


Figure 3.1 An example dataset containing four time series with a list of labels, that have occurred in the respective series. On the right side are five possible shapelet candidates.

In the dataset depicted in Fig. 3.1, there are four 1-dimensional training examples with a list of labels. Now a random label is chosen, for example l_1. The dataset is divided into two subsets, $D_1 = \{t_2, t_3\}$, containing all time series with l_1 and $D_0 = \{t_1, t_4\}$, containing all time series without l_1. The shapelet discovery algorithm generates possible short subsequences from the training examples in D_1. In this case these candidates are S_1, S_2 and S_3. Now all three candidates are tested for their ability to separate D_0 and D_1.

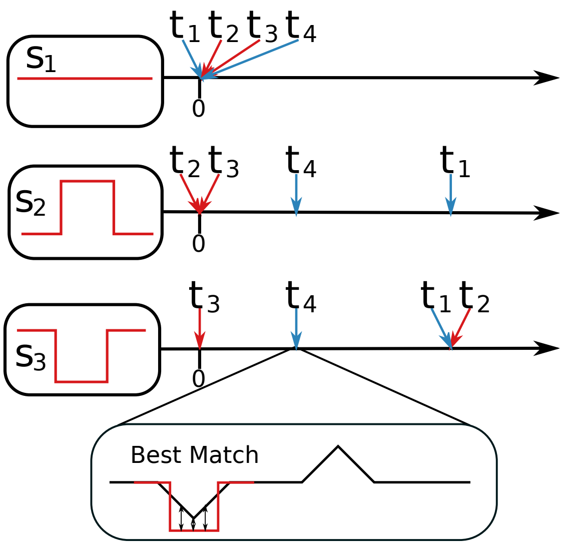


Figure 3.2 The 1-dimensional feature space of the three candidate shapelets for label l_1. Time series with and without the label are marked red and blue, respectively.

Each shapelet creates its own 1-dimensional feature space depicted in Fig. 3.2. The values for every time series corresponds to the distance the shapelet has at its best match. In this case, the manhattan distance is used, which is the sum of the distances between the point pairs of both time series. For example the first shapelet candidate has a perfect match in every time series, therefore all of them have a distance of 0 in its feature space. The second shapelet has a perfect match in the second and third time series, a decent match in t_4 and no good match in t_1. In this feature space D_0 and D_1 can be perfectly separated with a single threshold. The third shapelet achieves a better separation than S_1, because t_3 can be isolated. However, it has no good match in t_2. In conclusion, the second shapelet is the best candidate for l_1. As you can see this shapelet is in fact a unique profile for l_1. A distance threshold called δ can be computed from its feature space. In this case a point between the distances of t_2 and t_4 is best. The shapelet can now be used to detect an arbitrary amount of occurrences of l_1 in a time series by searching for matches that are closer than δ.

In a similar fashion, the S_3 turns out to be best suited at separating $D_1 = \{3\}$ from $D_0 = \{1, 2, 4\}$. However, S_4 and S_5 are equally good for both l_3 and l_4. These labels appear always together making it impossible to differentiate between them with the available information. That is why the algorithm relies on the assumption, that this situation does not happen. But, as you can see, a violation of this constraint has only an effect of the events that cause it. It is also much easier to satisfy than the availability of perfectly extracted patterns, which is the case for the vast majority of the published literature [20].

In the next sections a list of relevant definitions will be presented, followed by the proposed algorithm in detail.

3.2 Definitions and Notation

In this section I will introduce relevant definitions and notations which are necessary to communicate the proposed algorithm in the following sections.

Definition 1. *A **time series** t_x is a ordered sequence of real values, and $len(t_x)$ is its length. The real value at index i is denoted as $t_x[i]$, with the additional requirement $0 \leq i < len(t_x)$. A time series can be interpreted as a point in $\mathbb{R}^{len(t_x)}$.*

Definition 2. *A **time series subsequence** u_x of t_x is denoted by $u_x \sqsubseteq_l t_x$ with $l = len(u_x) \leq len(t_x)$, and is a sequence of consecutive values taken from t_x.*

Definition 3. *A **multidimensional time series** t is a set of time series, and $dim(t)$ denotes the set of associated dimension names. If $x \in dim(t)$ is the label of a particular dimension, the corresponding time series is called t_x. Additionally it is required that $\forall x, y \in dim(t) : len(t_x) = len(t_y) = len(t)$.*

Definition 4. *A **multidimensional time series subsequence** u of t denoted by $u \sqsubseteq_{(l,d)} t$, is a multidimensional time series with $l = len(u) \leq len(t)$, and $d = dim(u) \subseteq dim(t)$. Additionally, it is required that the time series subsequences u_x are generated with a consistent offset index i:*

$$\begin{aligned}&\forall x \in dim(u), \forall j \in \{0, .., len(u_x) - 1\},\\&\exists i \in \{0, .., len(t_x) - 1\} : t_x[i + j] = u_x[j].\end{aligned}$$

Definition 5. *A **weakly labeled dataset** D is a list of pairs (t, L_t), where t is a multidimensional time series and L_t is a list of event labels for t. This dataset is used during training.*

Definition 6. *A **test dataset** is a list of triples (t, L_t, r_t). A point in time for every event labeled in L_t is contained in r_t. This dataset is used only during the evaluation.*

Definition 7. *A **multidimensional time series shapelet (MTS)** S is a triple (s, δ, l), where s is a multidimensional time series, l is a class label and δ a distance threshold. If δ has not been determined, I write $(s,?,l)$.*

Definition 8. *The **mean** of a time series t_x is defined as the mean of all its points.*

$$\mu(t_x) = \frac{1}{n} \sum_{i=1}^{n} t_x[i] \tag{3.1}$$

Definition 9. *The **standard deviation** of a time series t_x as the standard deviation of its points.*

$$\sigma(t_x) = \sqrt{\mu(t_x^2) - (\mu(t_x))^2} \tag{3.2}$$

Definition 10. *The **z-normalization** sets a time series to mean 0 and standard deviation 1. Each time series of a multidimensional one is normalized in isolation.*

$$\begin{aligned} n_z(t) &= \{n_z(t_x) \mid t_x \in t\} \\ n_z(t_x) &= \begin{cases} \frac{t_x - \mu(t_x)}{\sigma(t_x)}, & \text{if } \sigma(t_x) \geq \sigma_{min} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.3}$$

Where σ_{min} is a user specified parameter that prevents the amplification of noise. If a number is subtracted from time series, all points of that time series are subtracted by that number.

Definition 11. *The **p-norm** of a time series is defined as:*

$$||t_x||_p = \left(\sum_{i=1}^{n} |t_x[i]|^p \right)^{1/p} \tag{3.4}$$

With $p \in \mathbb{R}^+$. Manhattan and euclidean norm are special cases with $p = 1$ and $p = 2$, respectively.

Definition 12. *A (distance) function is a **metric** if it satisfies the following properties*

1. $d(x, y) \geq 0$ *(non-negativity)*
2. $d(x, y) = 0 \Leftrightarrow x = y$ *(identity of indiscernibles)*
3. $d(x, y) = d(y, x)$ *(symmetry)*
4. $d(x, y) \leq d(x, y) + d(y, z)$ *(triangle inequality)*

Definition 13. *The generalization of the manhattan and euclidean distance is called **minkowski distance** and is a metric as well.*

$$minkowski(t_x, t_y) = ||t_x - t_y||_p \tag{3.5}$$

The manhattan and euclidean distances are special cases where $p = 1$ and $p = 2$, respectively.

Definition 14. *The n_p**-normalization** replaces the division by the standard deviation from the z-normalization with the p-norm of the time series.*

$$n_p(t_x) = \begin{cases} \frac{t_x - \mu(t_x)}{||t_x - \mu(t_x)||_p}, & \text{if } \sigma(t_x) \geq \sigma_{min} \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

Definition 15. *The **angular distance** is, under the assumption that x and y have a euclidean norm of* 1*, defined as*

$$ad(x, y) = \begin{cases} 0, & \text{if } x = y = \vec{0} \\ 1 - (x \bullet y), & \text{otherwise} \end{cases} \tag{3.7}$$

Where $\bullet$ denotes the dot product. This definition differs from other angular distance definitions in that it is defined if $x = 0 \vee y = 0$.

Definition 16. *The **best match distance (BMD)** between a MTS (s, δ, c) and a multidi-*

mensional time series t with $len(s) \leq len(t)$ is the distance between s and its closest subsequence $u \sqsubseteq_{(l,d)} t$.

$$BMD(s,t) = \min_{\substack{u \sqsubseteq_{(l,d)} t \\ l=len(s) \\ d=dim(s)}} \left\{ \frac{1}{|dim(s)|} \sum_{x \in dim(s)} dist(norm(s_x), norm(u_x)) \right\} \tag{3.8}$$

Where dist is a distance function, e.g., euclidean distance and norm is a normalization function used to achieve scale and offset invariance.

Definition 17. *The* ***entropy*** *of a dataset D consisting of two classes, 0 and 1, is given by:*

$$entropy(D) = -\left(P(0)\log_2 P(0) + P(1)\log_2 P(1)\right) \tag{3.9}$$

Where $P(x)$ is the proportion of objects from D belonging to class x.

Definition 18. *The* ***information gain (IG)*** *of a split strategy sp, which divides D into D_A and D_B, compares the entropy of D with the entropies of the new subsets after the split.*

$$IG(D, sp) = entropy(D) - \left(\frac{|D_A|}{|D|} entropy(D_A) + \frac{|D_B|}{|D|} entropy(D_B)\right) \tag{3.10}$$

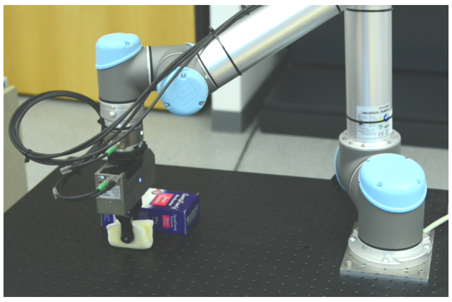
3.3 Experimental Setup

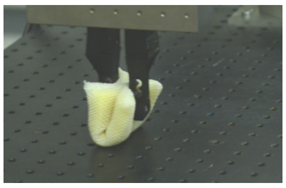
Figure 3.3a depicts the experimental setup that is used to execute the wiping tasks. It consists of a table-mounted 6-DOF manipulator[1] with a force/torque sensor[2] mounted between the end of the manipulator and an industrial parallel-jaw gripper[3]. The robot is holding a soft sponge in its gripper while performing wiping motions on the table surface.

All movements start in a contact-free state above the table. First, the gripper moves down to touch the table. Then, the robot wipes the sponge over the table in a straight line. If the goal position is reached, the gripper releases the contact with the surface and moves back to its starting position. In all experiments, the robot followed a predefined trajectory that applied different forces to the surface (including contact-free episodes). Furthermore the speed, direction relative to the gripper, and covered distance vary across the executions. The robot was controlled via ROS [38] and MoveIt! [46] was used for motion planning.

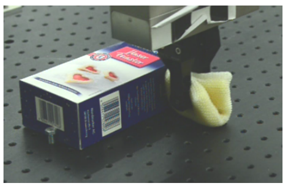
Different environments are created to produce various kinds of contact events. The following setups were used:

[1] *https://www.universal-robots.com/products/ur5-robot/*
[2] *https://www.weiss-robotics.com/en/produkte/force-torque-sensing/kms-40-en/*
[3] *https://www.weiss-robotics.com/en/produkte/gripping-systems/performance-line-en/wsg-50-en/*


(a) A table-mounted robot with a wrist-mounted force/torque sensor performs wiping tasks using a rag.


(b) Wiping over an empty table.

(d) Wiping into a movable box.


(c) Wiping along a fixated box.

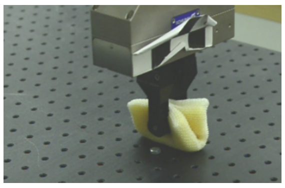

(e) Wiping over a tightened screw.

Figure 3.3 Experimental setup: Subfigure a) depicts the robot, while subfigures b) - e) show some of the contact events.

- An empty table. 3.3b
- A box placed into the path of the robot. 3.3d
- Up to three screws screwed into the table. 3.3e
- A box fixed on the table, such that the robot slides along it. 3.3c

During these experiments, I have identified the following contact events:

- *wipe_start*
- *wipe_end*
- *slide_right_start*
- *slide_right_end*
- *slide_left_start*
- *slide_left_end*
- *moveable_box*
- *fixed_screw*

Additionally recordings with the long-lived events *force_inc*, *force_dec* are included in the dataset. Here the robot was commanded to start with a low contact force towards the table which increased over the course of the execution, or vice versa.

3.3.1 Dataset

The weakly labeled dataset D is used for training. According to Def. 5, each training example contains a multidimensional time series t and a list of labels L_t. The time series t is 6 dimensional with $dim(t) = \{f_x, f_y, f_z, \tau_x, \tau_y, \tau_z\}$. These dimensions correspond to the force (f) and torque (τ) readings of the wrist-mounted sensor respectively. The measurements are recorded at 500hz, but downsampled to 25hz because it greatly reduces the learning time but still contains enough information to capture shape information. The list of labels L_t contains a subset of the contact event labels listed above and is created by hand. If an event happened twice during a wiping episode, the label is added only once. A few *slide* recordings have been cut in half and added to the dataset. Otherwise the start and end labels would always appear together, making it impossible for my algorithm to distinguish between them.

A second dataset is used for evaluation only, which is equal to the first one but each test instance contains an additional list of time points r_t. For example, if the robot wiped over two *fixed_screw*s during the recoding of t, then r_t contains two entries, one for each screw event.

All time series in the dataset are preprocessed such that the weight of the gripper is removed from the measurements and such that the x-axis points in the direction of the movement and the z-axis points towards the table. Fig. A.1 in the appendix depicts a few time series from the dataset.

3.3.1.1 Reduction of Gripper Influence

The force f_g and torque τ_g which are caused by the grippers weight can be estimated given its mass m, its center of mass c and the gravity vector g:

$$\begin{aligned} f_g &= mg \\ \tau_g &= mc \times g \end{aligned} \tag{3.11}$$

These values are subtracted from the measurements to reduce the grippers influence. A better estimation of the grippers weight is theoretically possible by including the speed, that the gripper is moving at, into the equation. However, this performed worse, probably because the speed is estimated based on the arms joint states instead of being directly measured by a sensor.

This step is not strictly necessary because my approach is offset invariant, but it makes the measurements easier to interpret.

3.3.1.2 Transformation into a Task-Specific Reference Frame

Not all wiping motions in the dataset are execution into the same direction, relative to the grippers orientation. As a result, the same contact events can produce their shape on different dimensions. As a countermeasure the sensor readings $F^{(S)}$ (force) and $\mathcal{T}^{(S)}$ (torque) are transformed from the sensor frame S into another frame A, whose x-axis points in the direction of the movement and the z-axis points towards the table.

The transformation from the world frame to the sensor frame is called $T_{S\leftarrow 0}$. It is provided by ROS but could alternatively be computed using the forward kinematics for the arm.

Given the start position v_s and the end position v_e of the wiping task, the transformation $T_{A\leftarrow 0}$ from the world frame to A can be calculated as follows (assuming that the z-axis of the world frame is orthogonal to the table).

$$x = \frac{v_e - v_s}{|v_e - v_s|}, z = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}, \quad y = \frac{z \times x}{|z \times x|}, T_{A\leftarrow 0} = \begin{pmatrix} x[0] & x[1] & x[2] \\ y[0] & y[1] & y[2] \\ z[0] & z[1] & z[2] \end{pmatrix} \tag{3.12}$$

The desired $F^{(A)}$ and $\mathcal{T}^{(A)}$ can then be computed by:

$$\begin{aligned} F^{(A)} &= T_{A\leftarrow 0} \cdot (T_{S\leftarrow 0})^{-1} \cdot F^{(S)} \\ \mathcal{T}^{(A)} &= T_{A\leftarrow 0} \cdot (T_{S\leftarrow 0})^{-1} \cdot \mathcal{T}^{(S)} \end{aligned} \tag{3.13}$$

3.4 Contact Event Detection and Classification in Time Series Streams

In this section, I will explain the inner workings of my algorithm in detail. I start off by describing the algorithm on a high abstraction level and follow up with listing different options for implementing the low level functions.

3.4.1 High-Level View on the Learning Algorithm

Before presenting the algorithm, I will explain how the intuition given in section 3.1 can be applied to multidimensional time series. The natural solution is to select multidimensional time series subsequences from all dimensions as shapelet candidates. However, events usually do not affect all dimensions evenly. For that reason *MTS* will only contain a subset of the possible dimensions. The *BMD* is therefore defined to only consider the distances in the dimensions of the *MTS* and report their average. For example, the *wipe_start* event only influences the x/z-axis of the force signal and the torque around the y-axis. Therefore, a good *MTS* will contain some of these dimensions. When searching for the best match all the other dimensions are ignored. This adds a new assumption, that the dimensions are always temporally aligned. A possible solution is the usage of DTW as the distance measure, but I will leave this problem unsolved for future research, because the force/torque readings come from the same sensor and have therefore no lag between them.

The original shapelet discovery algorithm described by Ye et al. [51], is a brute-force algorithm, where all time series subsequences are selected as shapelet candidates and tested. However, the dataset used here contains 520 6-dimensional training examples with an average length of approx. 300. This results in $\frac{300^2+300}{2} \cdot 520 \cdot (2^6 - 1) = 1,479,114,000$ candidates, which is a lot to say the least. Alternatively, Grabocka et al. [11] have presented a technique that learns the best possible shapelets and is therefore not limited to candidates from the dataset, but it is also to slow when facing my dataset.

I will therefore resort to speed up techniques for the brute-force algorithm, of which several have been proposed since the inception of shapelets. Some are more technical and focus on reuse of computations [34] and early abounding of e.g. distance calculations [51]. Others involve candidate pruning [34], [12] or time series compression [12], [40]. The single most effective technique is to limit the possible lengths *SL* for shapelet candidate. I will do with using Eq. (3.14), which is designed for easy parameter tuning.

$$SL = \left\{ \frac{sl_{max} \cdot i}{N_{max}} | i \in \{1..N_{max}\} \right\} \tag{3.14}$$

This introduces two parameters sl_{max} and N_{max}. The first parameter needs to be longer than the longest event and the second parameter should create a clear performance/training time trade-off, that converges quickly to maximum performance, while the overall number of candidates increases only linear, as N_{max} is raised. With $sl_{max} = 50$ (2 seconds, assuming a sampling rate of 25hz) and $N_{max} = 3$, Eq. (3.14) yields $[16, 33, 50]$, resulting in $(300 \cdot 3 - 16 - 33 - 50 + 3) \cdot 520 \cdot (2^6 - 1) = 26,339,040$ shapelet candidates. This equals a reduction by 98%, however, this number is still way too large. Therefore, the maximum number of dimensions that a *MTS* is allowed to use will be limited using the parameter dim_{max}. The assumption here is that events that produce unique shapes in all dimensions are rare and even if they exist, a lower amount could still be sufficient for them to be distinguished from other events.

