UNIVERSITY OF TWENTE.

# Development of Task-Level Programming Framework for Learning from Demonstration

**E.J.C.M. Dankers**
s2138530

Master Thesis
March 18, 2025

**Graduation Committee:**
prof.dr.ir. H. van der Kooij
dr.ir. M. Vlutters
dr.ir. F.J. Wouda

Faculty of Engineering Technology
Mechanical Engineering
University of Twente

# Summary

As robotic systems become more integrated into dynamic environments, enabling non-expert users to intuitively program robots remains a challenge. Learning from Demonstration (LfD) offers a promising solution by allowing robots to acquire skills through human demonstrations instead of explicit programming. Effectively storing and exploiting past user demonstrations is a fundamental challenge. A solution to this challenge is the organization of demonstrations in a skill library. However, this requires robust segmentation, classification, and novelty detection techniques to ensure effective learning and reusability.

This thesis presents a modular framework for skill recognition, segmentation, and novelty detection. The framework enables robots to identify and classify demonstrated motions by matching them to skills in a predefined skill library, ensuring that non-expert users can intuitively program robotic tasks. The system incorporates time series classification using Detach-ROCKET, change point detection for motion segmentation, and novelty detection to identify previously unseen skills. User confirmation is integrated into the process to improve robustness and ensure reliability despite imperfections in human demonstrations.

The framework is evaluated using position data of motion demonstrations recorded with the Franka Research 3 robotic arm via kinesthetic teaching. Results indicate that Detach-ROCKET with MINIROCKET kernels provides high classification accuracy while maintaining computational efficiency. Segmentation methods, including Pruned Exact Linear Time (PELT) segmentation and binary segmentation, exhibit robust segmentation. Additionally, novelty detection using Local Outlier Factor (LOF) effectively identifies new skills, enabling the expansion of the skill library.

By improving skill recognition and segmentation, this framework enhances the usability of LfD, making robot programming more accessible to non-experts. The relevance of this thesis extends beyond technical challenges. It also plays a role in making robotics more accessible in industries facing labor shortages. Future work could explore integrating additional contextual information and alternative teaching methods to further refine the learning process.

# Contents

# List of acronyms

| | |
|---|---|
| **LfD** | Learning from Demonstration |
| **IL** | Imitation Learning |
| **PbD** | Programming by Demonstration |
| **LfO** | Learning from Observation |
| **TSC** | Time Series Classification |
| **CNN** | Convolutional Neurnal Network |
| **MAX** | maximum values |
| **PPV** | proportion of positive values |
| **MTSC** | Multivariate Time Series Classification |
| **ROCKET** | **R**and**O**m **C**onvolutional **KE**rnel **T**ransform |
| **MINIROCKET** | **MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform |
| **MPV** | mean of positive values |
| **MIPV** | mean of indices of positive values |
| **LSPV** | longest stretch of positive values |
| **S-ROCKET** | **S**elective **R**and**O**m **C**onvolutional **KE**rnel **T**ransform |
| **POCKET** | **P**runing Rand**O**m **C**onvolutional **KE**rnel **T**ransform |
| **SFD** | Sequential Feature Detachment |
| **PELT** | Pruned Exact Linear Time |
| **ND** | Novelty Detection |
| **LOF** | Local Outlier Factor |
| **LRD** | Local Reachability Distance |
| **DTW** | Dynamic Time Warping |
| **SVD** | singular value decomposition |
| **LOOCV** | leave-one-out cross-validation |

# Chapter 1

# Introduction

The increasing complexity of robotic applications has driven the need for more flexible and intuitive programming methods. While robots have traditionally been used in industrial environments for pre-programmed, repetitive automation tasks, they are now being introduced into dynamic settings where they interact with humans and adapt to changing conditions. This shift has increased the demand for skilled robot programmers. However, traditional programming methods require specialized expertise, making it difficult for non-experts to teach robots new tasks. This knowledge barrier limits the accessibility and adaptability of robotic systems, particularly in industries where frequent task updates are needed.

To address this challenge, Learning from Demonstration (LfD) has emerged as an alternative approach, allowing robots to acquire new skills through human demonstrations rather than explicit coding. By eliminating the need for expert programming knowledge, LfD makes robot training more accessible to non-expert users. However, despite its advantages, LfD presents challenges in efficiently segmenting, recognizing, and reusing previously learned skills. Without effective frameworks, robots struggle to exploit past demonstrations, leading to inefficient learning processes and limited adaptability.

In literature, various terms are used to address the concept of 'Learning from Demonstration', such as Programming by Demonstration (PbD), Imitation Learning (IL) and Learning from Observation (LfO). Throughout this report, the term LfD will be used to encompass all synonymous terms that refer to this concept.

This study set out to develop a modular framework that integrates time series classification, segmentation, and novelty detection to enhance the intuitiveness and efficiency of task-level robot programming for non-expert users. By addressing key challenges in motion segmentation, recognition, and skill organization, the proposed framework seeks to improve the usability and practicality of LfD-based learning systems.

The remaining part of this report proceeds as follows: Firstly, Chapter 2 examines the context and aims of the project in more detail. Chapter 3 provides the background information relevant to the project and findings regarding these topics in literature. Chapter 4 describes the framework that has been designed and its components. This will be followed by an evaluation of components of the framework in Chapter 5. Finally the limitations, potential improvements, and future extensions of the framework, as well as recommendations for further research will be discussed in Chapter 6. The final chapter provides a conclusion.

Chapter 2

# Problem Definition

As robotics technology continues to evolve, robots are being integrated more and more into a wide range of applications beyond traditional industrial automation. In dynamic environments, robots are expected to perform complex tasks, often requiring adaptability and interaction with human operators. One approach to enabling such adaptability is LfD, a technique in which robots acquire new skills by observing and mimicking human demonstrations. This approach reduces the need for explicit programming, making robot training more accessible to non-experts.

Beyond the technological benefits, LfD enabling non-expert users to program robots has significant implications for addressing labor shortages and accelerating industry [1], [2]. Many industries are experiencing a shortage of skilled robotic programmers [3]. By allowing non-specialists to intuitively program robots to perform tasks, LfD can help bridge the gap between workforce demand and automation, ensuring that businesses remain productive even with a limited pool of skilled programmers [4].

However, for LfD to be practical in real-world scenarios, it must go beyond simple motion replication. Demonstrations need to be processed into structured, reusable skills, allowing robots to generalize from past experiences and efficiently learn new tasks [5]. Without an effective way to recognize, segment, and store these skills, LfD systems remain limited in scalability and usability. This research focuses on addressing these challenges in the context of the Franka Research 3 robotic arm at the Nakama Robotics Lab at the University of Twente.

There are different methods to perform demonstrations to the robots in LfD, e.g. kinesthetic teaching, teleoperation, and passive observation [4]. At the Nakama Robotics Lab, kinesthetic teaching is used to demonstrate movements to the robotic arm. Kinesthetic teaching is a method in which the demonstrator physically guides the robotic arm through the desired motion. The kinematics of each joint and the end effector are recorded.

## 2.1 Key challenges and limitations

A fundamental challenge in LfD is ensuring that demonstrated movements can be effectively stored and reused in future tasks. To achieve this, a skill library can be created, containing templates of skills that the robot has previously learned [1]. Each skill in this library represents a building block that can be used to construct more complex tasks. However, for this approach to be effective, these skills must be meaningful, i.e. they should be modular, parameterizable, and generalizable across different contexts. If skills are not structured properly, they may be difficult to reuse, limiting the flexibility of task composition. The challenge lies in automatically recognizing relevant skills from demonstrations and matching them to existing templates in the skill library.

A second challenge is handling non-expert demonstrations. Non-expert users may have a clear understanding of what a robot should do but often lack the precision to perform perfect demonstrations. This creates inconsistencies in recorded movements, making it difficult for the system to extract reliable skills. If the system were to rely solely on raw demonstrations, it would introduce variability and noise, leading to inconsistent task execution. To address this, the LfD framework should allow non-expert users to demonstrate tasks intuitively, while an algorithm maps these imperfect demonstrations to standardized skill templates. This would ensure that even if different users demonstrate the same task with variations, the robot can still recognize the intended skills and execute them consistently.

Additionally, demonstration often consists of a sequence of multiple skills rather than a single isolated movement. To interpret such demonstrations correctly, the system must segment them into distinct skills. This segmentation process is crucial for enabling robots to understand complex tasks as a composition of reusable skills rather than a single motion trajectory. Additionally, non-expert users may attempt to demonstrate movements that do not exist in the skill library. In such cases, the system must be able to detect novel skills and give the possibility to add these to the skill library. Without robust segmentation and novelty detection, the system risks either misclassifying movements or failing to accommodate new skills, thereby limiting its adaptability.

## 2.2 User Roles

The system is designed to support two primary user roles:

- **Expert demonstrator**: The expert demonstrator is a professional within a certain domain and determines which low-level tasks should be present in a skill library in order to compose more complex high-level tasks. The expert demonstrator is assumed to be able to demonstrate these low-level tasks (or skills) in a desired manner such that the skill is executed with sufficient accuracy and is meaningful for people.
- **Non-expert operator**: This user has enough knowledge to know which low-level tasks (or skills) are needed to compose more complex high-level tasks. However, this person may not have the skills to flawlessly demonstrate these tasks himself. The non-expert user interacts with the system via, e.g. a GUI or physical guidance of the robotic arm to combine skills from the skill library into more complex tasks.

## 2.3 Project Scope

To address these challenges, this project aims to develop a modular framework that enables recognition, segmentation, and novelty detection for robotic skill acquisition. The specific objectives of the framework are:

- A demonstration of a single skill performed by a non-expert user should be recognized as one of the template skills from the skill library, i.e. skills need to be classified.
- A demonstration of a sequence of skills performed by a non-expert user should be recognized as a sequence of template skills from the skill library, i.e. a sequence needs to be segmented and segments need to be classified.
- When the non-expert user demonstrates a skill that does not resemble a skill already in the skill library, this should be detected and the skill can be added to the skill library.

These objectives should contribute to a more intuitive way of programming a robotic arm for non-expert users, the re-usability of expert demonstrations, and getting more insights into the task execution of the robot. All this might lower the barrier to using this type of equipment in various settings. As mentioned, the demonstrations considered in this project are performed using kinesthetic teaching. The time series position data of the end-effector is used for further processing. However, note that there is potential to extend the framework to also work for other teaching methods and data inputs.

# Chapter 3

# Background

This project aims to use time series data of the end-effector position of a robotic arm to segment and classify data to contribute to an intuitive way to teach robots new tasks. Before diving into the content of the project, relevant topics will be discussed to ensure a proper understanding of related fields.

In LfD, human demonstrations often exhibit variability, particularly when performed by non-expert users. To ensure robust and consistent robot execution, this project assumes that an expert has predefined a skill library, which serves as a reference for motion recognition. Given a user demonstration, the goal is to segment and classify the demonstrated actions, mapping them to the most appropriate skills from the library. This approach refines imperfect demonstrations, ensuring that executed motions adhere to predefined standards. To achieve this, several key techniques are employed: Time Series Classification (TSC) for skill recognition, change point detection for segmenting demonstrations, and novelty detection for identifying actions not present in the skill library. This chapter provides an overview of these techniques and their relevance to the problem at hand.

## 3.1 Learning from Demonstration

The applications of robotics are emerging in many fields, ranging from industrial applications such as manufacturing to healthcare. In many cases, humans are still involved since they have to collaborate with or at least instruct or program the robots. It is often attempted to combine the physical capabilities of robots with the cognitive abilities of humans. This is what the LfD framework also attempts to do. In LfD, robots learn to execute new tasks using physical demonstrations performed by humans. There is no need for a robotic expert to program the robot to learn new tasks. This allows people without robotic or programming expertise to teach the robot new tasks. Furthermore, the robot learns from samples that show successful behavior instead of having to figure out the desired behavior itself [6]. Therefore, exploiting expert demonstrations has become one of the most effective ways for robots to acquire skills [7].

Much research on the LfD framework is done in the field of manufacturing and, specifically, assembly. A major shift from mass production towards mass customization can be observed in the field of manufacturing. This increases the need for flexible and adaptable robotic systems to meet the demands of the market [8]. LfD is considered to be a useful solution for this. Robots can be applied not only in industrial settings but also in settings where collaborative robots operate alongside humans. The application of LfD is also considered for these kinds of complex scenarios with uncertain and changing environments [2].

As mentioned, LfD allows people without programming expertise to easily teach and use robotic devices. Working with the LfD framework should therefore be accessible and intuitive for non-expert operators. In [5], the authors aim to bridge the gap between research and practice by providing practical guidelines on deciding what to demonstrate and how to demonstrate depending on the desired task. They conclude that demonstrating sequences of skills that together form a simple task can be efficient. However, for more complex tasks this might not be feasible. In these cases, it is more efficient to divide the task into smaller skills and demonstrate these single skills one by one. Furthermore, they give an overview of several learning methods that are commonly used in LfD, like movement primitives, reinforcement learning, and Gaussian mixture models. Lastly, it is discussed how the LfD learning process can be further refined in order to increase its generalizability, accuracy, and robustness. Enhancing the learning process and generalization performance seeks to align LfD algorithms more closely with human cognitive learning abilities. Improving the accuracy could be done by making the teaching methods even more intuitive and accessible to increase the precision of the demonstrations.

### 3.1.1 Teaching Methods

There are several ways to perform demonstrations to the robots in LfD. The methods most prominently described in literature are kinesthetic teaching, teleoperation, and passive observation as shown in Figure 3.1 [4]. Each of these methods has its own advantages and disadvantages. Therefore, it is important to consider which method is best to be used in certain situations and under certain circumstances [1]. In addition, it is of great importance that the demonstrator is an expert in his field since suboptimal demonstrations are likely to lead to suboptimal robot performance.

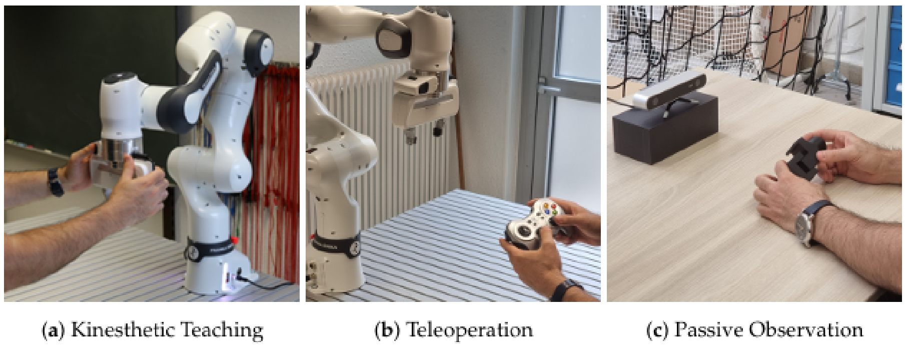


**Figure 3.1:** Common teaching methods in LfD [5]

Kinesthetic teaching is a method in which the demonstrator physically guides the robotic arm through the desired motion. The demonstration is performed in the configuration space of the robot. The kinematics of each joint and the end effector are recorded. A disadvantage of this method is that it can be difficult for the user to accurately perform the desired motion because it can be difficult to manually reach certain positions. Furthermore, the demonstrator's motions might be influenced by the kinematics of the robot.

When using teleoperation, the human demonstrator does not physically interact with the robot. Instead, the demonstrator controls the motions of the robot remotely via an input device, such as a haptic interface or controllers. This requires a robust communication system for control signals and feedback. This method can demonstrate a wider range of tasks than kinesthetic teaching because the user is not constrained to kinematic limitations [9]. Furthermore, teleoperation does not require the user to be in the same room as the robot. However, it requires a more complex setup including a control and communication interface.

The robot acts as a passive observer in the third commonly-used teaching method in LfD [4]. The demonstrator performs a task using his own body and possibly additional sensors while the robot captures relevant data. Data can be captured in various ways, e.g. using cameras, motion capture systems, and sensors. This barely requires training for the demonstrator. The downside of this is that it can be difficult to map the actions of the demonstrator to useful, abstract observations that will be used as input data for the learning algorithm.

### 3.1.2 Intuitive Task-level Programming and Skill Libraries

Task-level programming is commonly used in LfD to compose tasks from one or more predefined skills. It provides an intuitive and flexible approach to robot programming and can be well used in combination with the fast and intuitive teaching approach offered by the LfD framework [10]. Several robotic platforms, such as, Franka Desk [11], ArtiMinds RPS [12], and RAZER [13], incorporate task-level programming, allowing users to construct skill sequences by arranging predefined skills. Task can be composed by dragging skills into a sequence and kinesthetic teaching can be used for parametrization.

Movement primitives and Gaussian mixture models are widely used to model recorded motions. While simple tasks can be represented as a single skill encoded in a single movement primitive, complex tasks benefit from skill sequencing. This approach enables the reuse of generalized skills across multiple tasks, improving adaptability and efficiency. Furthermore, modifying a planned movement becomes easier by replacing one of the skills within the sequence with another one. To facilitate this modular approach, skill libraries serve as repositories for storing learned skills, enhancing reusability and scalability [14].