The main loop of the algorithm is depicted in Alg. 1. It starts off by creating a dictionary C in which a binary classifier will be saved for every label key. In the next step all unique labels are identified, as well as all dimension subsets that are smaller than dim_{max}. Candidate shapelets

Algorithm 1: Main loop of the learning algorithm.

```
1  function find_shapelets(dataset $D$, list of possible shapelet lengths $SL$, maximum number of dimensions $dim_{max}$):
2  $C \leftarrow$ empty dictionary
3  $L \leftarrow \{l \mid (\exists (t, L_t, r_t) \in D)\ [l \in L_t]\}$                     // unique labels in $D$
4  $tmp \leftarrow \{x \mid (\exists (t, L_t, r_t) \in D)\ [x \in dim(t)]\}$                // e.g.$\{f_x, f_y, f_z, \tau_x, \tau_y, \tau_z\}$
5  $dims \leftarrow \{x \mid (\exists x \in \mathcal{P}(tmp))\ [x \neq \emptyset \wedge |x| \leq dim_{max}]\}$     // $\{\{f_x\},\{f_x, f_y\}...\}$
6  $shapelets, possible\_labels \leftarrow candidates(D, SL, dims)$
7  foreach $l \in L$ do
8      $D_{binary} \leftarrow \{(t, 0) \mid (\exists (t, L_t) \in D)\ [l \notin L_t]\} \cup \{(t, 1) \mid (\exists (t, L_t) \in D)\ [l \in L_t]\}$
9      $bsf\_ig$, $bsf\_q \leftarrow 0, 0$                      // best so far IG and secondary measure
10     foreach $s \in shapelets$ do
11         if $l \in possible\_labels[s]$ then
12             $D_s \leftarrow \{(BMD(s, t), l_b) \mid (\exists (t, l_b) \in D_{binary})\}$     // shapelet transformed $D$
13             $\delta, ig, q \leftarrow determine\_separation\_threshold(D_s)$
14             if $ig > bsf\_ig$ or ($ig = bsf\_ig$ and $q > bsf\_q$) then
15                 $C[l] \leftarrow S = (s, \delta, l)$
16                 $bsf\_ig \leftarrow ig$
17                 $bsf\_q \leftarrow q$
18 return C
```

are then generated using the allowed lengths and dimensions combinations. Additional pruning techniques are used in *candidates*, which will be presented in detail in section 3.4.4. In short, similar candidates are pruned using clustering and only one candidate per cluster is tested. The method *candidates* also returns a list of possible labels, that the shapelet could explain. Given the clustering based pruning, this would be all labels from the shapelet candidates' original time series that are summarized in the cluster that a remaining candidate represents.

Next, the algorithm creates the dataset D_{binary}, by replacing the list of labels L_t in D with a 1 or 0 depending on whether or not the label under investigation is contained in L_t. Every shapelet that can represent l will now be tested for its ability to do so. Using the *BMD*, D_{binary} is transformed into the shapelets 1-dimensional feature space, yielding D_s. In Line 13 a separation threshold δ is computed. Different options for this step will be explained in section 3.4.3. The quality of the split is measured using two numbers, one of them is IG (see Def. 18). A perfect information gain can be accomplished, if δ creates a clean split. This happens frequently, for example with two versions of the same shapelet, but of different length. In this case a secondary quality measure q is employed, for which there are also multiple options explained in section 3.4.3. A simple one is to use the gap between the closest points to δ in D_s. If the newly tested *MTS* is better than the best one so far, it will replace it.

To summarize, the proposed algorithm selects all subsequences from training examples as *MTS* candidates, reduces this number using various pruning techniques, and then determines the best *MTS* for each label by testing it for its ability to separate training examples with label from training examples without label.

Overall the algorithm will have the following parameters: N_{max}, sl_{max}, dim_{max}, d_{max}, σ_f and σ_τ, w_{ext}. Except for σ_f and σ_τ, all of them are used to reduce the massive amount of *MTS* candidates. The last four parameters will be discussed in the following subsections and all of them will be tested for their ability to prune unimportant shapelets first in section 4.2.

In the following subsections I will discuss different options for the functions *BMD*, *determine_separation_threshold* and *candidates*, starting with normalization techniques and distances metrics used in *BMD*.

3.4.2 Best Match Distance

The goal of the best match distance is to capture similarities in shape. However, if a distance measure is applied to two raw time series, any similarity in shape gets dominated by offset or scale differences. For example in Fig. 3.4 the two closest series are the ones in the middle, which have the most dissimilar shape. This picture also shows the visual interpretation of the manhattan distance, it is the sum of the distances between pairs of points. As a countermeasure,

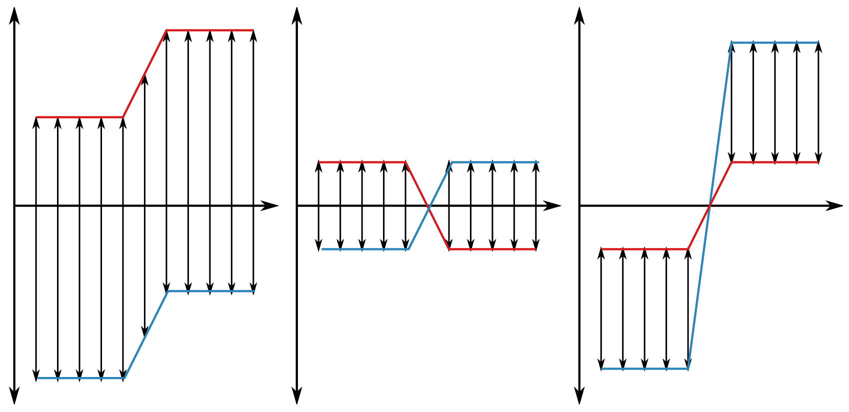


Figure 3.4 This figure shows that scale and offset differences can dominate any similarity in shape.

z-normalization was used in almost all of the shapelet literature that I reviewed and those who did not, ignored the problem. The z-normalization sets time series to a mean of zero and standard deviation of one. If all time series in Fig. 3.4 are z-normalized, then they would all look exactly the same, except for the blue one in the middle, which is therefore now the most dissimilar one to the others. For that reason the *BMD* normalizes the shapelet and every subsequence of the tested time series.

However, setting a time series, which is almost a horizontal line, to standard deviation 1 just amplifies the noise in it, thus creating a random shape. This has two disadvantages, on one hand, a shapelet can now have random good matches in an input time series. On the other hand random shapelets can make pruning techniques difficult, that try to filter out similar candidates. This problem was already addressed in [44] in a different context. The solution is to only set time series to standard deviation 1, if their old standard deviation exceeds a threshold σ_{min}. This threshold is dependent on the dataset, but could be estimated by inspecting the datasets noise level. Furthermore, if a time series is multidimensional, multiple σ_{min} thresholds are needed because the dimensions might describe totally different things. The dataset used in this thesis has six dimensions but the three force and torque dimensions just represent different axis of the same measured quantity. Therefore, two thresholds are sufficient which will be called σ_f and σ_τ, respectively. They can be estimated by calculating the standard deviation of short subsequences from a free space movement. This method suggests 0.4 for the force signal and 0.1 for the torque signal (rounded up) on sequences of length 16. I chose 16 because short sequences have higher standard deviations and 16 is the smallest shapelet length when using $N_{max} = 3$ and $sl_{max} = 50$.

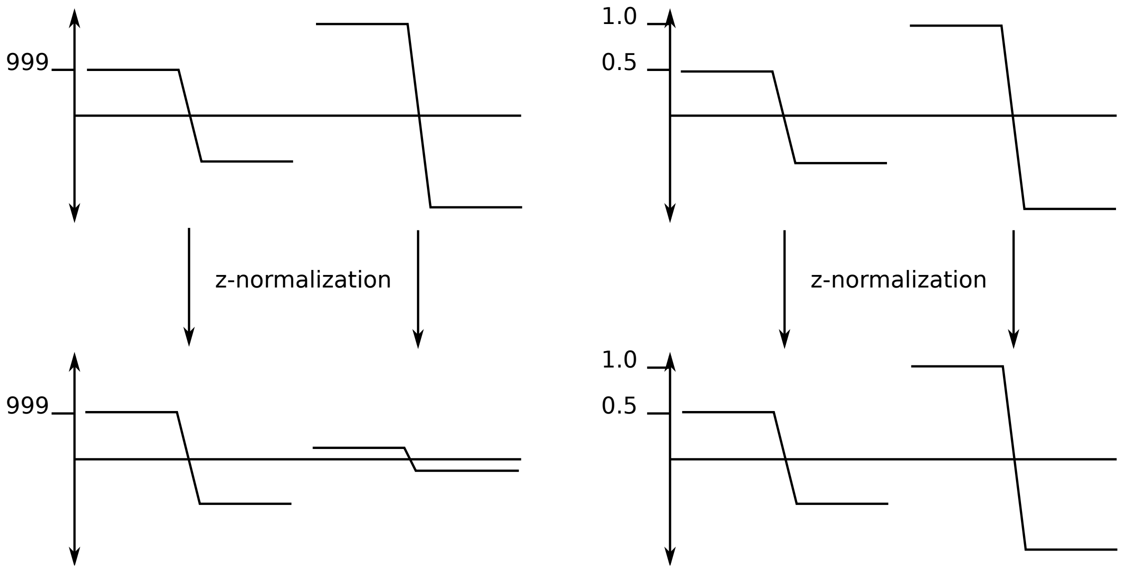


Figure 3.5 A (not true to scale) visualization of the problem that if σ_{min} is above 1, the normalized time series will have a lower standard deviation than a non-normalized one.

However, as you can see in Fig. 3.5, if σ_{min} has to be bigger than 1, the normalized time series will have a higher standard deviation than the non-normalized ones. This can be avoided by replacing time series with a low standard deviation with the 0-vector, $\vec{0}$.

As a distance measure for *BMD*, almost exclusively, length normalized euclidean distance was used which is described in Eq. (3.15).

$$\sqrt{\frac{1}{len(t_x)}}(||t_x - t_y||_2) \tag{3.15}$$

The length normalization (i.e. multiplication with $\sqrt{\frac{1}{len(t_x)}}$) ensures that the distances are comparable between time series of different lengths [40]. Even though not explicitly pointed out in any shapelet related paper that I have reviewed, it can be proved that the normalized euclidean distance of two z-normalized time series can not be greater than 2.

Corollary 1. *If $x, y \in \mathbb{R}^n$ and*

$$\mu(x) = \mu(y) = 0 \tag{3.16}$$

and

$$\sigma(x) = \sigma(y) = 1 \tag{3.17}$$

then $\sqrt{\frac{1}{n}}(||x - y||_2) \leq 2$.

An alternative for the euclidean distance is the manhattan distance, which can be length normalized as well (3.18).

$$\frac{1}{len(s)}(||t_x - t_y||_1) \tag{3.18}$$

A similar property can be achieved when using the length normalized manhattan distance.

Proposition 1. *If $x, y \in \mathbb{R}^n$ and*

$$\mu(x) = \mu(y) = 0 \tag{3.19}$$

and

$$\sigma(x) = \sigma(y) = 1 \tag{3.20}$$

then $\frac{1}{n}(||x - y||_1) \leq 2$.

Both proofs are in the appendix A.9. This also holds true, if one of the time series is $\vec{0}$ (the normalization special case), because the manhattan and euclidean distance are both metrics and satisfy the triangular inequality.

To understand this intuitively, we have to view a time series of length n as a point in $\mathbb{R}^n$, where every of the series points corresponds to one of the n coordinates. To make this easier I will discuss the special case, when a time series is replaced with $\vec{0}$, later. Now imagine a time series of length 3 as a point in $\mathbb{R}^3$. For example Fig. 3.6 depicts a few random points in yellow. If the mean of these points is subtracted from them (blue points), they are projected onto a plane, which is orthogonal to the $\vec{1}$ vector. This also holds true for higher dimensions.

Proposition 2. *If $x \in \mathbb{R}^n$ and*

$$\mu(x) = 0 \tag{3.21}$$

then x is orthogonal to $\vec{1}$.

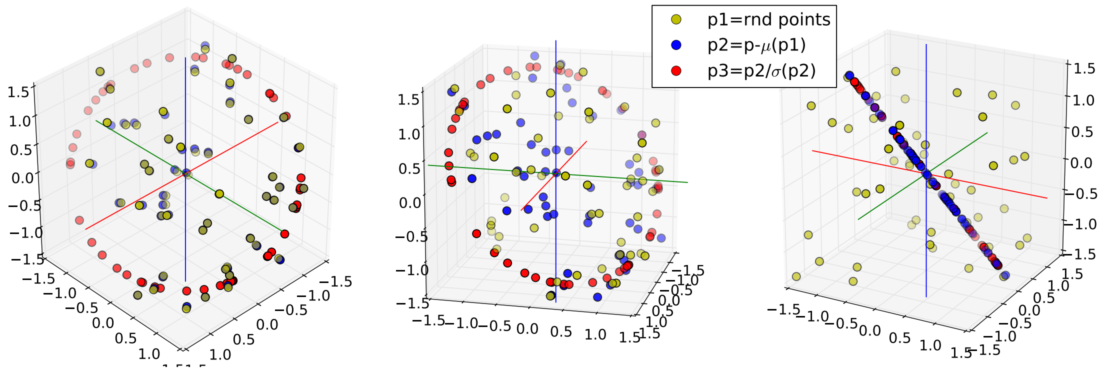


Figure 3.6 Three perspectives of the same random z-normalized points in 3d-space.

Proof.

$$x \bullet \vec{1} = 1 \cdot x_1 + ... + 1 \cdot x_n \overset{(3.21)}{=} 0 \tag{3.22}$$

□

If the 3d time series are now divided by their standard deviation (red points), they all end up with a distance of $\sqrt{3}$ to the origin, thus forming a circle. This is because the standard deviation of a time series with zero mean is equal to $\sqrt{\frac{1}{len(t_x)}(||t_x||_2)}$.

To summarize, the z-normalization first projects points from $\mathbb{R}^n$ onto a (n-1)-dimensional hyperplane. The division by standard deviation projects points from that plane onto the (n-2)-dimensional surface of a hypersphere, which lies inside the hyperplanes subspace. The radius of that sphere is $\sqrt{n}$. If the distance is now measured using the length normalized euclidean distance, $\sqrt{n}$ gets canceled out due to the $\sqrt{\frac{1}{n}}$ factor. Therefore, every point has a distance of 1 to the $\vec{0}$ (the normalizations special case) and a distance of 2 to the point on the other side of the sphere.

In fact z-normalized points are a special case for Theorem 1 with $c = \sqrt{\frac{1}{n}}$, $d = 1$ and $p = 2$.

Theorem 1. *If $x, y \in \mathbb{R}^n$, $c, d \in \mathbb{R}$ and $p \in \mathbb{R}^+$ such that*

$$c||x||_p = c||y||_p = d \tag{3.23}$$

then $c(||x - y||_p) \leq 2d$.

This proof is in the appendix as well (A.9). I will refer to this as the ≤ 2-property. The *BMD* of the multidimensional time series is also ≤ 2 because the *BMD* averages the distances from the individual dimensions. The requirement for Theorem 1 can be satisfied much more easily by choosing $c = d = 1$ and $p = 2$. Therefore we can remove unnecessary $\sqrt{\frac{1}{n}}$ computations. In fact every p with $c = d = 1$ satisfies Theorem 1. This opens up many new forms of normalization and distance metric combinations. I hypothesis, that all of them exhibit a different behavior.

To test this, the n_2-normalization + euclidean distance will be compared to the n_1-normalization + manhattan distance.

3.4.2.1 Angular Distance

As shown in the previous section, n_2-normalization creates a hypersphere, it might be a good idea to calculate the distance on that spheres surface, instead of through it, like the euclidean distance does.

The distance of two points on the surface of a sphere can be described by an angle. The cosine of this angle can be calculated using the dot product of the vectors that point from the origin to the points on the spheres surface. This angle interpretation also makes sense in higher dimensions, since this is the angle between both vectors on the 2d plane, that stay span. The angular distance, also called cosine similarity, cosine distance or angular similarity, is defined in Eq. (3.24).

$$ad(x, y) = 1 - \frac{x \bullet y}{||x||_2 \cdot ||y||_2} \tag{3.24}$$

It returns values from $[0, 2]$, where 0 represents an angle of 0 and 2 an angle of π radian. Equation (3.24) can be shortened to (3.25), if the shapelets are n_2-normalized, since they have a length of 1. As a side note, this means that the angular distance has inbuilt scale invariance, even if two input time series are not normalized.

$$ad(x, y) = 1 - x \bullet y \tag{3.25}$$

However, there is the special case where one or both of the shapelets are $\vec{0}$. The original definition (3.24) is undefined in this case because of a division by 0, but the second one yields 1. This would make sense, if only one of the inputs is $\vec{0}$, but we have to add a special case, to set the result to 0, if both are $\vec{0}$.

$$ad(x, y) = \begin{cases} 0, & \text{if } x = y = \vec{0} \\ 1 - \frac{1}{n}(x \bullet y), & \text{otherwise} \end{cases} \tag{3.26}$$

Here you can see another advantage of replacing shapelets with low standard deviation with $\vec{0}$. If they still had some short length, the original angular distance (3.24) would project them onto the spheres surface as well, thus reintroducing the problem of noise amplification. It is also worth noting, that the angular distance is always smaller or equal to the euclidean distance, but the special case of the straight line is relatively closer to any point in the euclidean distance. A big advantage is also, that the angular distance is faster to compute. In fact, it is very efficient to compute a distance matrix between two sets of time series by combining each set to a matrix and using the normal matrix multiplication.

To the best of my knowledge, the angular distance has not been used in combination with shapelets. However, it is relatively common in text clustering or classification. For time series classification overall Senin and Malinchik [44] have utilized the angular distance by first transforming the series into a text representation using SAX. The angular distance was also used by Ko et al. [24] for 1-NN classification of time series. But they did not justify this choice, apart from showing that it works better than euclidean distance (which I am assuming is thanks to the inbuilt scale invariance) in combination with DTW. Surprisingly the angular distance was not mentioned in the frequently cited comparison of time series similarity measures by Ding et al. [7].

3.4.2.2 Summary

To summarize, the most frequently used combination for *BMD* is the z-normalization in combination with length normalized euclidean distance. This is however just a special case for Theorem 1 and the resulting distances are equal to the n_2-normalization in combination with the normal euclidean distance. Theorem 1 also suggests, that are other valid normalization-metric combinations, based on the different p-norms. Furthermore, the geometric interpretation of the n_2-normalization is a projection onto a hypersphere. The distance on the surface of that sphere can be calculated with the angular distance. In total I will compare the following normalization/metric combinations for the *BMD* in the evaluation in chapter 4.

1. n_1-normalization with manhattan distance
2. n_2-normalization with euclidean distance
3. n_2-normalization with angular distance

For brevity, I will refer to these combinations only with the names of the distance metrics. It can be assumed, that shapelets and time series subsequences are always normalized with the respective function.