Several studies have explored methods for integrating skill segmentation, recognition, and task-level programming. In [15], the authors developed a framework that determines the optimal segmentation for a demonstration while simultaneously learning the movement primitives for the segments in an iterative manner. These movement primitives are then stored in a movement primitive library for future use. The correct segmentation is not known beforehand. Therefore, this is treated as a latent variable. A framework that combines task-level programming with semantic skill recognition is proposed in [10]. Tasks are demonstrated via kinesthetic teaching and a semantic skill recognizer is used to extract skills and their parameterization. This process involves constructing a world model that captures object relationships, including relative and absolute poses. The recognized skill sequence is presented in the task-level programming interface. Similarly, in [16], an ontological knowledge base replaces the world model to encode task-specific knowledge and relationships between objects. Schou et al. [2] employ task-level programming in an industrial setting. Pre- and postconditions specified for each skill are used to check if a skill can be executed safely and predictably. Cameras, sensors, and information from previous skills is exploited to get insights into the current state.

## 3.2 Time Series Classification

TSC is used to predict the class for a given instance of time series data. There are many different methods to extract features from time series and classify time series. The various TSC algorithms can be divided in several categories like distance-based, feature-based, interval-based, shapelet-based, dictionary-based, convolution-based, deep learning-based, and hybrid approaches [17].

Time series can be characterized as univariate, involving a single observed variable per time point, or multivariate, where multiple variables are recorded simultaneously. While most research has focused on univariate time series of uniform length, real-world scenarios often involve multivariate time series with varying lengths [18]. Multivariate Time Series Classification (MTSC) is less understood and most classifiers have not been designed to deal with this kind of data [19].

### 3.2.1 Varying length time series

Different cases of time series can differ in the number of time points in the series. The most common causes of varying lengths in time series are 1) variations in sampling rate and 2) variations between the start and the end points of a time series relative to one another [19]. Many of the frequently used time series classification methods cannot handle this and require the input time series to be of equal length. Therefore, a common strategy to deal with varying length time series is to preprocess the time series in order to ensure equal lengths. Consequently, conventional TSC methods can be applied to the uniform time series.

One technique to ensure equal length time series is uniform scaling. This means that shorter series will be interpolated and longer series will be downsampled with respect to another time series. These techniques are specifically efficient if the variation in length is caused by varying sample rates. The sampling rate of a time series may remain constant or vary throughout its duration relative to another time series. The latter can, for example, happen when a user unintentionally accelerates during the recording of a motion.

Another method to equalize time series is padding shorter time series with low amplitude noise. A benefit of this method is that the information in the original series is preserved. The low amplitude noise can be added at the start of the time series, at the end, or both. This method appears to be less successful when the length of the time series differs by a lot.

The most suitable preprocessing method depends on the cause of varying lengths and the classifier used. Therefore, matching the preprocessing technique with the classifier is essential to maximize classification accuracy for the given data type [19].

### 3.2.2 Random Convolutional Kernel Transform (ROCKET)

One of the TSC methods proven to be very time-efficient and accurate is ROCKET. ROCKET is a convolution-based classification method developed by Dempster et al [20]. It has been inspired by Convolutional Neurnal Network (CNN)s which are extensively used for image recognition. Convolutions are used to transform the time series data and pooling operations extract features that will be used for classification. The convolutional kernels used in the ROCKET algorithm are characterized by their length, weights, bias, dilation, and padding. These parameters are drawn from random distributions. More details on the characteristics of the kernels, as well as the ROCKET transform, are given in Appendix B. The standard ROCKET model uses 10,000 random kernels that convolute the input time series. Per kernel, the maximum values (MAX) and the proportion of positive values (PPV) are extracted. These features are used to train a simple linear classifier. This method has proven to achieve state-of-the-art performance while requiring less training time than methods that achieve similar performance [20]. Another advantage of this method is that no assumptions are made about the nature of the time series data.

ROCKET was initially developed for univariate time series but has been extended to support multivariate time series. For multivariate data, kernels convolute with a random combination of channels. The convolution of the data with the kernel is equivalent to the dot product of two matrices. For each kernel, the maximum value and PPV are computed across its assigned dimensions.

There are several variants of ROCKET that use the same basis principle. A set of kernels is used to transform the time series input to a vast amount of features. Subsequently, a linear classifier is fit using this high-dimensional data. **MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform (MINIROCKET) is a faster version of ROCKET [21]. Its accuracy is comparable to the accuracy of the original ROCKET algorithm. This version of ROCKET aims to reduce randomness and speed up the data transformation. Firstly, MINIROCKET uses a smaller, fixed set of kernels. All kernels in MINIROCKET contain nine values. This means that there can be $2^9 = 512$ different kernels. MINIROCKET uses the subset of 84 kernels of which six weights equal $\alpha$ and three weights equal $\beta$. The weights have either value $\alpha = -1$ or value $\beta = 2$. The choice of these values is arbitrary, the only constraint is that the sum of the weights should be equal to zero. This results in less computational effort while maintaining sufficient accuracy. Using these binary kernels allows to perform the convolution operation using additions. This also contributes to a more efficient algorithm. Secondly, MINIROCKET only extracts the PPV from the transformed data. Therefore, it relies on fewer features which simplifies the model. MINIROCKET can deal with multivariate time series in a similar way as ROCKET.

Dempster et al. [20] [21] developed a second variant of ROCKET, called MULTIROCKET. This variant extends MINIROCKET by using three additional pooling operators per kernel. Next to the PPV, it uses the mean of positive values (MPV), the mean of indices of positive values (MIPV), and the longest stretch of positive values (LSPV). Using more pooling operators increases the diversity and expressiveness of the extracted features [22]. In addition, it does not only transform the original time series data but also its first-order difference which estimates the first derivative of the time series. These enhancements increase the classification performance of MULTIROCKET compared to its predecessors at the cost of greater computational expense. The exact definition of all pooling operators is stated in Appendix B.

All variants of ROCKET mentioned above share the same strategy, namely using random convolutional kernels and pooling operators to extract features. Even though this method has proven to deliver great results, there are some downsides. Having a massive amount of random features, not all extracted features might be relevant for the classification task. Even among the relevant features, a significant portion may exhibit strong correlations. Therefore, the high number of features causes more computational effort than necessary, could increase the risk of overfitting and a substantial proportion of randomly generated kernels have a trivial impact on the classification results. These are driving factors to investigate ways to reduce the number of features by eliminating irrelevant features or kernels.

In [23], **S**elective **R**and**O**m **C**onvolutional **KE**rnel **T**ransform (S-ROCKET) is introduced. This variant of ROCKET selects the most important kernels and prunes the less relevant ones while maintaining classification accuracy. It uses a three-step pipeline consisting of 1) pre-training, 2) kernel selection, and 3) post-training. Similar performance was achieved with less than 40% of the original kernels. However, the second step in the pipeline requires many iterations and is therefore time-consuming. Chen et al. [24] proposed **P**runing Rand**O**m **C**onvolutional **KE**rnel **T**ransform (POCKET). This method first removes features that hardly contribute to the classification and then eliminates the corresponding kernels as well. This approach is faster than directly evaluating the kernels. The classifier is refit on the remaining features. Chen et al. proved that POCKET, similarly as S-ROCKET can maintain the performance of ROCKET while pruning more than 60% of the kernels. Recently, Uribarri et al. [25] [26] introduced another method, called Detach-ROCKET, which also has the objective to reduce the number of features. They propose a new algorithm, called Sequential Feature Detachment (SFD), to perform feature selection. Iteratively, features are ranked on importance, and a fixed percentage of the features is removed.

After each iteration, the accuracy of the model is evaluated using a validation set. The optimal proportion of pruned features is determined by an optimization problem. This novel method can eliminate up to 98% of all features without a significant loss of accuracy. However, the training time is increased by almost 20% compared to ROCKET.

## 3.3 Change Point Detection

Change point detection involves identifying transitions within time series data. Change point detection is used in various fields and has many applications, including monitoring medical conditions [27], detecting climate change [28], [29], and analyzing human activity [30]. Time series can be characterized as a sequence of distinct, non-overlapping segments with unique characteristics. Each segment could, for example, represent a motion. Insightful features can be derived from the segments once their temporal boundaries are identified [31]. There are two main categories in change point detection methods: 1) online methods, where changes are detected as soon as they occur, and 2) offline methods, where changes are detected after the entire time series has been collected.