3.4.3 Determining δ for multidimensional time series shapelets

In Fig. 3.7 you can see an example for a *MTS* and the shapelet transformed training dataset D_s. All training examples with the event label are red and the others are blue. The *BMD* between a shapelet and a time series is almost never bigger than 1, as long as the time series has a horizontal line section because of the ≤ 2-property proven above. Now a value for δ has to be determined that best separates both classes. The quality of a split is measured by its information gain. The

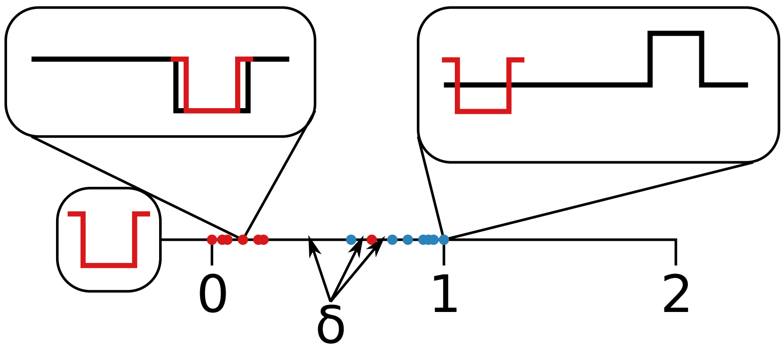


Figure 3.7 Example for a 1-dimensional feature space of a *MTS* and three intuitively feasible options for choosing a separation threshold δ. Red dots are the *BMD* between a *MTS* and multi. dim. time series that have the label for which the current shapelet is a candidate. Blue dots are distances to time series who do not have the label.

IG compares the entropy of a set before and after the split. For example, there are 7 red dots and 7 blue dots depicted in Fig. 3.7, which equals an entropy of 1. The left most marked δ creates two new sets, one with 6 red dots and one with 1 red and 7 blue dots. Both sets have a lot less entropy with 0 and 0.54, respectively, resulting in an IG of approx. 0.69. The second marked δ produces a set with 6 red dots and 1 blue dot on its left side and a second set with 1 red and 6 blue dots on its right side. In this case both new sets have entropy left (0.59) and the IG is approx. 0.4. The third δ creates the same information gain as the first because it is the same situation but with switched colors. Hence, we have a draw, but intuitively the first δ would be preferable because the true classes are further away from it.

However, there are infinitely many values for δ possible because the shapelets feature space is continuous. In the following subsections I will present three techniques that deal with this problem. The first two are from the literature and the last one exploits the ≤ 2-property of the feature space.

3.4.3.1 *max-IG-δ*

This approach is the most used in literature and always results in the split with the highest information gain. It will be called *max-IG-δ*. A visual intuition is depicted in Fig. 3.8 and the algorithm for computing it is shown in Alg. 2. To find the split yielding the highest IG, all points in the middle of two training examples are tested, since they are the only ones that result in a different IG. The secondary quality measure q (line 11) describes the distance between both classes because a huge gap indicates a better separation.

This approach has two main disadvantages. On one hand, it is relatively time consuming to calculate the information gain $|D_s| - 1$ times. On the other hand, the split with the highest IG

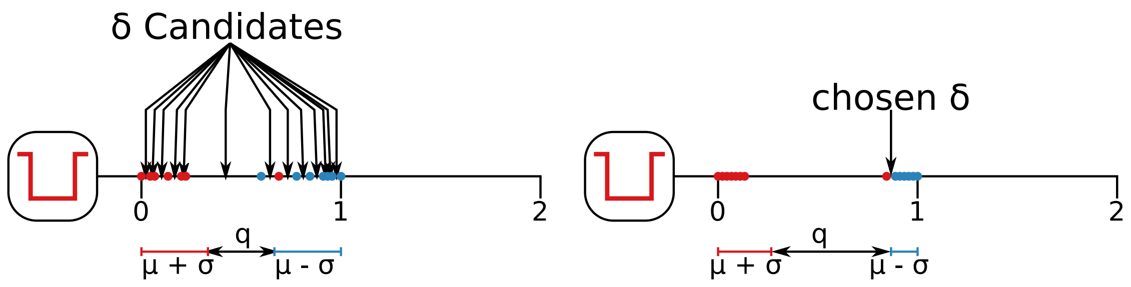


Figure 3.8 (Left) Visual intuition of the *max-IG-δ* method. All points that result in a different IG are tested. (Right) A case in which both the information gain and q are good, but δ is not.

Algorithm 2: Learning δ using the *max IG* method.

```
1  function determine_separation_threshold(shapelet transformed training dataset $D_s$):
2  $D_s \leftarrow sort\_ascending(D_s)$        // $(d_i, \_) = D_s[i] > (d_j, \_) = D_s[j]$ if $d_i > d_j$
3  $bsf\_\delta \leftarrow 0$                    // best so far separation threshold
4  $bsf\_ig \leftarrow 0$                        // best so far IG
5  $bsf\_q \leftarrow 0$                         // best so far secondary quality measure
6  foreach $i \in \{0, ..., len(D_s) - 2\}$ do
7      $\delta \leftarrow \frac{D_s[i].d + D_s[i+1].d}{2}$
8      $ig \leftarrow IG(D_s, \delta)$
9      $D_1 \leftarrow \{d \mid (\exists(d, l_b) \in D_s)\ [l_b = 1]\}$      // red dots
10     $D_0 \leftarrow \{d \mid (\exists(d, l_b) \in D_s)\ [l_b = 0]\}$      // blue dots
11     $q \leftarrow \mu(D_0) - \sigma(D_0) - (\mu(D_1) + \sigma(D_1))$
12     if $ig > bsf\_ig$ or $(ig = bsf\_ig$ and $q > bsf\_q)$ then
13         $bsf\_\delta \leftarrow \delta$
14         $bsf\_q \leftarrow q$
15         $bsf\_ig \leftarrow ig$
16 return $bsf\_\delta, bsf\_ig, bsf\_q$
```

is not necessarily the best split, as you can see in the second picture of Fig. 3.8. A single outlier or mislabeled training example can result in a bad δ.

3.4.3.2 *KDE-δ*

To avoid the susceptibility to outliers of the *max-IG-δ* method, Xing et al. [50] proposed to choose δ based on the estimated probability distributions as depicted in Algorithm 3. This method will be called *KDE-δ*. Here, the prior probabilities of both classes are first determined and then the probability density distribution is estimated using kernel density estimation (KDE). This can then be used to calculate the probability ($P_1(d)$) of a multidimensional time series with *BMD* d

to the *MTS*=$(s, ?, l)$ to have label l.

$$P_1(d) = \frac{p_1 f_1(d)}{p_0 f_0(d) + p_1 f_1(d)}, \quad d = BMD(s,t) \tag{3.27}$$

Algorithm 3: Learning δ using the KDE method.

```
1  function determine_separation_threshold(shapelet transformed training dataset $D_s$):
2  $D_1 \leftarrow \{d \mid (\exists (d, l_b) \in D_s)\ [l_b = 1]\}$
3  $D_0 \leftarrow \{d \mid (\exists (d, l_b) \in D_s)\ [l_b = 0]\}$
4  $f_1 \leftarrow KDE(D_1)$
5  $f_0 \leftarrow KDE(D_0)$
6  $p_1 \leftarrow |D_1| \ / \ |D_s|$
7  $p_0 \leftarrow |D_0| \ / \ |D_s|$
8  $\delta\_candidates \leftarrow \{d \in \mathbb{R} \mid P_1(d) = 0.5\}$
9  $\delta \leftarrow \underset{\delta' \in \delta\_candidates}{\operatorname{argmax}} IG(D_s, \delta')$
10 return $\delta, IG(D_s), -f_1(\delta)$
```

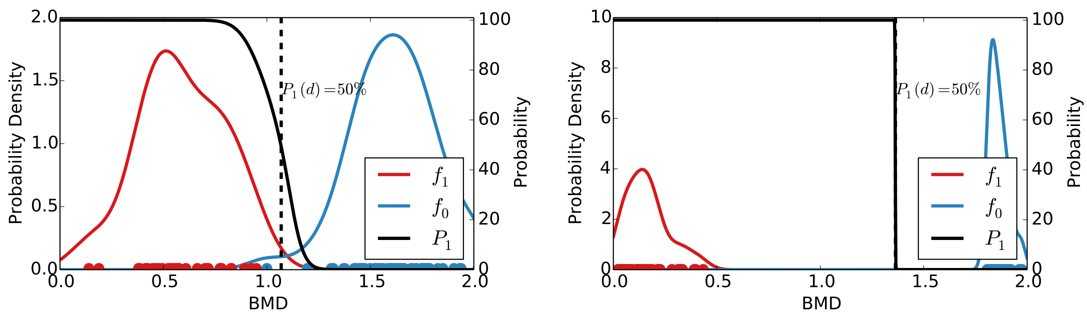


Figure 3.9 (Left) The distance threshold δ is chosen such that a multidimensional time series with a *BMD* of d has a 50% chance to have the label l. $P_1(d)$ is calculated using the estimated probability densities $f_1(d)$ and $f_0(d)$. (Right) If both classes are very far apart, this method can yield δs that are too large.

Next, every d with a 50% probability will be tested for its information gain and the best one is chosen for δ. An example is shown in Fig. 3.9.

Even though KDE also takes its time to compute, this cost should be outweighed by the amount of saved IG computations. It can also handle outliers much better than *max-IG-δ*. Ironically the main problem here is when both classes are extremely well separated (Fig. 3.9 right), because both f_1 and f_0 are almost 0 in the middle. Now simple rounding can shift δ by a lot. There is also a special case caused by the normalization, where a shapelet has a distance of 1 to all time series with label 0 or 1, thus making KDE undefined. This is handled by replacing f_1 or f_0 with a Gaussian distribution that has a very small variance.

As a secondary quality measure, $-f_1(\delta)$ will be used because it correlates with a large gap between the real classes and does not need additional computations.

3.4.3.3 *fixed-δ*

The third approach is to use a fixed δ for every shapelet and will thereby be called *fixed-δ*. The inspiration for this comes from the fact, that the distances are always between 0 and 2 and that horizontal lines has a distance of 1 to every non horizontal line. This knowledge can be used to narrow down the possible values for δ. Because of the second observation δ should always be smaller than 1, it should probably even be smaller than 0.5 as it lies in the middle. Furthermore *MTS* that have some $\vec{0}$ dimensions are even closer to the horizontal line.

Now consider the two shapelets depicted in Fig. 3.10. They have a manhattan and euclidean distance of approximately 0.5. If both shapelets represent different classes, than the feature space of the red shapelet will look similar to Fig. 3.10 (right). To separate between them, a δ of around .3 should do well. I hypothesis that this threshold is at least decent for all events. This number is also only very critical, if both classes are very close in the shapelets feature space, which means that the shapelet is probably bad anyway. Additionally, if δ is fixed, than the algorithm just searches for shapelets, for which that threshold is good.

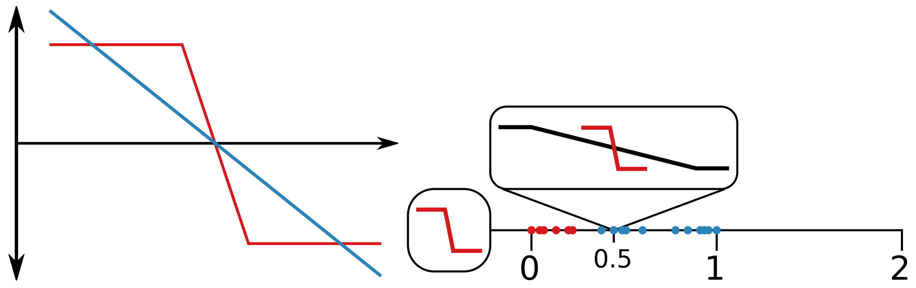


Figure 3.10 (Left) Two time series subsequences that have a manhattan and euclidean distance of approx. 0.5 and an angular distance of approx. 0.14. (Right) An example feature space with the red subsequence as a shapelet candidate. A time series that has the blue subsequence in it would have at least a (manhatten) distance of 0.5.

This method also gives the user some the power to control how likely the classifers are to detect FP. If it is important, that we do not have any FP, δ could be set to a low value and the algorithm finds shapelets for which such a low value is best.

Even if this technique results in worse classification performance, it will be significantly faster than the previous to methods, because only one IG calculation is needed. As a secondary quality measure, the same can be used as for *max-IG-δ*.

3.4.3.4 Summary

Three techniques to determine δ have been proposed. The first one, *max-IG-δ*, searches for the δ with the highest information gain and therefore requires a high computation time. The second one, *KDE-δ*, chooses δ based on the probability distributions of both classes and is faster to compute. The third one, *fixed-δ*, uses the same δ for every shapelet because the range for feasible values is greatly reduces as a consequence of the ≤ 2 property.

3.4.4 Candidate Pruning

To cut down on the huge amount of shapelet candidates, Algorithm 4 is utilized, which consists of three major steps. First, all training examples are searched for relative extrema in all dimensions and their derivatives. Then all candidates are removed which only contain zeros, because these are quite often due to the normalization and because they can not describe an event. And lastly the remaining candidates are clustered and the center of each cluster is used as a shapelet candidates. For each shapelet the information which labels have occurred in its cluster is also saved, that way we do not have to test every shapelet for every label.

Algorithm 4: Calculation of candidate shapelets.

```
1  function candidates(dataset D, shapelet length len, dimensions dims):
2  $tmp \leftarrow$ empty sequence
3  $shapelets \leftarrow$ empty sequence
4  $possible\_labels \leftarrow$ empty sequence
5  foreach $(t, L_t) \in D$ do
6      $extrema \leftarrow find\_relative\_extrema(t, dims, w_{ext})$   // Step 1: shapelets around extrema
7      foreach $e \in extrema$ do
8          $candidate \leftarrow normalize(subseq(t, len, dim, e))$
9          if $(\exists x \in candidate)\ [x \neq 0]$ then
10             $tmp \leftarrow append(tmp, candidate)$   // Step 2: no horizontal lines
11 foreach $s \in cluster(tmp)$ do
12     $shapelets \leftarrow append(shapelets, s)$   // Step 3: only cluster centers
13     $possible\_labels \leftarrow append(possible\_labels, \{l \mid (\exists(t, L_t) \in D)\ [l \in L_t \wedge BMD(s, t) \leq d_{max}]\})$
       /* All labels which can be found in the cluster of $s$. */
14 return $shapelets, possible\_labels$
```

3.4.4.1 Relative Extrema

The goal of the function *find_relative_extrema* is to remove different perspectives of the same candidates. It gets a multidimensional time series, a list of dimension names *dims* and a parameter w_{ext} as input.

Different perspectives of the same shapelet are removed by only returning those, who are centered around extrema. For example, Fig. 3.11 shows one dimension of a time series from the dataset. As you can see, using every local extremum would result in a lot of candidates due to noise. But using only global extrema could miss some good candidates. Therefore, every dimension listed in *dims* of the input series will be searched for relative extrema in a range of a $\pm w_{ext}$-window. This is a new user specified parameter, but its exact value should not be very critical because the events should result in the strongest extrema, thus remaining till the end. It is also to be expected, that the biggest performance gain is achieved while w_{ext} is relatively small. Additionally, each dimensions' derivative is searched of relative extrema, to capture high gradients. Fig. 3.11 shows the, subjectively, optimal result for the example time series using $w_{ext} = 50$ (two seconds).

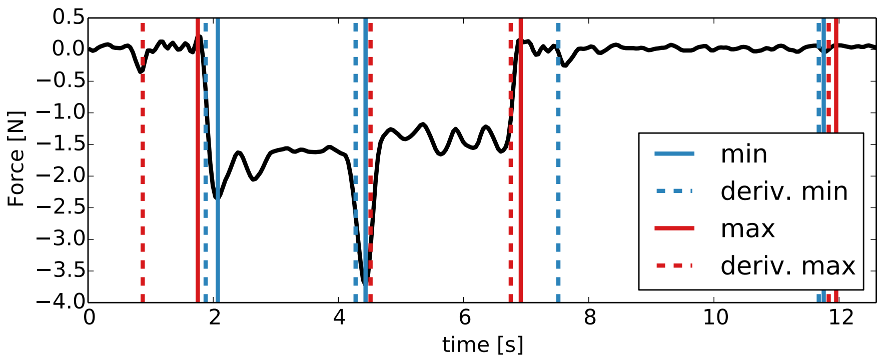


Figure 3.11 Detected relative extrema of a training example from the dataset using $w_{ext} = 50$.

3.4.4.2 Clustering

The extrema pruning of the previous step returns a lot of shapelet candidates that look similar, because they are all centered around extrema. A clustering algorithm seems to be the perfect fit for reducing this candidate set. All of these candidates stem from time series, which turns that into a subsequence time series clustering problem, which is usually meaningless as proven by Keogh et al. [22]. However, clustering will be meaningful in this situation for two reasons. Firstly, the subsequences are normalized which replaces some of them with the $\vec{0}$ vector. This removes the property, that the mean of all subsequences will be a straight line. Secondly, some of the data is ignored, as stated by Rakthanmanona and Keogh et al. [41], is necessary to get meaningful clusters. Especially the extrema pruning removes this „averaging" problem.

There are a lot of clustering techniques which could be employed here. A good overview of popular clustering methods and their effect on 2d datasets can be seen in Fig. 3.12. Since there is "no free lunch" in machine learning [8], we have to consider the pros and cons of the different

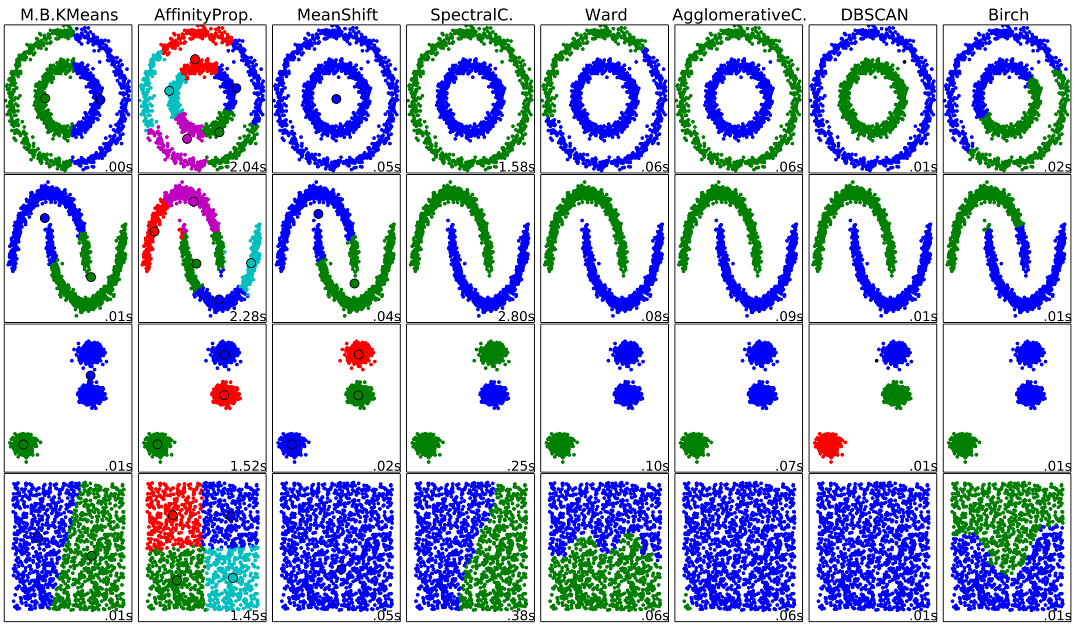


Figure 3.12 Effect of different clustering techniques on a 1500 sample artificial dataset. Source: *http://scikit-learn.org/stable/modules/clustering.html*, but recomputed on a 4.0 GHz CPU, for a fair runtime comparison.

techniques. The optimal clustering solution has to have the following properties. First, it has to be fast, otherwise it can not speed up the learning. It should also not require the number of clusters as an input, since we do not have this information. Additionally, the created clusters should not be to big. If the shapelet candidates are evenly spread over the space, a few single clusters are preferable to a single big one.

With that in mind, affinity propagation produces the best clusters because it splits connected areas and does not require the number of clusters as input. However, it is by far the slowest and therefore unfitting. K-means and Ward also split evenly distributed areas, but require the number of clusters as input parameter. This leaves us with birch, which does not perform well in high dimensional space.

There is a version of k-means called x-means, that does not require the number of clusters as input [37]. However, the k estimation would merge for example the two clusters in the bottom left of Fig. 3.12.

Instead, I will be using a modified version of the algorithm proposed by Grabocka et al. [12], which was used to prune shapelet candidates as well. They have seeded cluster centers at randomly chosen points and all shapelets closer than some parameter belonged to that cluster. This parameter will be called d_{max} and prevents clusters from becoming too big. Given the fact that two shapelets can not be further apart than 2, this parameter should be easy to tune.

The original algorithm has the problem, that cluster centers can be seeded everywhere, since they are randomly chosen. This design decision makes this algorithm prone to bad luck, if candidates from the edge of a real cluster are selected. To fix this problem I propose Algorithm 5. Where

Algorithm 5: Clustering to prune candidate shapelets.

```
1 function clustering(list of multi. time series $T$, cluster radius $d_{max}$):
2 $centers \leftarrow \emptyset$
3 $outs \leftarrow T$
4 while $out \neq \emptyset$ do
5     $ins \leftarrow \{t \in outs \mid dist(t, outs[1]) \leq d_{max}\}$
6     $centers \leftarrow centers \cup \{NN(mean(ins), outs)\}$
7     $outs \leftarrow \{t \in outs \mid (\forall c \in centers)\ [dist(t, c) > d_{max}]\}$
8 return centers
```

dist is the distance measure used for *BMD*. In Line 5 a cluster with random center is created, but the center is then shifted to the nearest neighbor of the mean of that cluster. The mean of multiple multidimensional time series is computed using Eq. (3.28).

$$mean_x(T)[i] = \sum_{t \in T} t_x[i]/|T|. \tag{3.28}$$

This step could theoretically be repeated, until the new center converges, but one step is enough to greatly reduce the impact of randomness and keeps this algorithm fast.

This algorithm terminates because the amount of candidates, that are not in a cluster (*outs*), shrinks by at least 1 in every loop. Additionally, this ensures that the individual dimensions still lie on the surface of the hypersphere (depending on the distance and norm functions used), which is not true for the mean cluster center.

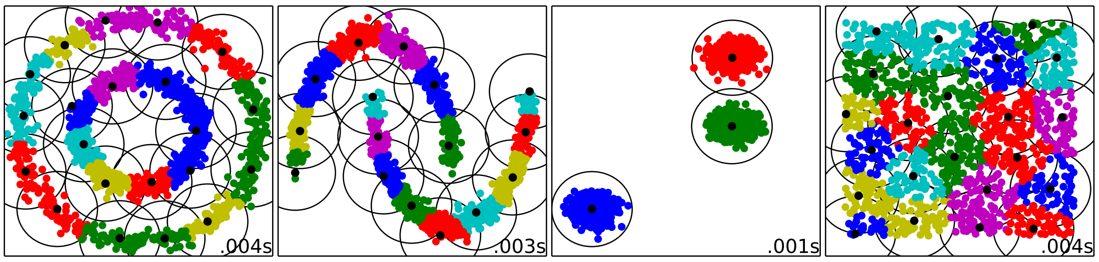


Figure 3.13 The effect of the proposed clustering on the same 2d artificial datasets that were used in Fig. 3.12. Sometimes two neighboring clusters have the same color because the graph coloring problem was not the focus of my thesis.

In Fig. 3.13 you can see how this clustering approach behaves on the datasets used in Fig. 3.12. It is significantly faster than all of the other presented clustering techniques. It is also able to center around isolated islands in the feature space, but breaks evenly distributed areas into smaller chunks.

In the context of our problem, it is worth noting that d_{max} should be smaller than 2. In fact, it should also be smaller than 1 because that is the distance from every shapelet to $\vec{0}$. A single cluster with $\vec{0}$ at its center would therefore enclose every other point. However, since those candidate are filtered out beforehand, values $d_{max} > 1$ still yield multiple clusters. In the end the last remaining clusters will have the shape, that is the most common in the dataset (other than $\vec{0}$). It is therefore likely, that the most common labels can still be learned. Using the same reasoning as in section 3.4.3.3, it is likely that a good threshold will be approx. 0.5 for the euclidean and manhattan distance and 0.14 for the angular distance because this is the distance between the to example shapelets depicted in Fig. 3.10.

Another interesting aspect is that the maximum number of clusters should be finite because the surface of a sphere is finite, but the exact number is difficult to compute, since the surface area of a hypersphere reaches its maximum in 7 dimensions and then quickly converges to zero.

3.5 Online Event Detection and Classification

Now that we have trained *MTS* for every event, we can use them to detect events online in a multidimensional time series t. To detect multiple event occurrences I exploit the fact, that input time series t for *BMD* can be as small as the shapelet itself. Hence, the distance between the shapelet and every subsequence of t with the length of S is calculated. As depicted in Alg. 6 the new time series $d_{t,S}$ contains the minimum of δ and *BMD*(s,t). If a *MTS* does not detect any events, $d_{t,S}$ does not have a minimum because it is a horizontal line, but at any detection there will be an area below this line with a minimum. To prevent matches in close proximity, it has to be a relative minimum in an area the length of the shapelet.

Algorithm 6: Detection of contact events using shapelets.

```
1 function detect(multidimensional time series $t$, MTS $S = (s, \delta, l)$):
2 $d_{t,S} \leftarrow$ empty sequence
3 foreach $u \sqsubseteq_{(len(s),dim(s))} t$ do
4     $d_{t,S} \leftarrow \text{append}(d_{t,S}, \min(\delta, BMD(s, u)))$
5 return $relative\_minima(d_{t,S}, len(s)) + len(s)/2$
```

Figure 3.14 visualizes this algorithm with an example. The top picture depicts the time series t, the middle plot shows the shapelet, that we classify with, and the bottom figure visualizes the distance time series $d_{t,S}$ and the detected time index. There is a time span from approximately 70 to 80, with a *BMD* lower than δ. Because we detect contact events with the relative minima operation, only one time index is selected. Half the shapelets length will be added to the detected time stamp, such that the detected point in time is in the middle of the match, instead of its starting position, because the starting position depends on the length of the shapelets.

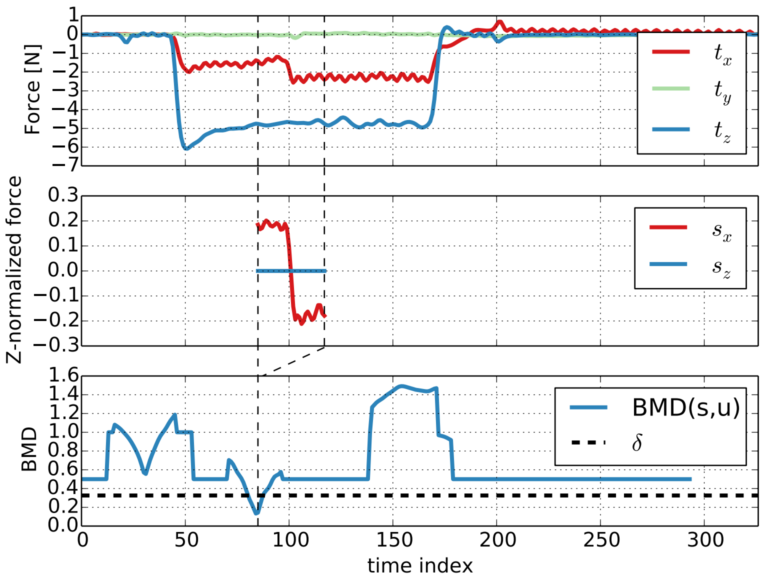


Figure 3.14 Visualization of Algorithm 6 that detects contact events of type l in a multidimensional time series t using a shapelet $S = (s, \delta, l)$.

To detect events online, we have to keep track of a window with the latest sensor readings, which has to be the size of the longest shapelets. It is also necessary to save $d_{t,S}$ for all *MTS*, here a few seconds worth of data should be enough to localize a minimum. We then apply Alg. 6 for every new incoming measurement. It is important to note, that a relative minimum will only be detected if the preceding and following point are higher. That means, if the current $d_{t,S}$ ends with the lowest point, it is not a minimum. Hence, events are not detected too early. This algorithm should also be very fast, only one distance measurement per sensor reading per shapelet has to be computed, which can also be done simultaneously.

The biggest disadvantage of this approach is that an event is only detected, when the full shapelet has a good match, thus making short shapelets preferable.

Algorithm 6 is a simplified version of the technique to find multiple matches in a time series stream, which was utilized in [24]. The authors classified with a 1-NN algorithm that considers one example per class and used DTW as a distance measure. Since they have also learned a threshold which serves the same function as δ (but for all classes) their *trained* system is very similar to the one proposed here. The main difference is the training phase because they need perfectly extracted training examples for every class. Their online detection and classification technique is also very complex and computationally expensive due to the usage of DTW.

As previously mentioned, they have also tried to use angular distance for DTW internally, but without giving the reason. For comparison, that means they can deal with temporal difference, because of DTW, and with scale differences, due to the angular distance and I can achieve scale and offset invariance because of the normalization used in the *BMD*. Using DTW as well might be a topic for future research.

Chapter 4

Evaluation

4.1 Evaluation Methodology

In this section I will present the evaluation method used to test the trained models ability to detect and classify the exact time of event occurrences. Most of the literature that I reviewed for this thesis defines TPs as the number of correctly classified instances. However, as you can see in Fig. 2.2, this is anything but trivial, when trying to detect and classify events in time series streams. The definition of „number as correctly classified instances" is therefore not specific enough.

Most of the reviewed algorithms used segmentation to preselect relevant subsequences, it seems therefore reasonable that the authors have assigned an event label to its closest segment. As a result, matches in close proximity do not happen, which resolves many problems. If this is true, a TP would be counted, if the right label is assigned for the segment and the negatives are all segments, that do not contain events. This method is overly optimistic because the tested system helps to evaluate itself, by providing the segments.

I, however, argue that the algorithm under evaluation should be treated as a black box. Algorithm 7 describes how TPs can be defined to achieve this. First, the event occurrences in a time series

Algorithm 7: Calculation of true positives TP.

```
1 function TP(multi dim. time series $t$, its list of real events $r_t$, class label $l$, MTS $S = (s, \delta, l)$):
2 $p_t \leftarrow detect(t, S)$
3 $TPs \leftarrow$ empty dictionary
4 foreach $r'_t \in r_t(l)$ do
5     $p'_t \leftarrow \underset{p_t \in p_t(l)}{\operatorname{argmin}} |p_t - r'_t|$
6     if $|p'_t - r'_t| < \Delta t_{max}$ and $|p'_t - r'_t| < |p'_t - TPs[p'_t]|$ then
7         $TPs[p'_t] \leftarrow r'_t$
8 return $|TPs|$
```

from the test set are predicted using *detect*. Then the nearest neighbor in the predictions for each real event in r_t is identified. If this distance is below a threshold Δt_{max}, it is a true positive. However, every prediction is only allowed to account for a single labeled event occurrence. Given this TP definition, we can now define FP, false negative (FN) and TN.

$$FP(t,l) = p_t(l) - TP(t,l) \tag{4.1}$$

$$FN(t,l) = r_t(l) - TP(t,l) \tag{4.2}$$

$$TN(t,l) = len(t) - r_t(l) - FP(t,l). \tag{4.3}$$

I have decided to use a relatively high value of 2 seconds for Δt_{max} and to record the time differences as well. That way it can be determined, if a classifier always detects its event too early (negative difference) or late (positive difference) with a very low variance. This additional insight comes with the cost of to little FN and to many TP. Therefore, the average time difference (μ td) for true positives as well as its standard deviation (σ td) will always be reported whenever I reason about a good performance. The reader can then decide whether the differences are acceptable or not.

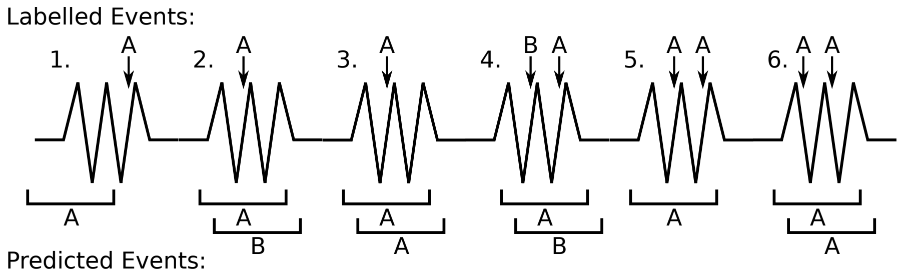


Figure 4.1 Problematic situations that arise when assigning TP and FP. The window over the predictions indicates a range of $\pm\Delta t_{max}$.

To get an intuitive understanding for the proposed TP definition, Alg. 7 will now be used to resolve the problematic situations in Fig. 4.1.

1. This is a FP for A because the predicted event is further than Δt_{max} away from the labeled event.
2. This is a TP for A and a FP for B.
3. This is a TP for the first A prediction and a FP for the second one, because a labeled event is only assigned to its best prediction.
4. This is a TP for both A and B because the event classes are evaluated separately.
5. This is a TP for the first predicted A because it is the closest to the labeled event. The second labeled event results in a FN because there is no prediction left.
6. These are two TP.

Given the definitions above, the number of P is much smaller than the number of N, therefore evaluation metrics that include TN are not very meaningful. For example a simple classifier that never detects an event would achieve almost perfect accuracy $\left(\frac{TP+TN}{N+P}\right)$. Instead, precision and recall will be used because they do not include TN. A good precision shows that the classifier does not produce many FP and a good recall indicates that the classifier is unlikely to miss events. Both performance measures are calculated for every label independently.

$$precision(D,l) = \frac{\sum\limits_{t \in D} TP(t,l)}{\sum\limits_{t \in D} (TP(t,l) + FP(t,l))} \tag{4.4}$$

$$recall(D,l) = \frac{\sum\limits_{t \in D} TP(t,l)}{\sum\limits_{t \in D} (TP(t,l) + FN(t,l))} \tag{4.5}$$

In the next sections I will investigate the following questions:

1. What influence do the parameters have on the algorithms performance?
2. Which of the three presented distance metric + normalization combinations is the best? Manhatten + n_1-normalization, euclidean + n_2-normalization or angular + n_2-normalization?
3. What is the best technique to determine δ?
4. How effective are the extrema detection and clustering as pruning techniques?
5. What is the best precision/recall I can achieve?
6. How effective is this method at classifying events online?

Unless stated otherwise, a 10-fold cross validation (10-fold-CV) was used to estimate the prediction performance of a given model on an independent dataset. That means, the whole dataset is randomly split into 10 equally sized sections. The model is trained with 9 of these sections and tested on the remaining one. This process is repeated until every section was the test set. It can be problematic to use a 10-fold-CV on an unbalanced dataset, like the one used here, because a label could be underrepresented in the training set. There are a few options to address this problem. For example, the majority class could be undersampled, i.e., examples from the majority class are only sampled until its number equals the minority class. Alternatively, the minority examples in the training set can be duplicated until it is balanced. However, I have decided to ignore the problem because it is very unlikely that no examples of a label are included in a randomly chosen 90% subset. Additionally, I will report the average precision and recall, if they are not reported for every label, because this number is more pessimistic then an average weighted by label distribution.

All experiments were conducted on a 4x4GHz CPU with 16GB main memory and the code was implemented in python.

4.2 Parameter Influence

In this section the influence of the algorithms parameters on the training time and prediction performance will be investigated. Furthermore, all experiments in this section are repeated for all three distance metrics. To recap, my algorithm has the following parameters:

- d_{max}, radius for the cluster pruning
- σ_f and σ_τ, to prevent noise amplification in force/torque data during normalization
- w_{ext}, window size for relative extrema pruning
- N_{max}, number of allowed shapelet lengths
- sl_{max}, maximum shapelet length
- dim_{max}, maximum number of dimensions allowed for a single *MTS*

All of them, expect for σ_f and σ_τ, are designed to offer a training time / performance trade-off because they remove shapelets from the candidate pool. Therefore, values of a parameter at one end of their spectrum should result in the highest training time, as well as the highest precision and recall. In the best case scenario, the training time will first rapidly decrease and then converge to a low level as the parameter is tuned towards the other end of its spectrum, while the performance decreases much more slowly because unimportant candidates were pruned first.

The goal of σ_f and σ_τ is to remove shapelets, that only contain noise. Therefore low values for this parameter likely will not result in a good performance. It is to be expected, that the performance has a clear maximum, which should also be in an area, where the training time is relatively low. However, the size of this area of good performance is probably dependent on the dataset.

All parameter combinations are tested with a 10-fold-CV. This, however, results in extremely high computation times, especially when the parameters are tuned towards performance. Therefore, the training set will only contain the force measurements during this section, if not stated otherwise. This most likely reduces the maximum performance that is possible, but should be sufficient to show the trade-off produced by the parameters.

Parameters that are not under investigation will be set to a fixed value. The experiments could still require days to be computed, that is why parameters are generally chosen in favor of a low training time.

1. $d_{max} = 0.5$ for euclidean and manhattan distance and $d_{max} = 0.15$ for angular distance. These values were chosen based on the distance between two example shapelets, see Section 3.4.4.2 for more details.
2. $\sigma_f = 0.4$ and $\sigma_\tau = 0.1$. These were determined by calculating the standard deviation of short subsequnces in episodes of free space movements, see Section 3.4.2 for more details.
3. $w_{ext} = 200$. This equals a window of ± 8 seconds. This is much longer then every event lasts

and was chosen to be that high to accelerate the learning. It could therefore be possible, that too many candidates are pruned.