The ruptures python package is a widely used tool for offline change point detection [32]. It provides an accessible and modular framework for applying various detection algorithms to multivariate time series data.
The developers of the rupture package argue that change point detection can be seen as a model selection problem. The best possible segmentation should be chosen so that a cost function is minimized. The most suitable cost function depends on the nature of the task and the prior knowledge available about it [31]. Ruptures has implementations of both parametric cost functions and non-parametric cost functions. Parametric cost functions assume a specific data distribution and rely on parameters, such as mean and variance, to model changes. This offers computational efficiency, but limited robustness if the assumed distribution does not match the actual data. On the other hand, non-parametric cost functions do not make assumptions about data distribution. Data-driven methods, such as empirical distributions or kernel-based measures, are used. This provides greater flexibility and robustness at the cost of greater computational complexity.

The ruptures package implements various change point detection methods, including search methods commonly described in the literature, such as:

- **Linearly penalized segmentation** [33]; Pruned Exact Linear Time (PELT)computes the optimal segmentation that minimizes the cost for a given cost function and penalty level. In contrast to the dynamic programming method, this method can be used when the number of change points is not known beforehand. It includes a pruning step which should reduce the computational cost of the method.
- **Binary segmentation** [34]; This is a greedy sequential method. It tries to find the change points that minimize the costs. The time series data is split up at the location of the detected change point. The number of change points should be known beforehand and is used as input for this method. The change point detection process is repeated until a stopping criterion is met.

The search methods can be used in combination with any of the cost functions implemented in the rupture package or a custom cost function can be defined.

## 3.4 Novelty Detection

Novelty Detection (ND) aims to identify whether new observations belong to novel classes that have not been previously seen in training data. The term 'novelty detection' is often used interchangeably with the terms 'outlier detection' and 'anomaly detection' because they are related. However, they have different meanings. Both ND and outlier detection could be considered as subsets of anomaly detection [35]. Anomaly detection encompasses any method for detecting data points that deviate from the expected observations. The difference between ND and outlier detection is that the former focuses on identifying unlabeled, but valid deviations. It assumes that the novelties are structurally different from the training data and that the training data is not polluted by outliers. Outlier detection, on the other hand, aims to detect extreme, potentially erroneous points or patterns that are often caused by noise or measurement errors [36]. Additionally, in ND the model can often be updated with the newly detected observations. This is not the case for outlier detection [37].

ND algorithms usually first use the original training data to fit a classifier model that is capable of classifying known classes. A popular ND method is the Local Outlier Factor (LOF). This method measures the ratio between the local density of a given sample and the local densities of its k-nearest neighboring points [38].

For each of the k-nearest neighbors of the potential novelty, $p$, the reachability distance is calculated. The reachability distance of point $p$ with respect to point $o$ is defined as:

$$\text{reach-dist}_k(p, o) = \max\{\text{k-distance}(o), d(p, o)\} \tag{3.1}$$

k-distance(o) is the distance of $o$ to its k-nearest and $d(p, o)$ is the distance between point $p$ and point $o$. Consequently, the Local Reachability Distance (LRD) can be calculated for point $p$ and its k-nearest neighbors. The LRD is a measure of the density of the k-nearest neighbors. It is the inverse of the sum of all reachability distances of neighboring points. So, a low reachability distance means a high density. The LRD is defined as:

$$\text{LRD}_\text{k}(p) = 1/\left(\frac{\sum\limits_{o\in N_\text{k-distance}(p)} \text{reach-dist}_\text{k}(p, o)}{|N_\text{k-distance}(p)|}\right) \tag{3.2}$$

in which $N_\text{k-distance}$ is the $k$-distance neighborhood of point $p$.

The LOF, which is the ratio of the average LRDs of the neighboring points and the LRD of the potential novelty, can be calculated with:

$$\text{LOF}_\text{k}(p) = \frac{\sum\limits_{o\in N_\text{k-distance}(p)} \frac{\text{LRD}_\text{k}(o)}{\text{LRD}_\text{k}(p)}}{|N_\text{k-distance}(p)|} \tag{3.3}$$

Points located within a cluster will have a LOF value of approximately one or less than one. For a value larger than one, the object is considered to be a novelty.

Chapter 4

# Framework

The objectives of the project included designing a framework that can 1) recognize skills, 2) segment sequences of skills, and 3) detect whether a skill has already been taught or not. In this chapter, the developed framework will be discussed in detail. First, a general overview of the framework will be given in Section 4.1, after which specific parts will be discussed more elaborately in the remainder of the chapter.

## 4.1 General Outline

The framework is implemented in modular components. The main outline of the framework will first be explained before diving deeper into the distinct components that form this framework.

### 4.1.1 Expert Demonstrations & Preprocessing

An expert user performs demonstrations of single skills relevant in a certain field. These recorded skills are labeled and saved in a skill library. These time series data of the motions form the basis for the framework. Therefore, skills in the skill library must be useful and meaningful in the context of a certain field or task.

Before these demonstrations are further used, they are preprocessed. The time series in this project are mostly of unequal length. Even if executed by the same person, the duration of the motion is very likely to vary. All time series shorter than the longest one are interpolated to ensure uniform length across the all time series. (Possibly, all motions are aligned along the same axis.) Data is not aligned temporally, using for example Dynamic Time Warping (DTW). Differences in timing are likely to occur during demonstrations and these imperfections should not hinder the classification. Furthermore, temporal alignment using DTW is not possible when the class of a motion is not known yet since DTW aims to find a minimal cost alignment between the time series obtained by demonstration and a reference time series.

To reduce the need for extensive manual demonstrations by the expert and ensure the classifier can handle imperfect demonstrations from non-expert users, data augmentation is applied after preprocessing. This serves two main purposes: first, it artificially increases the number of training samples, minimizing the effort required from the expert while enhancing the model's robustness. Secondly, by adding 'imperfections,' the classifier learns to recognize skills despite natural inconsistencies in demonstrations.

### 4.1.2 Time Series Classification

All time series used for training are transformed by convolving ROCKET kernels with the time series data. The maximum values and the proportion of positive values are extracted for each kernel. The transformed data set has shape $(n_{\text{samples}}, 2 \cdot n_{\text{kernels}})$. This data is used to fit the classifier. A simple linear classifier, like RidgeClassifierCV, is usually used in combination with ROCKET. The transformed data is also used to fit the novelty detector.

### 4.1.3 Segmentation

When a non-expert performs a demonstration, it should first be clear whether this is a single motion or a sequence of motions. In the case of a single motion, the time series can directly be preprocessed. When the demonstration is a sequence of consequential motions, these need to be segmented. A change point algorithm is used to detect the change points in the sequence. The detected change points are proposed to the user for confirmation. If the detection change points appear meaningful, they will be used to segment the data sequence. If one or more of the detected change points do not seem to be placed reasonably, the user can manually adjust the change points before the sequence is split up into segments. The original time series data can be used as input in the segmentation method, but the ROCKET transformed data could be used as well. Both options have been considered and are discussed in Section 4.5.2 and Section 4.5.1, respectively.

### 4.1.4 Novelty Detection

The transformed segments are input for the novelty detection algorithm. This algorithm predicts whether a motion segment is a skill from the skill library or a new motion that has not been seen previously. When the motion is a known skill, it can be classified. Once a new motion has been detected, the user will be asked to confirm that this might be a new skill. If this is indeed the case, the segment will be labeled by the user, and a new class will be made. More samples for this class will be generated via data augmentation. The new samples will be transformed and added to the training data to re-fit the classifier to ensure continuous learning.

### 4.1.5 User Confirmation

Some checkpoints for user confirmation have been added to make the segmentation and novelty detection process more robust. When the system fails to detect change points or novelties correctly, this can be adapted manually. Instead of trying to solve complex segmentation tasks autonomously, the human operator is kept in the loop by taking advantage of its knowledge and intuition. In the end, decisions made regarding segmentation and detection of new motion should be meaningful to humans. The user confirmation also reduces the chance of a mismatch between the user's intention and the robot's interpretation.

### 4.1.6 Next step after the framework

The full framework is shown in Figure 4.1. The workflow of the framework ends when all task skills have been classified or when new skills have been recognized. This output gives the user feedback on how the system has interpreted the demonstrated task. The next step would be to let the robot execute the task. Characteristics or parameters can be extracted from the demonstration data and used to adapt encoded template skills from the skill library. These templates could be in the form of, for example, dynamic movement primitives. Dynamic features could be extracted from the human demonstration to auto-tune parameters of dynamic movement primitives. [39]. The precise implementation of these next steps is beyond the project scope and will not be addressed in this report.