4. $sl_{max} = 50$. Events in this scenario do not last for long, which is why the maximum shapelet length can be reduced to two seconds.
5. $N_{max} = 3$. This results in shapelets with the lengths $[16, 33, 50]$.
6. $dim_{max} = 3$. Since most experiments will only use either force or torque data, therefore the number of possible dimension combinations is not too high and all of them can be tested.

The threshold δ is estimated using the *KDE-δ* method because it is faster to compute than the *max-IG-δ* method and the goal in this section is to demonstrate the trade-off not to accomplish optimal prediction performances.

4.2.1 Influence of d_{max}

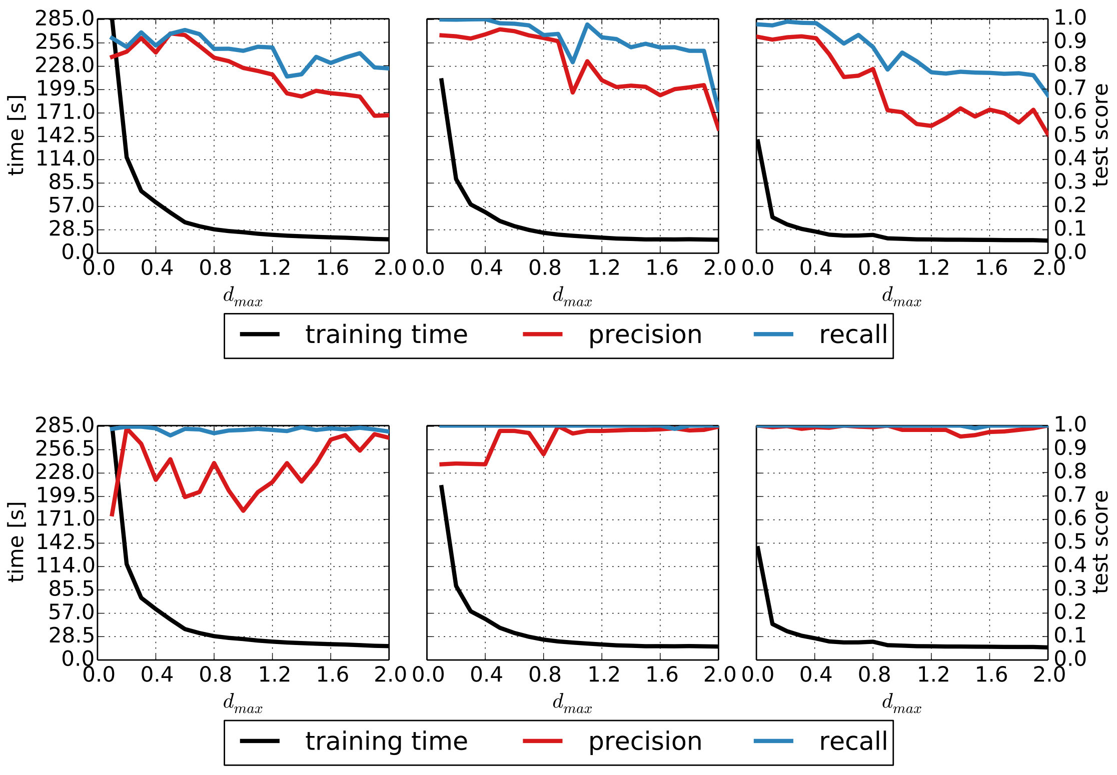


Figure 4.2 (Top) Influence of $d_{max} \in [0.1, 0.2, ..., 2.0]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively. The first value for angular distance is $d_{max} = 0.01$, not 0.
(Bottom) Same as before but only the precision and recall for *wipe_start* are shown.

Fig. 4.2 shows how different values for d_{max} influence the performance. A value of 0 would result in no candidate pruning due to clustering and is omitted as this would result in hours of training time. All values ≥ 2 do not change the results, since this is the maximum distance between shapelets.

The minimum number of shapelet candidates in this case is 21 ($|[16, 33, 50]| \cdot (2^3 - 1)$). In all three cases the number of candidates left after pruning converges quickly to this number, and so does the training time. The performance only starts to drop significantly, as the minimum amount is almost reached. Therefore this parameter clearly achieves the goal of pruning unimportant candidates first. On top of that, even the 21 last shapelets are good for some events. The performances towards the end result in a few shapelets being detected reliably and some not at all. For example *wipe_start* (Fig. 4.2 Bottom) and *wipe_end* can still be classified. This is because those events are the most frequent in the dataset.

It is also worth noting, that the performance does not drop to zero for $d_{max} \geq 1$, because $\vec{0}$s are removed before clustering. Otherwise a single cluster with $\vec{0}$ at its center would be formed because it has a distance of 1 to every shapelet and outnumbers all other similar looking shapelet candidates.

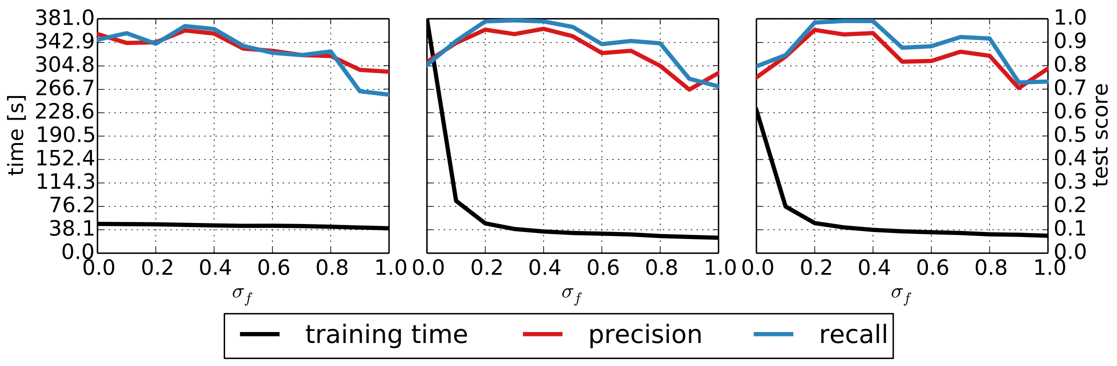
Based on these results, optimal values are $d_{max} \leq 0.8$ for manhattan and euclidean distance and $d_{max} \leq 0.4$ for angular distance. Which is a relatively big range considering the soft limit of 1. However, the manhattan distance performs the least consistent in this range.

4.2.2 Influence of σ_f and σ_τ

The influences of σ_f and σ_τ are depicted in Fig. 4.3. To test σ_τ the algorithm was only allowed to use the torque data, but the parameters were left unchanged. The first observation is that the events are more difficult to detect in the torque data alone, compared to force data. We can also see that there exists a hill of relatively good performance in the ranges $\sigma_f \in [0.2, .., 0.4]$ and $\sigma_\tau \in [0.02, .., 0.1]$. The values that were estimated from free space movements lie right at the upper end of this area. Furthermore the training time improvements start to level off at the same time as the performance is at its peak for the euclidean and angular distance. The performance gets worse as the parameter increases because more candidates are turned into $\vec{0}$. Therefore this parameter achieves the goal of preventing noise amplification for the last two distance metrics.

However, the results for the manhattan distance are both unexpected and interesting. This parameter has little influence on both the performance and the training time.

The reason for this behavior is that the number of candidates left after clustering is always approximately the same and very close to the minimum of 21 for the torque dataset. That means that for low values for σ_f and all values for σ_τ, all the random shapelets from amplified



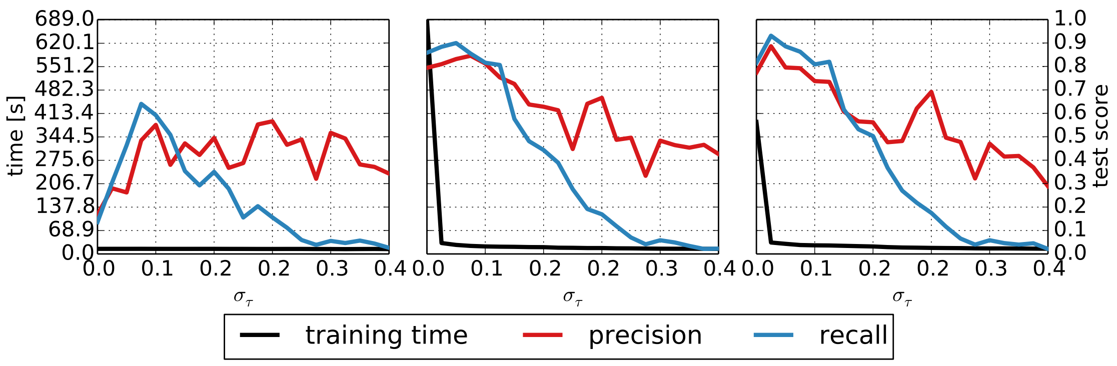


Figure 4.3 (Top) Influence of $\sigma_f \in [0, 0.1, ..., 1.0]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively.
(Bottom) Influence of $\sigma_\tau \in [0, 0.02, ...0.4]$, here the algorithm only used torque data.

noise are pruned during clustering. The shapelets that are removed first by this parameter are the ones that were originally almost straight lines. These lines usually contain very low zero mean noise that, when amplified by the normalization, results in strong high frequency waves. Such waves seem to get pruned during the clustering when the manhatten distance + normalization combination is used. Supportingly, a frequency analysis revealed that the torque data contained stronger high frequencies, than the force measurements. This could explain why almost all candidates are pruned when the torque data is used. Furthermore, with σ_τ set to 0.1, the manhattan distance only started to produces more than 100 candidates, when d_{max} is lowered below 0.01, resulting in precision and recall of above 70%, thereby proving that the candidates are just very similar.

My best explanation for this effect comes from the fact that n_1-normalized shapelets do not lie on a sphere in the Cartesian space, but some kind of polygon. During the clustering, the mean of multiple shapelets is calculated, but the resulting point does not lie in the surface of that object any more. In the next step its nearest neighbor is selected as the cluster center. I am assuming that the combination of these two steps results in some kind of bias towards a side rather then a corner of that polygon. The exact cause and possible exploitations are an interesting topic for future research.

4.2.3 Influence of w_{ext}

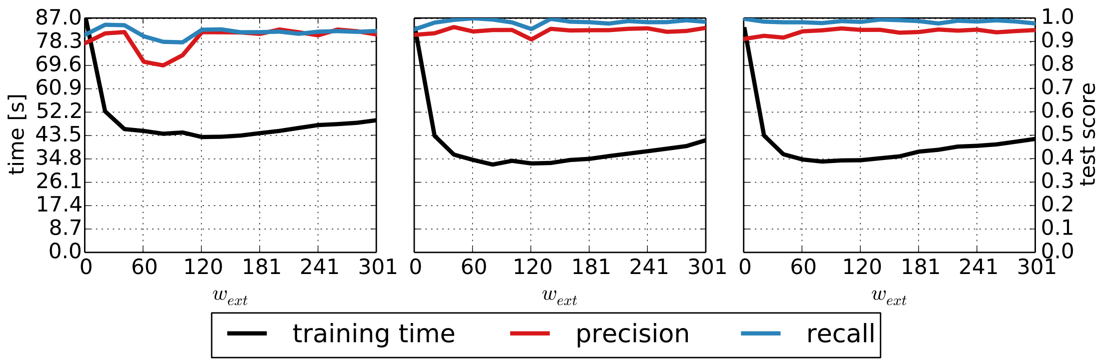


Figure 4.4 Influence of $w_{ext} \in [1, 21, 41, ..., 301]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively.

In Fig. 4.4 you can see the influence of w_{ext}. In this case, with the exception of the manhattan distance, both precision and recall stay consistently high. The training time first rapidly decreases but starts to raise again. This is because in the end the relative extrema function takes more time to prune candidates, than is saved by fewer candidates. The highest tested value ($w_{ext} = 301$) only returns the global extrema of many time series, indicating that this parameter can be removed. And in fact, repeating the experiments and using only the global extrema yields 95%/94%, 95.8%/98.5% and 95.1%/97.7% for precision/recall for manhattan, euclidean and angular distance respectively. This function is also a lot faster with 34, 24, 25 seconds average training time for the distance metrics, respectively.

However, I do not expect this result from every dataset. Only using the global extrema would add the additional assumption, that every event creates these in at least one training example. In conclusion, this parameter achieves the goal of removing undesired shapelet candidates first and, for this dataset, never removes important ones.

The classification performance drop of the manhattan distance is caused because *force_dec* could barely be classified and *wipe_start* less reliably. This time, however, it seems to be a coincidence where all the parameter are just „right" to produce this result.

4.2.4 Influence of N_{max} and sl_{max}

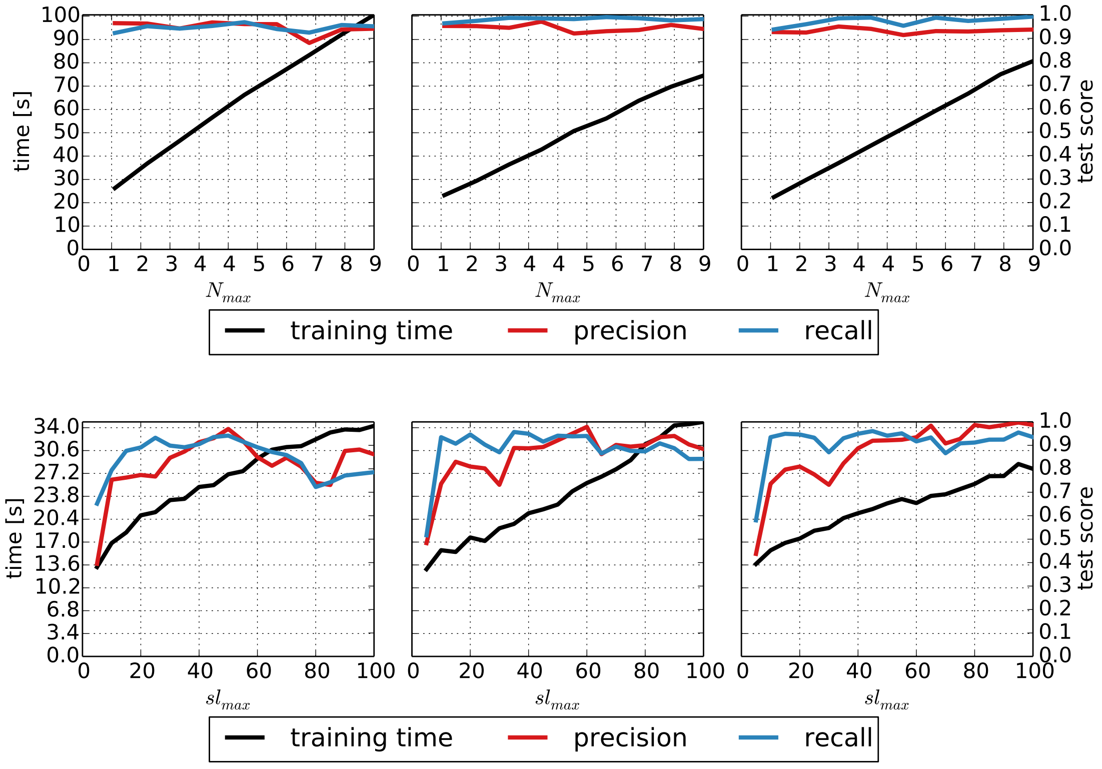


Figure 4.5 (Top) Influence of $N_{max} \in [1, 2, ..., 9]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively.
(Bottom) Influence of $sl_{max} \in [5, 10, ..., 55]$ (with $N_{max} = 1$) on average precision, recall and training time for manhattan, euclidean, angular distance, respectively.

Fig. 4.5(Top) shows the influence of N_{max}. The training time rises linearly because the clustering does not prune similar shapelets of different lengths. The performance is always good, but this can be attributed to the fact, that shapelets of length $sl_{max} = 50$ are always included and yield good precision and recall. Therefore, I have tested different values for sl_{max} while N_{max} was set to one to see how good the different length are on their own. The results are depicted in 4.5(Bottom). Short length perform poorly in terms of precision. This happens because short shapelets are not long enough to distinguish similar looking events, but they are long enough to rarely miss a real one. The only labels that performed good with very short lengths are *fixed_screw*, *wipe_start* and *wipe_end*. As sl_{max} increases, recall drops first. The reason for this is that the algorithm is no longer able to represent all training examples for the same label with a single shapelet and therefore starts to specialize on the biggest similar looking subset for each event.

4.2.5 Influence of dim_{max}

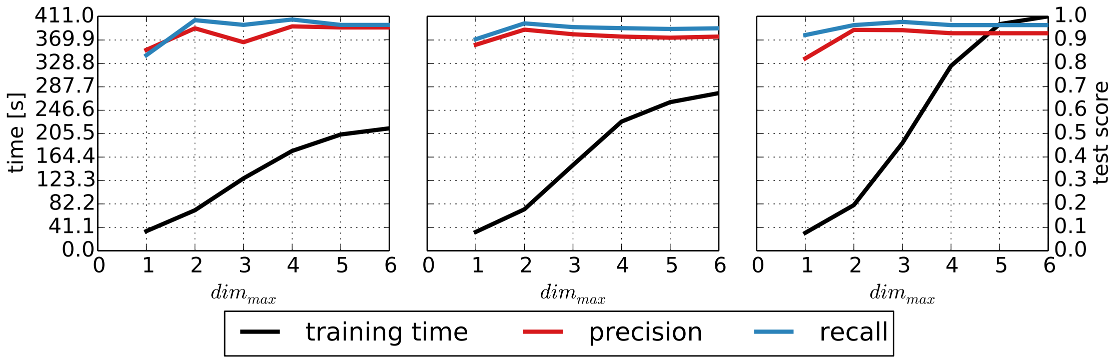


Figure 4.6 Influence of $dim_{max} \in [1, 2, ...6]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively.

For this experiment, force as well as torque data was used to increase the amount of possible values for dim_{max}. The results are shown in Fig. 4.6. After this parameter surpasses 2, precision and recall stay almost exactly the same. The reason for this is that high dimensional *MTS* do not seem to perform well. The average number of chosen dimensions converges to 2.4 for manhattan and euclidean and 2.2 for angular distance and a shapelet with 6 dimensions was never selected. This behavior probably comes from the fact that high dimensional shapelets almost certainly have some 0 dimensions and are therefore relatively closer to the sequence, that only consists of $\vec{0}$s. In particular, a torque around the z is almost never measured. The differences in training time between the distance measures is caused by the amount of shapelet candidates that remain after clustering. Especially the manhattan distance is fast, because it prunes many torque shapelets, as we have seen in section 4.2.2. In summary, this parameter also achieves its goal of pruning unimportant shapelets first. Given these findings, $dim_{max} = 3$ should be a good default value.

4.2.6 Summary

To answer the question ”What influence do the parameters have on the algorithms performance?”, it can be concluded, that all the parameters achieved their goal, of pruning unimportant shapelet candidates first. However, N_{max} and w_{ext} do have almost no influence on the classification performance. This can be contributed to the nature of the dataset. If $N_{max} = 1$, only shapelets the length of sl_{max} are tested and $sl_{max} = 50$ happened to yield a good result.

Additionally the dataset satisfies the constraint, that all events creates a global extreme in at least on training example, which is why w_{ext} has no influence on either precision or recall.

Given this insight, I will leave most parameters unchanged for the next experiments, since they were at or close to the best results. The following parameters will be changed.

- $w_{ext} = \infty$, meaning that only shapelets around global extrema will be selected, as this results in a significant training time improvement, without influencing the classification performance.
- $\sigma_{\mathcal{T}} = 0.08$, because this increases the performance, without increasing the training time by much.
- $dim_{max} = 3$, because the average dimensions used converge to approx. 2.5.

4.3 Methods to Estimate δ

In this section I will compare the three techniques presented in section 3.4.3 to estimate the separation threshold δ for *MTS*. First, I will start by comparing *max-IG-δ* and *KDE-δ* using a 10-fold-CV with all 6 dimensions with the parameters chosen according to the previous section 4.2.6. In order to not interrupt the reading flow, Table F.1 and G.1 are in the results are in the appendix.

Overall, both techniques achieve good precision and recall for all labels, except for *moveable_box*. For that label, *KDE-δ* significantly outperforms *max-IG-δ*.

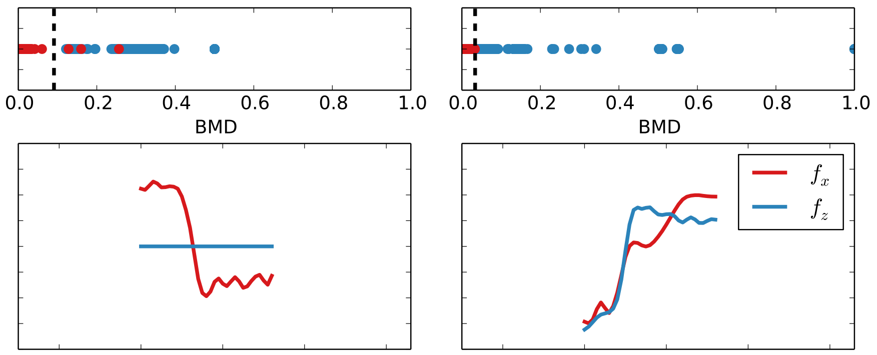


Figure 4.7 (Top) shows the 1-dimensional feature space of the shapelets below. Red dots are training examples that have the label *moveable_box* (D_1) and blue dots are the examples who do not (D_0). The dotted line indicates δ selected by *max-IG-δ*.

To understand this behavior, two shapelets and their 1d-feature space are depicted in Fig. 4.7. The left shapelet shows a possible candidate that is intuitively right based on the dataset examples shown in appendix A.1, however this one can not achieve a perfect IG. The shapelet on

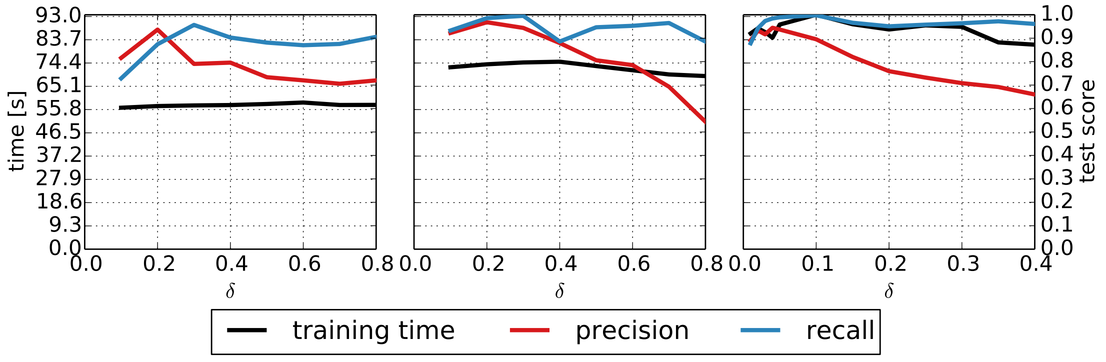


Figure 4.8 Influence of $\delta \in [0.1, 0.2, ..., 0.8]$ on average precision, recall and training time for manhattan, euclidean, angular distance, respectively. The angular distances was tested with $\delta \in [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.15, 0.2, ..., 0.4]$.

the right shows the wipe end of an experiment with a *moveable_box*. It turns out, that the box slightly pushes back on the gripper while it moves away from the table, thereby creating this shape along in f_x. This shapelet allows for a perfect separation, therefore *max-IG-δ* chooses it over the left shapelet. On the other hand, *KDE-δ* chooses δ based on the estimated distributions and is therefore less likely to create such a fine tuned split for the right shapelet. It is, however, still possible, which is why this event label often performs worst.

In fact, this situation is a violation of the assumption, that there are no two events, that always happen in the same training examples. The new event will from now on be referred to as *push_end* and it shows that events does not have to be labeled in order to violate the assumption. There are a few ways to fix this problem. First, a few push trials could be cut between those two events and added to the dataset, as it was done to split slide start and end. This puts the information gain back into the favour of the left shapelet. Alternatively, it is in this case possible to tune the parameters in such a way, that *push_end* gets pruned, because it is very similar to a normal *wipe_end*. Another option is to define a quality measure for shapelets that looks at other features, such as the class gap in the shapelets feature space, earlier in the decision process.

To summarize, it can be concluded, that *KDE-δ* has the upper hand in terms of accuracy and recall, compared to *max-IG-δ*.

distance metric	manhattan		euclidean		angular	
δ estimation technique	*max-IG-δ*	*KDE-δ*	*max-IG-δ*	*KDE-δ*	*max-IG-δ*	*KDE-δ*
average δ	0.452	0.451	0.367	0.319	0.2	0.147

Table 4.1 Average δ determined by *max-IG-δ* and *KDE-δ* during the experiments for Tables F.1 and G.1.

In terms of training time *KDE-δ* significantly outperforms *max-IG-δ*. The average training time with *max-IG-δ* for a single fold of the 10-fold-CV was 206, 270 and 366 seconds for manhattan, euclidean and angular distance respectively. In contrast, *KDE-δ* only took 106, 125 and 165 seconds on average, which is approx. a 50% speed improvement. When using smaller portions of the dataset, this difference decreases, meaning that *KDE-δ* scales better with the dataset size, than *max-IG-δ* does. The three distance metrics cause different training times mainly because the chosen parameter result in different amounts of candidates.

label	#	prec	recall	μ td [s]	σ td
wipe_start	430	1.0	1.0	-0.527	0.101
wipe_end	420	1.0	1.0	0.0397	0.0668
force_inc	30	1.0	1.0	1.42	0.147
force_dec	30	1.0	1.0	-0.769	0.267
slide_left_start	50	0.833	1.0	-0.741	0.159
slide_left_end	50	1.0	1.0	0.361	0.0841
slide_right_start	60	0.923	1.0	-0.7	0.0818
slide_right_end	60	0.9	0.9	0.797	0.117
movable_box	80	1.0	0.938	0.163	0.0763
fixed_screw	170	1.0	0.994	-0.102	0.0268
average	-	0.966	0.983	-	0.113

Table 4.2 Best performance achieved. Distance metric is euclidean, and a fixed $\delta = .2$ was used. The other parameters are $d_{max} = .5$, $N_{max} = 3$, $sl_{max} = 50$, $w_{ext} = \infty$, $dim_{max} = 3$, $\sigma_f = .4$ and $\sigma_\tau = .08$.

In conclusion *KDE-δ* outperforms *max-IG-δ* in both classification performance and training time.

For the third estimation technique, *fixed-δ*, different fixed values have been tested, the result can be seen in Fig. 4.8. These ranges were chosen, because they are also good choices for d_{max}. For comparison, Table 4.1 shows the average δs that the estimation techniques have chosen in experiments used to create Tables F.1 and G.1. The manhattan distance never showed convincingly results, however the euclidean distance performed well for $\delta \in [0.1, 0.2, 0.3]$. In fact it outperformed *max-IG-δ* and *KDE-δ* with $\delta = 0.2$, by achieving the best precision and recall during the evaluation. The detailed results are depicted in Table 4.2. The angular distance was able to achieve decent results as well, but in a smaller range. With increasing δ, the precision falls off first because the *MTS* looses its ability to reject tested subsequences, thus creating a lot of FP. It is important to point out the similarity between δ and d_{max}, both parameters are trying to isolate similar shapelets. However, the goal during the clustering is to search for a good center, for that reason, it is not very important if d_{max} is a little too big. Especially if the true cluster is isolated any way. In contrast, during the classification process the radius around the cluster center is much more important. If it is too big, we get too many false positives. Therefore good values for δ are smaller than good values for d_{max}. Since d_{max} already has to be lower than 1, this reduces the possible values for δ even further.

To summarize, *fixed-δ* clearly outperforms both *KDE-δ* and *max-IG-δ* in terms of training time. Precision and Recall, especially in combination with the euclidean distance, can outperform *KDE-δ* and *max-IG-δ* as well, if the right δ is chosen. I therefore conclude that *KDE-δ* is the best default choice, since it does not have an additional parameter. If training time is a problem, *fixed-δ* is the best choice, at least for this dataset. It is an interesting topic for future research to test this technique on other datasets.

4.4 Distance Metrics

In this section I will summarize the observations made regarding the three distance metrics in the previous sections.

Performance wise, all three distance metrics can yield similar results. When testing a wide range of parameters, the euclidean and angular distance yield the most consistent results. For most parameters, all three perform equally, except for d_{max} and σ_f/σ_τ. The angular distance has a much smaller range for d_{max} in which it can compete with the precision and recall of the other two distance metrics. The manhattan distance seems to have the property of filtering out high frequency wave shapelet candidates during clustering, thereby making σ_f/σ_τ almost irrelevant.

Regarding training time, even though the angular distance is the fastest to compute, all the differences in training time in all plots shown can be contributed to a different number of shapelet candidates, after pruning. However, the manhattan distance might have the upper hand in candidate pruning due to its apparent noisy shapelet pruning property. This is also the reason why the manhattan distance was by far the fastest option for computing the results for section 4.3, even though the parameters are almost the same as in section 4.2, where it was generally the slowest. This is because so many torque shapelets were pruned. On the contrary, the angular distance seems to produce more shapelets candidates as the dataset size increases, relative to the other metrics.

In conclusion, when looking for stability, euclidean and angular distance are preferable, but the manhattan distance might have some potential for candidate pruning.

4.5 Candidate Pruning

In this section I will showcase the effectiveness of the used pruning techniques. In particular, the effectiveness of the extrema pruning and clustering was tested. Additionally, both techniques will also be tested without the $\vec{0}$ removal, in order to show that this one is not doing all the pruning. To do so, the 8 possible pruning combinations were tested on a 10%/90% train-test split and a 90%/10% one. This shows the influence of the dataset size on the pruning effectiveness. Additionally, the 90% train set experiment did not finish when using no pruning techniques, therefore the numbers for a small training set can give an idea for the effectiveness on the 90% train set. The primary reason for not finishing was not the training time. Instead, the algorithm exhausted the 16GB working memory limit of the test machine. This is because the implementation was optimized towards a low run time at the cost of additional memory usage.

Furthermore, several steps had to be taken to make this computable. First, only the force measurements were used. Second, δ was fixed to 0.05. Third, the angular distance with $d_{max} = 0.15$ was used, because it is the fastest to compute. And finally, no 10-fold-CV was used this time. Instead the same (randomly chosen) training and test sets were used for each pruning combination. For the 10%-sized training set it was ensured that the training set had at least 3 examples of every label. Therefore, precision and recall of the following results should be taken with a grain of salt. Their only purpose is to show, by how much both values vary with the different pruning method combinations.

The results are in the appendix in Table H.2, to not interrupt the reading flow. The parameters combination is exactly the same as in the previous chapter, with the exception of w_{ext}. This value was chosen more conservatively with 50 in favor of performance because this is the point at which the number of additionally pruned candidates starts to level off.

The results show clearly, that all pruning techniques together are extremely effective. Both, the zero and extrema pruning, cut away approximately the same percentage of candidates on both dataset sizes because they do not prune across different training examples. However, the clustering reduces the number of shapelet candidates to approximately the same absolute number, for both dataset sizes. This proves that it is effective at grouping similar candidates. The runtime improvements of the clustering/no-$\vec{0}$s combination is almost exclusively gained because the clustering has to do less work. The removal of $\vec{0}$s before the clustering does result in exactly 21 fewer clusters, one for every dimension combination allowed for the *MTS*, thereby showing that all of them are combined in a single cluster. If the extrema pruning is used in addition, the number of clusters is reduced even further because the remaining shapelet candidates are always centered around the same points, thus more similar. For the 90% training dataset, there are 846 candidates left. That means that there are on average 14 shapelet candidates for each of the 63 length/dimension combination allowed for *MTS*.

Therefore, I believe that new pruning techniques that either remove whole length/dimension combinations or that prune similar candidates across those combinations will be the most promising direction for future work in this area.

It is also worth noting that the relatively bad performance when no pruning is used is mainly caused by *moveable_box*. The unlabeled *push_end* event, which was discovered during the evaluation, always appears in the same training examples and can create a split with almost perfect IG. If no pruning is used, than the algorithm is almost guaranteed to find a shapelet for *push_end*, for which $\delta = 0.05$ creates a perfect separation, because there are so many to choose from.

4.6 Online Event Detection and Classification

To answer the last question „How effective is this method at classifying events online?", we have to first define what ‚effective" means in this context. Most important is that the classification is both precise and sensitive, meaning that optimally no events are missed or falsely detected. Additionally, the delay between the real event occurrence and detection should be low, otherwise it is to late for a high-level planning architecture to respond.

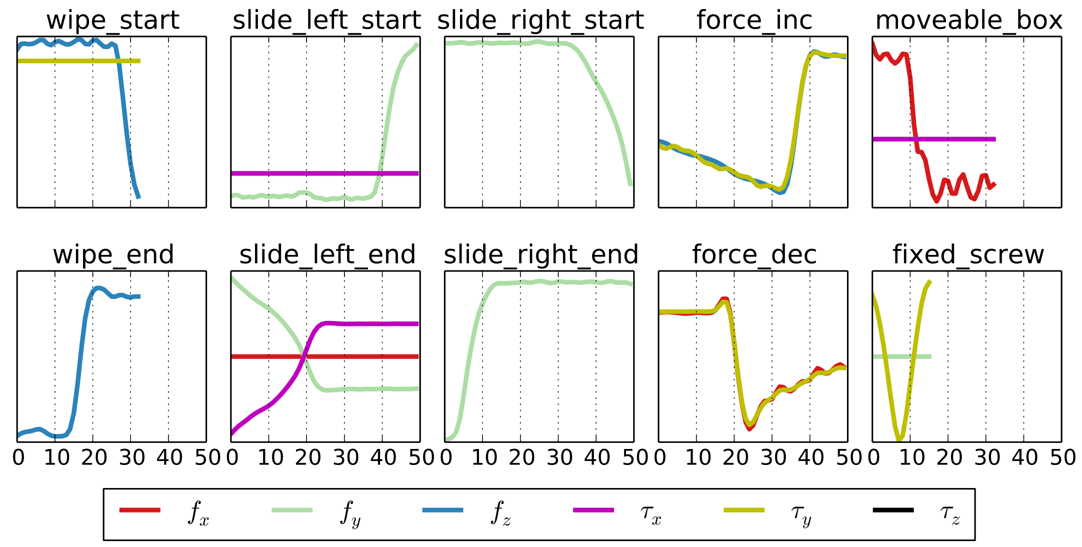


Figure 4.9 The learned shapelets from the first fold in Table 4.2. Please note that the *MTS* for *moveable_box* is three dimensional, a horizontal f_z line is overlapped by the τ_x line.

As shown in the previous sections, an average precision and recall of over 90% is achievable for many parameter setups. The parameters are also easy to tune because they offer a performance-training time trade off.

The best parameter combination found during the previous sections is depicted in Table 4.2, achieving a precision of 96.6% and a recall of 98.3%. Additionally the standard deviation of the time differences is relatively low, with the exception of *force_dec*, which will be addressed later on. This indicates, that the *MTS* learned a shape that happens either before or after the point that I have assigned the label to.

Fig. 4.9 depicts the shapelets, that were learned during the first fold of the 10-fold-CV for Table 4.2. If we compare these with the dataset examples in appendix A.1, then we can conclude, that the all *MTS* did learn shapes that correlate with the events, even if the absolute time differences are bigger. For example the *wipe_start MTS* ends with a strong force increment (in absolute terms) along the z-axis. The horizontal line for the torque around the y-axis probably captures the fact, that the force along the z-axis changes first. That force increment, however, is very close to the point, that I have assigned the label to. Therefore the distance between the center of the shapelet has to be approx. 0.5 seconds before the labeled point, if it matches perfectly. The same holds true for the slide related shapelets. However, the *wipe_end*-, *moveable_box*- and *fixed_screw*-shapelets are centered right around the point that was also labeled by me, thus resulting in very low time differences. The last two labels, *force_inc* and *force_dec*, seem to require the information that the force was high in order to tell that it has decreased, or vice versa. Furthermore, a particular point in time can not be determined for such long lived events. I have decided to label the mid point of these events. The training examples for *force_dec* have a high variety in length and the *force_inc* recordings happen to be longer on average. This explains why the *force_dec* has a high variance in time differences and the *force_inc* has a high absolute difference. Therefore, I conclude that the classifiers are reliable in terms of precision and recall.

It should also be noted that I have observed that there exists a small set of feasible shapelets for each event. For example, *wipe_start* is often represented by a *MTS* that only consists of f_z and is centered around the rapid force increment, such as *wipe_end* is in this example. It depends on the parameter setup and train/test split, which shapelet happens to be selected.