## 4.2 Preprocessing

Both the demonstrations recorded by the expert and the time series data from the non-expert are preprocessed. The preprocessing consists of three main steps: 1) downsampling, 2) interpolation, and 3) spatial alignment. Furthermore, the start and end of the recordings are trimmed if the robot is not moving.

### 4.2.1 Downsampling

When performing the demonstration, the data is logged with a certain frequency. The time series data will contain a more detailed representation of the motion when a higher frequency is used. However, not all details are necessary for subsequent actions like classification and novelty detection. Therefore, both the skills in the skill library and the motion demonstrated by the user are downsampled. Having a more compact representation of the motion is more time-efficient when convolving the ROCKET kernels over the time series data.

### 4.2.2 Interpolation

As mentioned in Section 3.2.1, various TSC methods cannot deal with time series of varying lengths. This also holds for ROCKET since the dilation of the kernels depends on the length of the time series. When the ROCKET kernels are generated, the dilation is calculated such that the longest kernel is not longer than the time series. However, this is based on one time series sample. Therefore, shorter time series are interpolated to ensure they have at least longer than the longest kernel. Padding shorter time series is not used because it affects the PPV. Cubic polynomials are used to ensure that the interpolated time series, as well as its first and second derivatives, are continuous. If the lengths between the original time series and the interpolated time series differ too much, this could influence the ROCKET transformation. However, it is assumed that the time series lengths of all individual skills in this project are in the same order of magnitude.

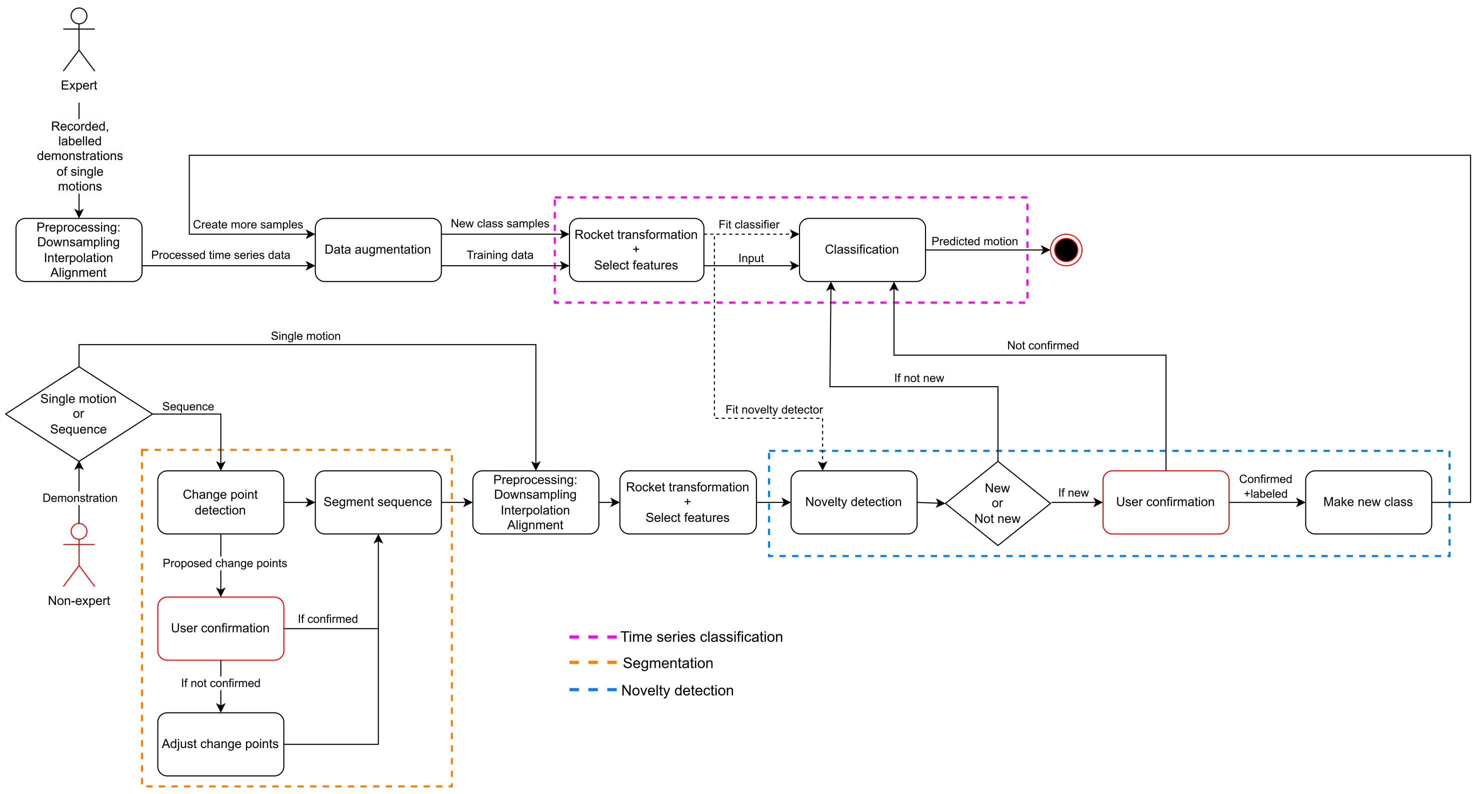


**Figure 4.1:** Framework overview; The colored dashed boxes indicate the main components of the framework: 1) segmentation, 2) TSC, and 3) novelty detection. The black dot marks the end of the framework. At this point, a sequence of motions is segmented and each segment is classified or detected as a new motion.

### 4.2.3 Spatial Alignment

When data is transformed using ROCKET kernels, similar motions in different directions can end up in different places in the feature space. When all dimensions are used by all kernels, there is no clear difference between drawing a line upward (positive $y$-direction) and a line to the right (positive $x$-direction). These result in almost the same features. However, default ROCKET uses a random combination of channels when transforming the data. In that case, a line upwards and a line to the right will result in slightly different features.

Being invariant with respect to direction or orientation when classifying motions might be desirable. However, one could argue that the importance of the orientation and direction of motions is task-specific. In some situations, the orientation/direction of motions does not have any meaning, while in another context it might have. This will be further discussed in Section 6.2. For now, it is assumed that at least some level of invariance is desired since this results in a more compact representation of a motion.

In order to ensure that comparable motions that have been performed in different directions map to comparable places in the feature space, some preprocessing happens before the data is transformed. For each motion or motion segment, the mean position is calculated and subtracted. So, all motions are translated to the same origin. Then singular value decomposition (SVD) is performed to obtain a factorization of each motion. Position data $X$ can be decomposed into an orthogonal matrix $U$, a diagonal matrix of singular values $\Sigma$ and another orthogonal matrix $V^T$, as shown in Equation 4.1.

$$X = U\Sigma V^T \tag{4.1}$$

This factorization separates the motion into rotational components, encoded in $U$ and $V$, and scaling factors, encoded in $\Sigma$. The equation can be rewritten to isolate the motion along its principal components, aligned by their singular values:

$$U^{-1}X = \Sigma V^T \tag{4.2}$$

This transformation standardizes the orientation of the motion by factoring out the influence of $U$.

## 4.3 Data Augmentation

As mentioned previously, data augmentation serves two main purposes: 1) increasing the number of training samples to enhance the robustness of the classifier without requiring the expert to manually record numerous demonstrations, and 2) introducing controlled variations to account for imperfections in demonstrations performed by non-expert users. By applying different types of data augmentation to the original time series data, the classifier becomes more adaptable to variations in execution while still recognizing the intended skills. Figure 4.2 illustrates the original, unaugmented data for three different motions.

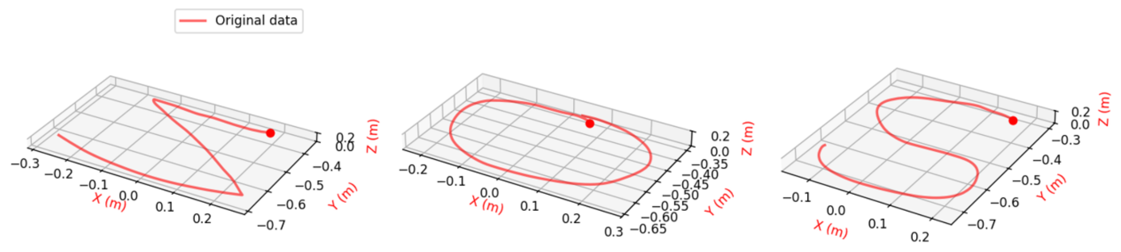


**Figure 4.2:** Original time series data of mirrored Z-shape, CCW circle, and S-shape. The red dot marks the starting point of the motions.