To test the time it takes for an event to be detected by a *MTS*, I have implemented a tool that uses them to classifies events in a data stream. ROS, which was used to communicate with the robot, offers the functionality to record sent messages and replay them as if they were streamed from the real sensor. This was utilized to replay recordings, that were not in the training set for the *MTS* depicted above. In one such recording, a box was fixed on the table. The movement started over the edge of the box such that the sponge touched it before the table while it was being moved down. Then a long wiping action was commanded such that the *slide_right_end* event happened way before *wipe_end*. This recording and the events detected by the shapelets from Fig. 4.9 are depicted in Fig. 4.10. [1] To showcase the classification time, the tool buffered the last 10 seconds of sensor measurements and the *MTS* were used to detect events in the whole

[1] A few more examples can be found in this video *https://youtu.be/l4Ttcmyy34o*

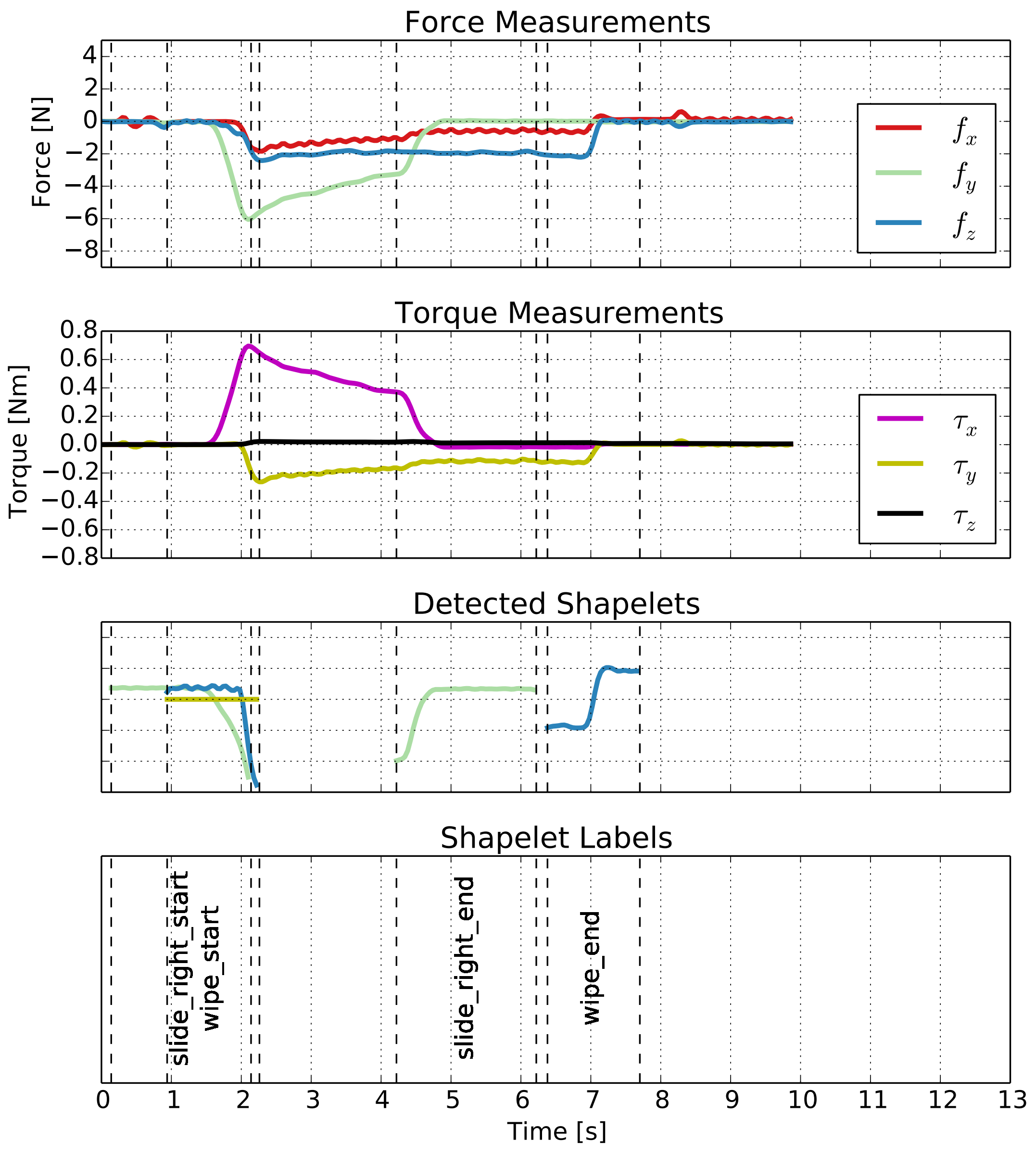


Figure 4.10 Detecting events online using the shapelets depicted in Fig. 4.9.

window. This took on average 0.0023 seconds per label or 0.023 together, if the computations are not computed simultaneously, which is more than fast enough to handle data streamed at 25hz. Therefore the time it takes to classify the events does not put a limit on the systems responsiveness. There are also ways to speed up the process, for example by buffering a window the size of the longest shapelet and only keeping track of the minimum distances from the previous windows.

All events from this recording could be detected. Both *wipe_start* and *slide_right_start* report the detected event shortly after the impact. However, *slide_right_end* is detected later. It has a length of 50 and therefore reports the event about 2 seconds too late. Therefore the responsiveness of the system is dependent on the learned shapelet.

A possible countermeasure would be a shapelet quality measure that favors shapelets that have „the event at there end", if this is possible. Limiting the maximum length of shapelets will result in more false positives, as shown during the evaluation of sl_{max} in section 4.2.4. However, if this system is integrated into a high-level planning architecture, knowledge from that plan can be used to discard false positives. For example, a detected *wipe_end* is likely a false positive, if no *wipe_start* was detected beforehand.

To summarize, *MTS* are able to reliably detect and classify events online and the classification time does not put a limit on the systems responsiveness. However, the time differences between the real events and classification output depends on the chosen shapelet. To mitigate this problem, a low maximum shapelets length can be selected, if a high-level reasoning system can discard the resulting false positives.

4.7 Summary and Discussion

To summarize the above, I will answer the questions asked in the first section of this chapter.

„What influence do the parameters have on the algorithms performance?"
The parameters used to prevent noise amplification during the normalization (σ_f and σ_τ) behave as expected. For low values the training times and performances are low due to random shapelets. Values around the threshold estimated from free-space movement recordings show a fast training time and a peak in precision and recall. As these values increase, the performance drops because important *MTS* candidates are pruned. All the other parameters achieve the goal of providing a good performance/training time trade-off, by removing unimportant candidates first.

„Which of the three presented distance metric + normalization combinations is the best? Manhattan + n_1-normalization, euclidean + n_2-normalization or angular + n_2-normalization?"
All three combinations can achieve a similarly good precision and recall, but the manhattan distance is the least consistent. However, it seems to have the property of pruning high frequency shaped candidates. Due to this property, it outperforms the other distance metrics in terms of training time by pruning torque candidates. The angular distance is the fastest to compute, but this advantage is mitigated by the observation that it seems to produce the most candidates as the size of the dataset increased. The euclidean distance lies in between and managed to produce the single best precision and recall during the experiments.

„What is the best technique to determine δ?"
The *KDE-δ* technique outperforms *max-IG-δ* in both, training time and precision/recall. It is less susceptible to single outliers and is able to classify *moveable_box* most of the time, even though there is a second unlabeled event *push_end* in the dataset, that always appears in the same time series. The third method, *fixed-δ*, can achieve a significant training time improvement, by adding an additional parameter. However, due to the fact, that shapelets have a maximum distance the parameter range can be greatly reduced. In combination with the euclidean distance, an average precision and recall of over 90% is achievable for a relatively huge range. Additionally, high values only results in a drop in precision. It is therefore a good option, when the training time with *KDE-δ* would be too high. However, more testing is needed to confirm this behavior on other dataset.

„How effective are the extrema detection and clustering as pruning techniques?"
Both pruning techniques are very effective. In combination they can prune up to 99.969% of the shapelet candidates, leaving on average 14 candidates per length/dimension subset combination.

„What is the best precision/recall I can achieve?"
The highest precision and recall achieved are 96.6% and 98.3% in a reasonable amount of training time, with 69 seconds. Considering the nature of the parameters, it is also important to emphasize that a performance of over 90% can be achieved with a low training time of under a minute using only force readings.

„How effective is this method at classifying events online?"
The online detection is both accurate and fast enough. However, a limit is put on the responsiveness by the fact that the whole shapelet has to be matched. A promising countermeasure is to reduce the maximum length for shapelets and reject the resulting false positives using a reasoning system.

In contract to other event detection and classification algorithms, the one presented in this thesis makes the least assumptions on the training set by learning from weakly labeled training data. The only assumption left is that no two events always occur in the same training examples. Even though this was not satisfied for the *moveable_box* event in the dataset used here due to a unlabeled one detected during evaluation, the algorithm chose the right shapelet more often than not, when *max-IG-δ* was not used. It is a topic for future research to improve this ability.

The most similar approach to this learning algorithm was presented by Hu et al. [20], who have also pointed out the flaw of requiring perfectly extracted patterns of other algorithms. However, they still require that a training example only belongs to a single class. Their approach is also not based on shapelets, instead, scale and offset invariance is achieved by selecting multiple subsequences that represent a class. Furthermore, no training times were reported.

The classifiers in their trained form produced by the algorithm proposed here is the most similar to the one proposed by Ko et al. [24]. They have also represented a class with a single subsequence and combined it with a rejection threshold which was the same for all classes, similar to δ in shapelets. They are able to deal with lag between dimensions by utilizing DTW, but this slows down the classification process significantly. The subsequences used for classification were selected from a set of perfectly extracted examples for each class.

Both techniques also suffer from the problem, that the whole subsequence has to be matched, until the event can be classified. Therefore the responsiveness of their systems is also limited in the same way that the one proposed in this thesis is.

Chapter 5

Conclusion and Perspective

The objective of this thesis was to develop an algorithm that can learn task specific force profiles of contact events in force/torque sensor readings using only weakly labeled training examples, in order to detect and classify such events online. As a model problem, robotic wiping tasks were investigated because they will make up most of a future service robots todo-list and because camera vision is usually impaired by the robots hand, arm or held tool. It was shown that the shapelet discovery algorithm can be exploited to learn multidimensional time series shapelets from weakly labeled training data that capture the distinct shape profiles of contact events. To evaluate the algorithm, a dataset of 520 real world robotic wiping episodes was recorded, in which several contact events are labeled. However, the specific points in time of event occurrence was not available to the algorithm. Nevertheless, the learned *MTS* are able to achieve a very high precision and recall of up to 96.6% and 98.8%, respectively, in a 10-fold-CV. The time differences between labeled events and those predicted by the *MTS* show a low variance, which indicates that a shape has been selected that correlates with the event. Furthermore, it was shown that *MTS* can classify events in streamed data reliably and fast online. The only factor limiting the responsiveness of the detection is the fact, that the whole shapelet has to be matched. This can result in a delay depending on the length of the shapelet and depending on which part of the event it is representing. Limiting the maximum shapelet length can dampen this effect, but results in more false positives. However, this problem is not unique to this algorithm but a consequence of detecting events by matching smaller sequences/shapelets. The first major direction for future work is the integration of this contact event detection system into a high-level planning architecture. This combination can increase the responsiveness by limiting the shapelet length and rejecting false positives using reasoning based on the current plan.

Several pruning techniques were proposed to speed up the learning process because the general shapelets discovery algorithm scales very poorly with the training set size. Most notably, centering *MTS* candidates around extrema and pruning similar ones using time series clustering could reduce the number of candidates by up to 99.969%. All pruning techniques introduce a new parameter that offers a performance/training time trade-off, where most of the training time reduction is achieved without significantly influencing the performance. Precision and recall scores of over 90% can be achieved in a training time of a few minutes. The second major direction for future research is to speed up the learning process even further, in particular, pruning similar candidates with different length/dimensions seems to be most promising.

Furthermore, it was proven that the normalization applied to shapelets in order to achieve scale and offset invariance limits the maximum distance they can have. This property reduces the range of feasible separation thresholds for the *MTS* so much, that decent results can be achieved in a significantly reduced training time by setting that threshold to the same fixed value for all shapelets. In fact, such a fixed threshold resulted in the highest precision and recall listed above. Another consequence is the theoretical justification for normalization + distance measure combination alternatives to the z-normalization + length normalized euclidean distance used in the literature. Hence, a manhattan based normalization and distance combination, as well as the angular distance are compared to the euclidean default. Especially the manhattan combination shows interesting behavior when facing shapelets in the form of high frequency waves. Hence, the third major direction for future work is the further investigation of the mathematical properties of *MTS* and their distance measures. In particular, replacing low variance shapelets with the $\vec{0}$ vector is likely not the optimal solution to prevent noise amplification because it creates unwanted side effects, e.g., the special case for the angular distance. Furthermore, it would be interesting to see, whether the parameters used for the wiping dataset are also feasible for other datasets, as I hypothesis.

Appendix A

Appendix

A.1 List of Figures

A.2 List of Tables

A.3 Bibliography

[1] A Bagnall, A Bostrom, J Large, and J Lines. “The great time series classification bake off: an experimental evaluation of recently proposed algorithms”. In: *Extended Version. CoRR, abs/1602.01711* (2016).

[2] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. “Time-series classification with COTE: the collective of transformation-based ensembles”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (2015), pp. 2522–2535.

[3] Maya Cakmak and Leila Takayama. “Towards a comprehensive chore list for domestic robots”. In: *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*. IEEE Press. 2013, pp. 93–94.

[4] Graeme S Chambers, Svetha Venkatesh, Geoff AW West, and Hung Hai Bui. “Segmentation of intentional human gestures for sports video annotation”. In: *Multimedia Modelling Conference, 2004. Proceedings. 10th International*. IEEE. 2004, pp. 124–129.

[5] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, GC Maling, David E Nelson, Charles M Rader, and Peter D Welch. “What is the fast Fourier transform?” In: *Proceedings of the IEEE* 55.10 (1967), pp. 1664–1674.

[6] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. “Rule Discovery from Time Series.” In: *KDD*. Vol. 98. 1. 1998, pp. 16–22.

[7] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. “Querying and mining of time series data: experimental comparison of representations and distance measures”. In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1542–1552.

[8] Pedro Domingos. “A few useful things to know about machine learning”. In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[9] Brian Eberman and J Kenneth Salisbury. “Application of change detection to dynamic contact sensing”. In: *The International Journal of Robotics Research* 13.5 (1994), pp. 369–394.

[10] J Randall Flanagan, Miles C Bowman, and Roland S Johansson. “Control strategies in object manipulation tasks”. In: *Current opinion in neurobiology* 16.6 (2006), pp. 650–659.

[11] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. “Learning time-series shapelets”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 392–401.

[12] Josif Grabocka, Martin Wistuba, and Lars Schmidt-Thieme. “Fast classification of univariate and multivariate time series through shapelet discovery”. In: *Knowledge and Information Systems* 49.2 (2016), pp. 429–454.

[13] Valery Guralnik and Jaideep Srivastava. “Event detection from time series data”. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 1999, pp. 33–42.

[14] Ehtesham Hassan, Gautam Shroff, and Puneet Agarwal. "Multi-sensor event detection using shape histograms". In: *Proceedings of the Second ACM IKDD Conference on Data Sciences*. ACM. 2015, pp. 20–29.

[15] Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. "Learning the state transition model to efficiently clean surfaces with mobile manipulation robots". In: *Proc. of the Workshop on Manipulation under Uncertainty at the IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2011.

[16] Bernhard Hommel. "Action control according to TEC (theory of event coding)". In: *Psychological Research PRPF* 73.4 (2009), pp. 512–526.

[17] GE Hovland and Brenan J McCarragher. "Frequency-domain force measurements for discrete event contact recognition". In: *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 2. IEEE. 1996, pp. 1166–1171.

[18] Geir E Hovland and Brenan J McCarragher. "Combining force and position measurements for the monitoring of robotic assembly". In: *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*. Vol. 2. IEEE. 1997, pp. 654–660.

[19] Geir E Hovland and Brenan J McCarragher. "Hidden Markov models as a process monitor in robotic assembly". In: *The International Journal of Robotics Research* 17.2 (1998), pp. 153–168.

[20] Bing Hu, Yanping Chen, and Eamonn Keogh. "Time series classification under more realistic assumptions". In: *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM. 2013, pp. 578–586.

[21] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. "Gesture spotting with body-worn inertial sensors to detect user activities". In: *Pattern Recognition* 41.6 (2008), pp. 2010–2024.

[22] Eamonn Keogh, Jessica Lin, and Wagner Truppel. "Clustering of time series subsequences is meaningless: Implications for previous and future research". In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE. 2003, pp. 115–122.

[23] Günther Knoblich and Rüdiger Flach. "Predicting the effects of actions: Interactions of perception and action". In: *Psychological science* 12.6 (2001), pp. 467–472.

[24] Ming Hsiao Ko, Geoff West, Svetha Venkatesh, and Mohan Kumar. "Using dynamic time warping for online temporal fusion in multisensor systems". In: *Information Fusion* 9.3 (2008), pp. 370–388.

[25] Daniel Leidner and Michael Beetz. "Inferring the effects of wiping motions based on haptic perception". In: *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE. 2016, pp. 461–468.

[26] Daniel Leidner, Wissam Bejjani, Alin Albu-Schäffer, and Michael Beetz. "Robotic agents representing, reasoning, and executing wiping tasks for daily household chores". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*.

International Foundation for Autonomous Agents and Multiagent Systems. 2016, pp. 1006–1014.

[27] Daniel Leidner, Christoph Borst, Alexander Dietrich, Michael Beetz, and Alin Albu-Schäffer. "Classifying compliant manipulation tasks for automated planning in robotics". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE. 2015, pp. 1769–1776.

[28] Daniel Leidner, Alexander Dietrich, Michael Beetz, and Alin Albu-Schäffer. "Knowledge-enabled parameterization of whole-body control strategies for compliant service robots". In: *Autonomous Robots* 40.3 (2016), pp. 519–536.

[29] T Warren Liao. "Clustering of time series data—a survey". In: *Pattern recognition* 38.11 (2005), pp. 1857–1874.

[30] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. "A symbolic representation of time series, with implications for streaming algorithms". In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery.* ACM. 2003, pp. 2–11.

[31] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. "A shapelet transform for time series classification". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2012, pp. 289–297.

[32] Li Liu, Yuxin Peng, Shu Wang, Ming Liu, and Zigang Huang. "Complex activity recognition using time series pattern dictionary learned from ubiquitous sensors". In: *Information Sciences* 340 (2016), pp. 41–57.

[33] Shengfa Miao, Ugo Vespier, Ricardo Cachucho, Marvin Meeng, and Arno Knobbe. "Predefined pattern detection in large time series". In: *Information Sciences* 329 (2016), pp. 950–964.

[34] Abdullah Mueen, Eamonn Keogh, and Neal Young. "Logical-shapelets: an expressive primitive for time series classification". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2011, pp. 1154–1162.

[35] Valerio Ortenzi, Maxime Adjigble, Jeffrey A Kuo, Rustam Stolkin, and Michael Mistry. "An experimental study of robot control during environmental contacts based on projected operational space dynamics". In: *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on.* IEEE. 2014, pp. 407–412.

[36] Om P Patri, Anand V Panangadan, Charalampos Chelmis, and Viktor K Prasanna. "Extracting discriminative features for event-based electricity disaggregation". In: *Technologies for Sustainability (SusTech), 2014 IEEE Conference on.* IEEE. 2014, pp. 232–238.

[37] Dan Pelleg, Andrew W Moore, et al. "X-means: Extending K-means with Efficient Estimation of the Number of Clusters." In: *ICML.* Vol. 1. 2000, pp. 727–734.

[38] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software.* Vol. 3. 3.2. 2009, p. 5.

[39] Lawrence R Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[40] Thanawin Rakthanmanon and Eamonn Keogh. “Fast shapelets: A scalable algorithm for discovering time series shapelets”. In: *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM. 2013, pp. 668–676.

[41] Thanawin Rakthanmanon, Eamonn J Keogh, Stefano Lonardi, and Scott Evans. “Time series epenthesis: Clustering time series streams requires ignoring some data”. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE. 2011, pp. 547–556.

[42] Stan Salvador and Philip Chan. “Toward accurate dynamic time warping in linear time and space”. In: *Intelligent Data Analysis* 11.5 (2007), pp. 561–580.

[43] Christopher Schindlbeck and Sami Haddadin. “Unified passivity-based cartesian force/impedance control for rigid and flexible joint robots via task-energy tanks”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 440–447.

[44] Pavel Senin and Sergey Malinchik. “Sax-vsm: Interpretable time series classification using sax and vector space model”. In: *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE. 2013, pp. 1175–1180.

[45] Karsten Sternickel. “Automatic pattern recognition in ECG time series”. In: *Computer methods and programs in biomedicine* 68.2 (2002), pp. 109–115.

[46] Ioan A Sucan and Sachin Chitta. “Moveit!” In: *Online at http://moveit. ros. org* (2013).

[47] Ken Ueno, Xiaopeng Xi, Eamonn Keogh, and Dah-Jye Lee. “Anytime classification using the nearest neighbor algorithm with applications to stream mining”. In: *Data Mining, 2006. ICDM’06. Sixth International Conference on*. IEEE. 2006, pp. 623–632.

[48] Jan Winkler and Michael Beetz. “Robot action plans that form and maintain expectations”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 5174–5180.

[49] Hui-Hua Wu and Shanhe Wu. “Various proofs of the Cauchy-Schwarz inequality”. In: *Octogon Mathematical Magazine* 17.1 (2009), pp. 221–229.

[50] Zhengzheng Xing, Jian Pei, Philip S Yu, and Ke Wang. “Extracting interpretable features for early classification on time series”. In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM. 2011, pp. 247–258.

[51] Lexiang Ye and Eamonn Keogh. “Time series shapelets: a new primitive for data mining”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 947–956.

[52] Lipeng Zhu, Chao Lu, and Yuanzhang Sun. “Time series shapelet classification based online short-term voltage stability assessment”. In: *IEEE Transactions on Power Systems* 31.2 (2016), pp. 1430–1439.

A.4 List of Abbreviations

BMD best match distance, S. 16, 17, 21–24, 26, 28–31, 33, 36, 37, 39, 65, 66

MTS multidimensional time series shapelets, S. 4, 16, 21–23, 28, 29, 31, 32, 37, 38, 41, 44, 50, 51, 53, 55–57, 59, 63–65, 67

1-NN 1-nearest neighbour, S. 8, 9, 28, 38

10-fold-CV 10-fold cross validation, S. 43, 44, 51, 53, 55, 57, 63, 67, 74, 75

ANN artificial neural networks, S. 9

COTE collective of transformation based ensembles, S. 9

DTW dynamic time warping, S. 8, 9, 11, 21, 28, 38, 39, 61

FFT fast fourier transformation, S. 7

FN false negative, S. 42, 43

FP false positive, S. 10, 11, 32, 42, 43, 53, 65, 66

HMM hidden markov models, S. 9

IG information gain, S. 17, 22, 29–32, 51, 56, 66

KDE kernel density estimation, S. 30, 31

N negatives, S. 11, 42

P positives, S. 11, 42

SAX Symbolic Aggregate approXimation, S. 9, 28

STS subsequence time series, S. 11

SVM support vector machine, S. 9

TN true negatives, S. 11, 42, 43

TP true positive, S. 10, 11, 41–43, 65, 66

A.5 Examples from the Dataset

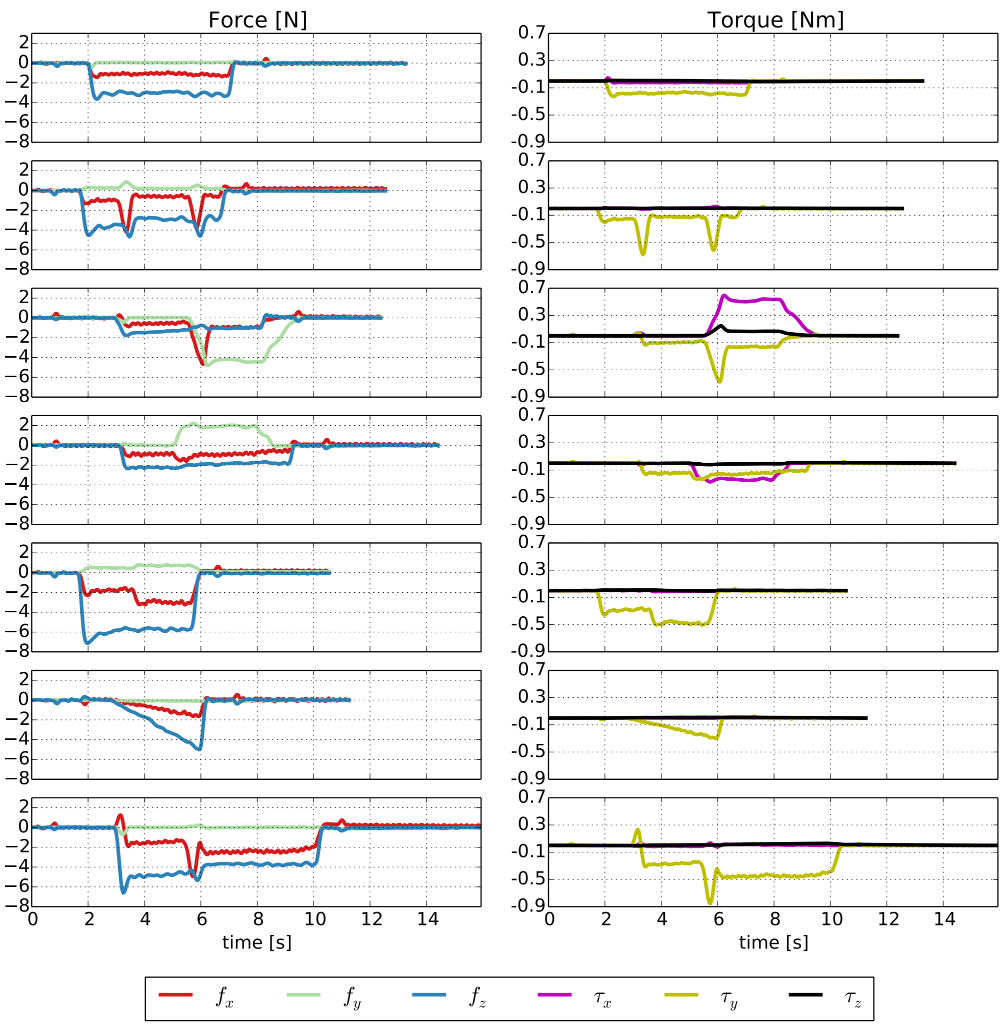


Figure A.1 Example time series from the dataset. Each row shows the force and torque measurements from the same recording.

All of these experiments are labeled with *wipe_start/end*, except for row 7. In Row 1 the table is empty. In Row 2 the table had two screws in it, labels: *fixed_screw*. In Row 3 a fixed box was on the right side of the gripper, labels: *slide_right_start/end*. In Row 4 the fixed box was on the grippers left side, labels: *slide_left_start/end*. In Row 5 an impact with a moveable object is shown labels: *moveable_box*. In Row 7 the pressure towards the table is increasing, labels: *force_inc*, *wipe_end*. In Row 8 a moveable object is directly behind a fixed screw, labels: *fixed_screw*, *moveable_box*.

A.6 Results for the *max-IG*-δ-based Technique

		manhattan				euclidean				angular			
label	#	prec	recall	μ td [s]	σ td	prec	recall	μt [s]	σ td	prec	recall	μ td [s]	σ td
wipe_start	430	0.962	0.995	0.509	0.17	0.984	1.0	0.274	0.123	1.0	1.0	0.805	0.104
wipe_end	420	0.809	1.0	-0.282	0.173	0.966	1.0	-0.423	0.0678	0.886	0.976	-0.78	0.0776
force_inc	30	1.0	1.0	1.08	0.147	1.0	1.0	1.15	0.229	1.0	1.0	1.11	0.168
force_dec	30	0.968	1.0	-0.633	0.248	1.0	1.0	-0.763	0.261	1.0	1.0	-0.785	0.242
slide_left_start	50	0.98	1.0	-0.687	0.175	1.0	1.0	-0.408	0.13	1.0	1.0	-0.424	0.137
slide_left_end	50	1.0	1.0	0.408	0.198	1.0	1.0	0.361	0.0841	1.0	1.0	0.3	0.0764
slide_right_start	60	0.983	0.983	-0.447	0.0731	1.0	0.983	0.214	0.271	1.0	0.983	0.21	0.282
slide_right_end	60	1.0	0.95	-0.366	0.129	0.908	0.983	0.831	0.155	0.908	0.983	0.831	0.155
movable_box	80	0.461	0.438	0.247	0.458	0.537	0.537	0.389	0.601	0.291	0.287	0.522	0.764
fixed_screw	170	1.0	1.0	-0.0849	0.0242	1.0	0.982	-0.0515	0.0936	1.0	1.0	-0.064	0.0227
average	-	0.916	0.937	-	0.18	0.939	0.949	-	0.202	0.908	0.923	-	0.203

Table F.1 Results from 10-fold-CV using force and torque data and *max-IG-δ* to determine δ. Parameters: $d_{max} = .5$ for manhattan and euclidean and $d_{max} = .15$ for angular distance, $N_{max} = 3$, $sl_{max} = 50$, $w_{ext} = \infty$, $dim_{max} = 3$, $\sigma_f = .4$ and $\sigma_\tau = .08$.

A.7 Results for the *KDE*-δ-based Technique

		manhattan				euclidean				angular			
label	#	prec	recall	μ td [s]	σ td	prec	recall	μ td [s]	σ td	prec	recall	μ td [s]	σ d
wipe_start	430	0.968	1.0	0.56	0.162	0.971	1.0	0.535	0.148	0.995	1.0	0.736	0.143
wipe_end	420	0.911	1.0	-0.212	0.163	0.963	1.0	-0.13	0.122	0.919	0.976	-0.78	0.0776
force_inc	30	1.0	0.867	1.05	0.13	1.0	1.0	1.05	0.21	1.0	0.967	1.12	0.16
force_dec	30	1.0	0.967	-0.665	0.266	1.0	1.0	-0.723	0.235	1.0	1.0	-0.691	0.227
slide_left_start	50	0.98	1.0	-0.687	0.175	0.893	1.0	-0.741	0.159	0.926	1.0	-0.746	0.156
slide_left_end	50	0.909	1.0	0.376	0.228	0.909	1.0	0.313	0.0795	0.847	1.0	0.553	0.215
slide_right_start	60	1.0	1.0	-0.467	0.0825	1.0	1.0	-0.263	0.224	1.0	1.0	-0.23	0.228
slide_right_end	60	1.0	0.95	-0.366	0.129	0.844	0.9	0.794	0.37	0.812	0.933	0.801	0.162
movable_box	80	0.883	0.85	0.112	0.056	0.605	0.613	0.275	0.579	0.91	0.887	0.0727	0.0736
fixed_screw	170	1.0	0.959	-0.177	0.126	1.0	1.0	-0.1	0.0258	1.0	1.0	-0.064	0.0227
average	-	0.965	0.959	-	0.152	0.918	0.951	-	0.215	0.941	0.976	-	0.147

Table G.1 Results from 10-fold-CV using force and torque data and *KDE*-δ to determine δ. Parameters: $d_{max} = .5$ for manhattan and euclidean and $d_{max} = .15$ for angular distance, $N_{max} = 3$, $sl_{max} = 3$, $w_{ext} = 50$, $w_{ext} = \infty$, $dim_{max} = 3$, $\sigma_f = .4$ and $\sigma_\tau = .08$.

A.8 Comparison of Pruning Techniques

prun. tech.	$\lvert S\rvert$ left	time [s]	avg prec/ avg recall
[ ] no-$\vec{0}$s [ ] extrema [ ] cluster	100%/ 2,697,639	n/a	n/a
[x] no-$\vec{0}$s [ ] extrema [ ] cluster	22.1%/ 596,774	n/a	n/a
[ ] no-$\vec{0}$s [x] extrema [ ] cluster	6.11%/ 164,950	2334	0.93 / 0.988
[x] no-$\vec{0}$s [x] extrema [ ] cluster	2.45%/ 66,141	722	0.93 / 0.988
[ ] no-$\vec{0}$s [ ] extrema [x] cluster	0.065%/ 1,756	81.4	0.91 / 0.988
[x] no-$\vec{0}$s [ ] extrema [x] cluster	0.064%/ 1,735	61.3	0.91 / 0.988
[ ] no-$\vec{0}$s [x] extrema [x] cluster	0.032%/ 867	18.9	0.884 / 0.954
[x] no-$\vec{0}$s [x] extrema [x] cluster	0.031%/ 846	17.7	0.884/ 0.954

(a) 90% train, 10% test split.

prun. tech.	$\lvert S\rvert$ left	time [s]	avg prec/ avg recall
[ ] no-$\vec{0}$s [ ] extrema [ ] cluster	100%/ 298,914	506.8	0.856 / 0.887
[x] no-$\vec{0}$s [ ] extrema [ ] cluster	22.6%/ 67,714	93.5	0.856 / 0.887
[ ] no-$\vec{0}$s [x] extrema [ ] cluster	6.1%/ 18,357	31.4	0.924 / 0.99
[x] no-$\vec{0}$s [x] extrema [ ] cluster	2.5%/ 7,512	10.49	0.924 / 0.99
[ ] no-$\vec{0}$s [ ] extrema [x] cluster	0.51%/ 1,541	7.95	0.9 / 0.954
[x] no-$\vec{0}$s [ ] extrema [x] cluster	0.50%/ 1,520	6.97	0.9 / 0.954
[ ] no-$\vec{0}$s [x] extrema [x] cluster	0.22%/ 683	2.15	0.85 / 0.902
[x] no-$\vec{0}$s [x] extrema [x] cluster	0.22%/ 662	2.07	0.85 / 0.902

(b) 10% train, 90% test split.

Table H.2 Evaluation of the pruning techniques. All rows of the same subtables have used the same training and test set. A fixed $\delta = 0.05$ was used to speed up the training time. The parameters were set to $d_{max} = .15$, $\sigma_{min} = .4$, $N_{max} = 3$, $w_{ext} = 50$, $sl_{max} = 50$, $dim_{max} = 3$, distance metric = angular.

A.9 Proofs

This appendix contains the proofs referenced in this thesis. The Cauchy-Schwarzsche inequality will be necessary, a proof can be found in [49]:

$$\left(\sum_{i=1}^{n} a_i \cdot b_i\right)^2 \leq \left(\sum_{i=1}^{n} a_i^2\right) \cdot \left(\sum_{i=1}^{n} b_i^2\right) \tag{A.1}$$

Theorem 1. *If $x, y \in \mathbb{R}^n$, $c, d \in \mathbb{R}$ and $p \in \mathbb{R}^+$ such that*

$$c||x||_p = c||y||_p = d \tag{A.2}$$

then $c(||x-y||_p) \leq 2d$.

Proof of Theorem 1.

$$\begin{aligned} & c(||x-y||_p) \\ \overset{\textit{triangle inequality}}{\leq} & c(||x||_p + ||y||_p) \\ = & c(||x||_p) + c(||y||_p) \\ \overset{(A.2)}{=} & d + d \\ = & 2d \end{aligned}$$

□

Corollary 1. *If $x, y \in \mathbb{R}^n$ and*

$$\mu(x) = \mu(y) = 0 \tag{A.3}$$

and

$$\sigma(x) = \sigma(y) = 1 \tag{A.4}$$

then $\sqrt{\frac{1}{n}}(||x-y||_2) \leq 2$.

Proof of Corollary 1.

$$\begin{aligned} \sigma(y) = \sigma(x) &\overset{\text{Def. 9}}{=} \sqrt{\mu(x^2) - (\mu(x))^2} \\ &\overset{(A.3)}{=} \sqrt{\mu(x^2)} \\ &\overset{\text{Def. 8}}{=} \sqrt{\frac{1}{n}}||x||_2 \end{aligned} \tag{A.5}$$

If we now choose $p = 2$ and $c = \sqrt{\frac{1}{n}}$ then $d \overset{(A.2)}{=} c(||x||_2) \overset{(A.5)}{=} \sigma(x) \overset{(A.4)}{=} 1$.

Hence,

$$\begin{aligned}
&\sqrt{\frac{1}{n}}(||x-y||_2)\\
=&c(||x-y||_2)\\
\overset{\text{Theorem 1}}{\leq}&\ 2d\\
=&2
\end{aligned}$$

□

Proposition 1. *If $x, y \in \mathbb{R}^n$ and*

$$\mu(x) = \mu(y) = 0 \tag{A.6}$$

and

$$\sigma(x) = \sigma(y) = 1 \tag{A.7}$$

then $\frac{1}{n}(||x-y||_1) \leq 2$.

Proof of Proposition 1. Part I:

$$\begin{aligned}
1 \overset{\text{analog to (A.5)}}{=}& \sqrt{\frac{1}{n}}||x||_2\\
1 \overset{\text{Def. 11}}{=}& \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\\
1 =& \frac{1}{n}\sum_{i=1}^{n} x_i^2\\
n =& \sum_{i=1}^{n} x_i^2
\end{aligned} \tag{A.8}$$

Part II:

$$
\begin{aligned}
&\frac{1}{n}(||x-y||_1)\\
\overset{\textit{triangle inequality}}{\leq}&\frac{1}{n}(||x_i||_1+||y_i||_1)\\
\overset{\text{Def. 11}}{=}&\frac{1}{n}\left(\sum_{i=1}^{n}|x_i|+\sum_{i=1}^{n}|y_i|\right)\\
=&\frac{1}{n}\left(\sqrt{(\sum_{i=1}^{n}|x_i|)^2}+\sqrt{(\sum_{i=1}^{n}|y_i|)^2}\right)\\
=&\frac{1}{n}\left(\sqrt{(\sum_{i=1}^{n}|x_i|\cdot 1)^2}+\sqrt{(\sum_{i=1}^{n}|y_i|\cdot 1)^2}\right)\\
\overset{(A.1)}{\leq}&\frac{1}{n}\left(\sqrt{\sum_{i=1}^{n}|x_i|^2\cdot\sum_{i=1}^{n}1^2}+\sqrt{\sum_{i=1}^{n}|y_i|^2\cdot\sum_{i=1}^{n}1^2}\right)\\
=&\frac{1}{n}\left(\sqrt{\sum_{i=1}^{n}(x_i)^2\cdot n}+\sqrt{\sum_{i=1}^{n}(y_i)^2\cdot n}\right)\\
\overset{(A.8)}{=}&\frac{1}{n}\left(\sqrt{n^2}+\sqrt{n^2}\right)=\frac{2n}{n}=2
\end{aligned}
$$

□