### 4.3.1 Drift

First of all, drift is added to the expert skill demonstrations. This causes time series values to deviate from the original values in a random and smooth manner. The drifting is added independently for each channel in an additive way. This simulates deviations that can occur from the original motion trajectory when a non-expert performs demonstrations. Figure 4.3 shows three generated samples for each of the motions shown previously.

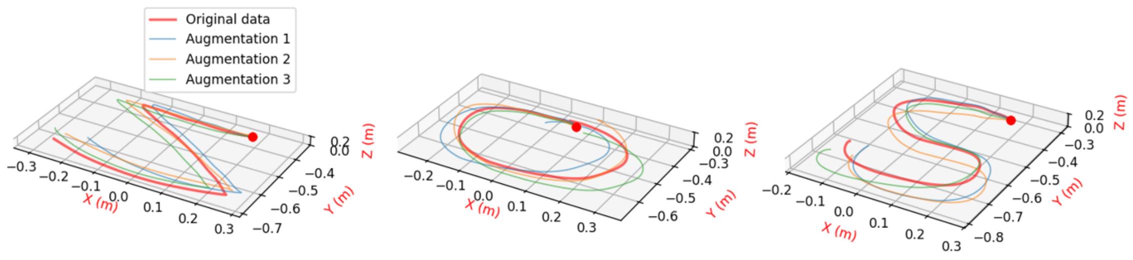


**Figure 4.3:** Augmented time series data of mirrored Z-shape, CCW circle, and S-shape. Additive drift has been added to the original time series data. The red dot marks the starting point of the motions.

The effect of drift is clearly visible. All samples slightly deviate from the original trajectory. Figure 4.4 shows the position data of the z-shape motion over time. The variations due to the added drift are proportional to the variance in each direction.

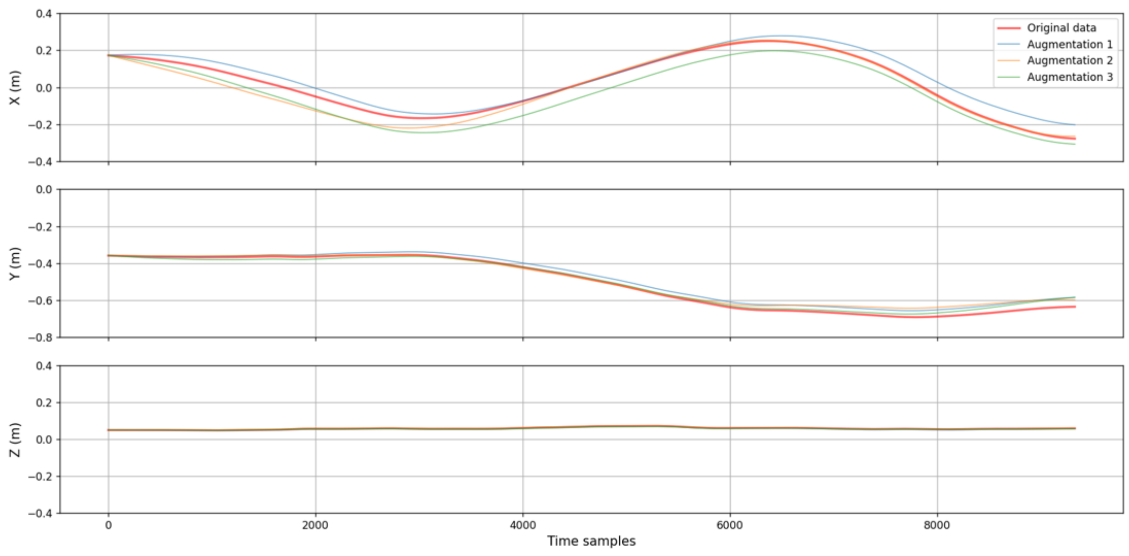


**Figure 4.4:** Effect of the added drift over time for the mirrored Z-shape.

### 4.3.2 Time warping

Time warping changes the speed of the timeline of the time series. This is likely to happen in reality as well because users might demonstrate skills with varying speeds. The number of speed changes can be controlled when augmenting the data as well as the ratio between the maximum and minimum speed. Time warping does not change the shape of the motion, as can be seen in Figure 4.5.

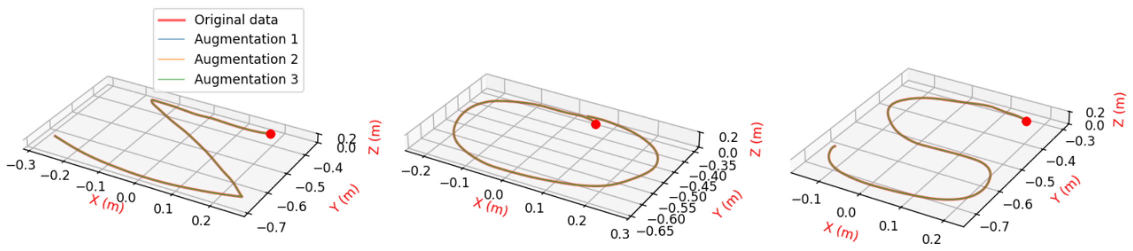


**Figure 4.5:** Augmented time series data of mirrored Z-shape, CCW circle, and S-shape. Random time warping has been added to the original time series data. The red dot marks the starting point of the motions.

The effect of time warping can be seen when observing the timeline of the positional data of the z-shape in each direction, shown in Figure 4.6.

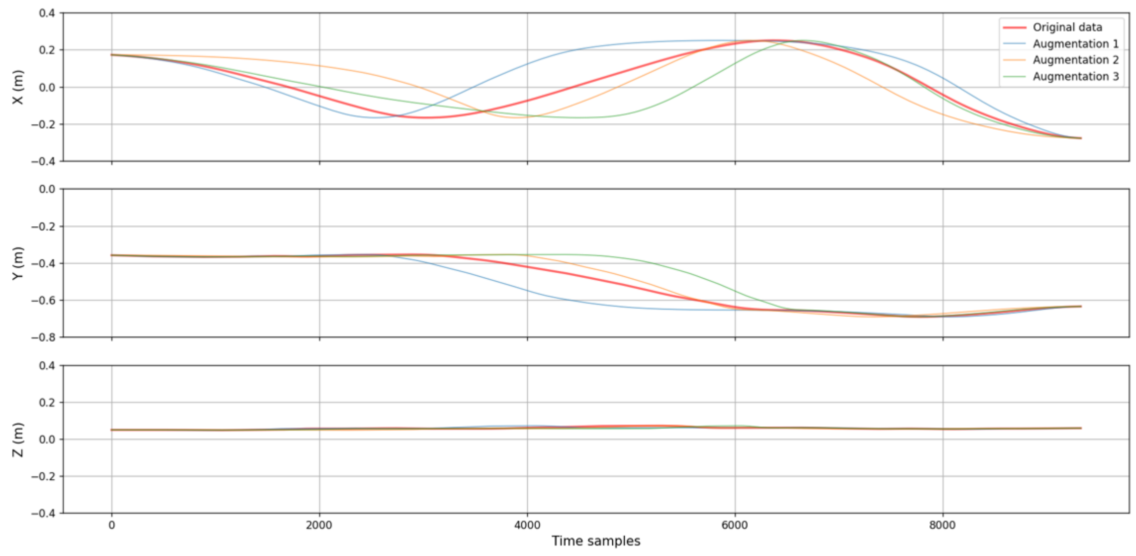


**Figure 4.6:** Effect of the added time warping over time for the mirrored Z-shape.

The data used for further processing is augmented with both drift and time warping. The combination of these effects should simulate the varying and imperfect behavior of humans. For each demonstration in the skill library, 100 augmented samples are created.

## 4.4 Time Series Classification

TSC using ROCKET consists of two main steps: 1) transformation of the data using the random kernels and 2) classification. Exact details on the kernels and the classifier are given below.

### 4.4.1 Convolutional Kernels

Detach-ROCKET is used to transform the position data of the end-effector, as it effectively removes a significant part of irrelevant features as explained in Section 3.2.2. It can be used in combination with ROCKET, MINIROCKET, or MULTIROCKET kernels and pooling operators. The default version for Detack-ROCKET is ROCKET. However, the authors of [20]–[22] suggest using MINIROCKET as the default version because it is more time-efficient and performance similar to ROCKET. Which version performs best and is most time-efficient is tested in the evaluation. The characteristics of all kernels and a more detailed description of the ROCKET and SFD algorithm can be found in Appendix B.

The feature elimination process results in a feature mask. The recorded data from the skill library is used to fit the Detach-ROCKET instance and create the feature mask. When a new input is given, the data is transformed using all the kernels. The features mask is then applied to eliminate part of the features. This feature elimination reduces the dimensionality of the transformed data, which is advantageous for downstream tasks such as novelty detection. Detection accuracy can be affected in higher dimensions. For this reason, reducing the dimensionality of the data enhances the performance and accuracy of algorithms like LOF. The phenomena of challenges and complications arising when analyzing data in high-dimensional spaces is referred to as the curse of dimensionality. As the number of dimensions increases, more data is required to analyze and resolve problems adequately.

### 4.4.2 Classifier

The features extracted from the convoluted data are used as input for the classifier. ROCKET could potentially be used with all classifiers. However, Dempster et al. [20]–[22] suggest the use of a ridge regression classifier in combination with ROCKET. Therefore, the scikit-learn ridge regression classifier is used [40]. The classifier performs leave-one-out cross-validation (LOOCV) to determine the regularization parameter $\lambda$ that reduces the chance of overfitting. The optimization problem to be solved is defined as follows:

$$\hat{\theta}_t^{\text{ridge}} = \arg\min_{\theta} \left\{ \sum_{i=1}^{N} \left( y_i - \theta_0 - \sum_{k=1}^{F} x_{ik}\theta_k \right)^2 + \lambda \sum_{k=1}^{F} \theta_k^2 \right\} \tag{4.3}$$

With $N$ being the number of training samples and $F$ being the number of ROCKET features. $y_i$ are the target labels.

For a multiclass classification problem, the ridge classifier applies a one-vs-rest strategy, meaning that one classifier is trained for each class. Each classifier learns to distinguish between one class and all other classes. For each class $k$, there is a hyperplane, defined by its weight vector $w_k$ and bias $b_k$. The hyperplanes represent the decision boundaries that separate one class from the others. Once the model is trained, the decision function calculates the distance of each test sample $x$ to each class-specific hyperplane. These values form the confidence score for $x$ being in each class $k$. If the confidence score is positive, the sample is classified as belonging to that class. If the confidence score is negative, the sample is considered not to belong to the class. The magnitude indicates the distance from the sample to the hyperplane. The sample is assigned to the class with the highest confidence score. In other words, the sample is assigned to the class for which it has the highest positive distance.

## 4.5 Segmentation

Several methods have been considered for the segmentation of sequences of motions. Initially, an online segmentation method was developed using sliding windows. Even though this method showed promising results, it comes with challenges that make it unsuitable for the framework. An elaboration on the method and its challenges will be provided in Section 4.5.1. Alternatively, the offline methods provided in the ruptures python package have been considered and proved to fit better in the framework, as discussed in Section 4.5.2.

### 4.5.1 Online Method

This method uses a window to slide over the time series data. The window segment is transformed using an ensemble of ROCKET transformers. Then, a class label is predicted for the given window segment, and the probability estimates of the class labels are obtained. This happens iteratively. However, in contrast to original sliding window methods, the window size is not constant. A minimum window size is set beforehand and the size increases each iteration with a fixed amount until the probability estimate of the predicted class label is above a certain threshold for $n$ times in a row. The prediction probability estimate threshold and $n$ have to be determined beforehand. When the specified condition is met, the window moves to the next part of the time series, the size of the window is reset to the minimum window size, and the same process is applied to detect the subsequent motion.

An advantage of the method is that it does not require information about the number of motions present in the sequence or the length of the time series. Some examples of the segmentation and classification of a 2D data sequence of letters are shown in Figure 4.7.

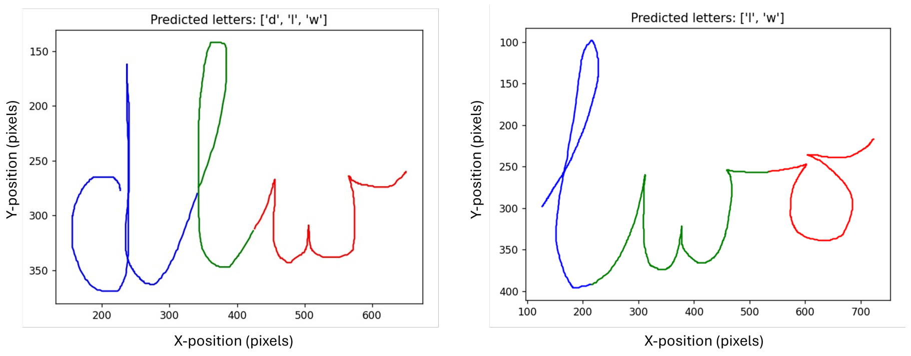


**Figure 4.7:** 2D letter sequences segmented using the increasing sliding window method. In the left figure, all letters have been classified. In the right figure, the last letter has not been classified because the threshold was not exceeded because the end of the time series had already been reached.

Despite the promising result of the left plot, the method comes with several challenges. Firstly, errors in segmentation and classification at the start can propagate, resulting in incorrect predictions throughout the rest of the sequence. Secondly, the final motion in the sequence may remain unrecognized if the prediction threshold is not met by the end of the time series, as can be seen on the right-hand side in Figure 4.7. The algorithm was close to detecting the final letter of the sequence but did not meet the condition yet. This issue could be mitigated by extending the time series or fine-tuning hyperparameters. The performance of the method is highly influenced by hyperparameters such as the minimum window size, window increment, $n$, and prediction probability threshold, which are linked to the velocity of motion execution. Consequently, the accuracy is highly dependent on the demonstration velocity.

In the context of the rest of the framework, there are some additional considerations. Combining this method with novelty detection is particularly challenging, as it is difficult to discern whether a window segment represents a part of a known motion whose continuation might still unfold beyond the current window or an entirely new motion. This does not play a role in offline change point detection where the time series data is first segmented and then transformed and classified. Furthermore, preprocessing motion segments is more straightforward when they are predefined. For these reasons, offline methods are considered to be more suitable in the framework.

### 4.5.2 Offline Method

As mentioned in Section 3.3, the ruptures python package is commonly used for offline change point detection. Also in this project, methods implemented in ruptures are used to detect the change points in sequences of motions. The complete time series containing position data of a demonstrated sequence is the input for the segmentation method.

#### Search Method

The PELT method is used to detect the change points. This method is more accurate than approximate alternatives as it computes the exact solution. In addition, it is faster than other exact methods [41]. The algorithm requires either the number of change points in the sequence or a complexity penalty. This complexity penalty is related to the amplitude of the changes. If the penalty is too small, too many change points are detected. If the penalty is too big, only the most significant changes are detected, or none [33]. Different penalty values are evaluated in Section 5.2.1.

Some other hyperparameters can need to be set:

- **min_size**; determines the minimum size of a segment. Each motion in the skill library contains at least 200 samples, so this parameter is set to 200.
- **jump**; defines the interval between possible change points, such that only every $k^{th}$ point (where $k$ = jump) is evaluated. This can reduce the computational complexity and control the resolution of change point detection. The jump parameter equals one in this project. In this way, each sample could be a potential change point.

Alternatively, binary segmentation is a good option when the number of change points is known beforehand. This method does not compute the exact solution and is therefore more time-efficient. The values for min_size and jump stated above can be the same for this method.

#### Cost Function

A cost function is minimized to detect the change points. The choice of the cost function depends on the type of data being analyzed and the goal of the analysis. There are parametric cost functions, as well as non-parametric cost functions available in ruptures. A non-parametric cost function is chosen to minimize assumptions about the underlying data distribution. The following cost function is used:

$$c(x_{a..b}) = (b-a) - \frac{1}{b-a} \sum_{s,t=a+1}^{b} \exp\left(-\gamma \|x_s - x_t\|^2\right) \tag{4.4}$$

with $x_{a..b}$ is the segment $\{x_t\}_{t=a+1}^{b}$ $(1 \leq a < b \leq T)$ taken from multivariate time series $\{x_t\}_{t=1}^{T}$. $a$ is the starting index of a segment, $b$ is the ending index. $\gamma$ is the so-called bandwidth parameter. The cost function in Equation 4.4 uses a Gaussian kernel to map the original signal onto a higher-dimensional Hilbert space to better capture nonlinear patterns in the signal.

This cost function was chosen because it handles the nonlinear and multidimensional motions of robotic arms and captures complex trajectory transitions better than cost functions that only detect changes in mean or variance.

## 4.6 Novelty Detection

The LOF method, as described in Section 3.4, is used for novelty detection in this framework. The preprocessed and ROCKET transformed data is used to fit the novelty detector. The LOF algorithm implemented by scikit-learn is by default only meant for outlier detection. It requires the 'novelty' parameter to be set to true if used for novelty detection. When set to true, the LOF detector is fit on the training data, the data in the skill library, and all new inputs are considered as potential new observations. The number of neighbors used for which the local densities are calculated can be set. This value is at least two and at most all provided samples. A too small number of neighbors might result in undesirable statistical fluctuations. It is suggested to use at least as many neighbors as there are objects in a cluster [38]. The Euclidian distance is used as the distance metric.

Chapter 5

# Framework Evaluation

For the evaluation of the framework, a dataset of 3D trajectories has been recorded through kinesthetic teaching with the Franka Research 3 robotic arm. The logging frequency of the recorded motions is 1000Hz. The individual skills that have been recorded can be divided into four categories and are listed in Table 5.1. All skills have been recorded twice.

**Table 5.1:** Recorded skills categorized in four distinct categories. '+x' and '-x' indicate movement in the positive and negative x-direction, respectively, while '+y' and '-y' refer to movement along the y-axis.

| Lines | Circle-shapes | S - Z shapes | Up-down-up-down |
|---|---|---|---|
| Line +y | CCW circle | S-shape | Up-down-up-down y-direction |
| Line -y | CW circle | Mirrored Z-shape | Up-down-up-down z-direction |
| Line +x | CCW semi-circle | | |
| Diagonal line +x +y | | | |

To test the segmentation of sequences of skills, two different sequences have been recorded. Both sequences consist of three skills, as shown in Table 5.2.

**Table 5.2:** Recorded sequences including the individual skills in each sequence.

| Sequence 1 | Sequence 2 |
|---|---|
| Line +x | S-shape |
| Up-down-up-down z-direction | Up-down-up-down z-direction |
| CCW circle | Mirrored Z-shape |

A visual overview of the recorded skills is shown in Appendix C. The position data of all recordings is expressed in the base frame of the robot, which is shown in Figure 5.1.

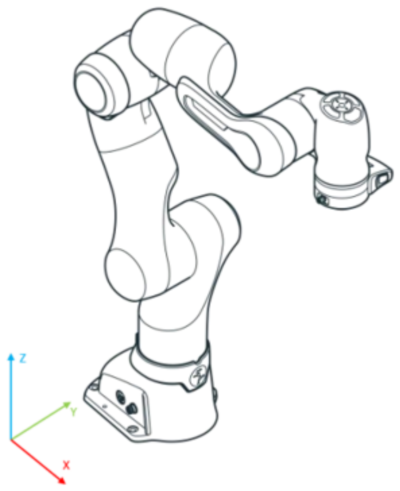


**Figure 5.1:** Franka Research 3 and the reference frame in which all motions are expressed.

## 5.1 Classification evaluation

The classification has been tested by evaluating the success rate when classifying skills from the skill library. Detach-ROCKET is used for the skill classification. All recorded skills have been divided over a training set and a test set in order to evaluate the classification. Each set contains one recording of each skill stated in Table 5.1.

The data of both datasets is preprocessed in the same way. First, the data is downsampled by a factor of 10 to reduce computational load. Then, for each recording in the training set, 100 augmentations are generated with added drift and time warping. The augmented training set is transformed using 10,000 MINIROCKET kernels. For each kernel convolution, the PPV is calculated. Afterward, the SFD algorithm, as proposed in [25], is used to eliminate part of the features. Over 95% of the features are pruned. Finally, the remaining features of the transformed training set are used to fit the classifier.

Similarly, the test set is transformed and classified. The test set was classified with an accuracy of 100%, meaning that all 11 skills were classified correctly.

### 5.1.1 Use of different ROCKET versions

Detach-ROCKET can be used with the kernels and pooling operators of ROCKET, MINIROCKET, or MULTIROCKET. All three options have been considered. To find the most suitable option, they have been tested on classification accuracy and time efficiency. In all three scenarios, 10,000 kernels have been used. The results in Table 5.3 show that MINIROCKET is more time-efficient than the other versions while classifying all motions correctly.

**Table 5.3:** Classification accuracy using different versions of ROCKET and time needed to fit the Detach-ROCKET object. Fitting the Detach-ROCKET object includes the SFD process to select the most relevant features and fit the classifier.

| ROCKET kernels | Classification accuracy | Time (s) |
|---|---|---|
| ROCKET | 92% | 409.43 |
| MINIROCKET | 100% | 20.65 |
| MULTIROCKET | 100% | 128.70 |

### 5.1.2 The Effect of Downsampling and Interpolation

It has been evaluated how the downsampling of the test data influences the classification performance. Manipulating the time series data can have a negative impact on classification accuracy because it changes the information content of the time series. The preprocessing step in the framework includes downsampling the time series, mainly to increase time efficiency, and subsequently, time series are interpolated to match the longest time series used to generate the random kernels.

At the beginning of this evaluation, the training set is still downsampled by a factor of 10, where the samples in the training set have a length of 850 time instances after downsampling. Then the downsampling factor of the test set is increased by steps of 10 until the classification accuracy drops. This happens when the downsampling factor reaches 100, after which the data in the test set consists of 94 time instances. At that point, one of the skills in the test set is not predicted correctly anymore. The mirrored Z-shape is predicted as an S. This can be expected since the mirrored Z and the S follow a similar trajectory. When the motion trajectory of a mirrored Z-shape is excessively downsampled, the points that define its sharp angles are likely to be lost. During interpolation, the limited sampled time points cannot fully reconstruct the original abrupt changes, resulting in a smoother trajectory that resembles an S-shape. This happens because interpolation algorithms naturally favor smooth curves over sharp angles when there is insufficient data resolution.

## 5.2 Segmentation Evaluation

The segmentation evaluation focuses on inter-rater reliability, agreement between the user and algorithm, and consistency of the algorithm outputs. This is done instead of comparing the detected change points to an objective ground truth because the nature of a good segmentation is subjective. Furthermore, segmentation is the first step of the framework, so the segmentation output influences downstream processes. Therefore, the influence of several hyperparameters on the segmentation outcomes will be shown in this section. Lastly, the classification accuracy of the segments produced by different methods will be analyzed, as well as the segments resulting from the human segmentation.

### 5.2.1 Segmentation Consistency

The consistency of segmentation algorithms in finding the change points in several demonstrations of the same sequence is analyzed. Assessing segmentation consistency is important because large variations in detected change points across repeated demonstrations can impact the reliability of downstream processes, such as classification. The sequence used for evaluation has been demonstrated five times in total by three different individuals. The five demonstrations have been preprocessed to ensure equal lengths and have been downsampled by a factor of 10 for time-efficiency reasons. This allows for easier comparison between the different demonstrations. After downsampling, the time series of the demonstrated sequences contains 2163 time instances. Each demonstration has been segmented using the same method and hyperparameters. The location of the change points can be compared to check the consistency of the segmentation algorithm's output. The segmentation consistency is analyzed using two segmentation algorithms. Firstly PELT segmentation, which can be used without explicitly giving the number of change points, is used. Secondly, binary segmentation is used, which assumes the number of change points is known. The five demonstrations of the sequence of skills used for this evaluation are shown in Figure 5.2.

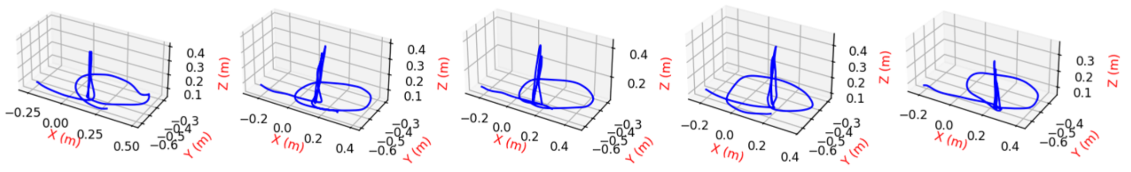


**Figure 5.2:** Five demonstrations of a motion sequence consisting of 1) line in $x$-direction, 2) up-down-up-down in $z$-direction 3) CCW circle

#### PELT Segmentation

PELT segmentation has been used to segment the sequence of motion skills. The location of the change points in all five demonstrations is shown in Figure 5.3. All demonstrations have been split into four segments due to the resemblance between a CCW circle and two CCW semi-circles. Although the expected segmentation was three segments, the algorithm's result is not necessarily incorrect, as both a full CCW circle and two CCW semi-circles represent valid motion patterns. The detected change points are closely aligned across demonstrations. The detected change points that can be seen in the plot are also stated in Table 5.4. This shows that the algorithm is capable of providing a consistent segmentation despite the imperfections induced by different demonstrators.

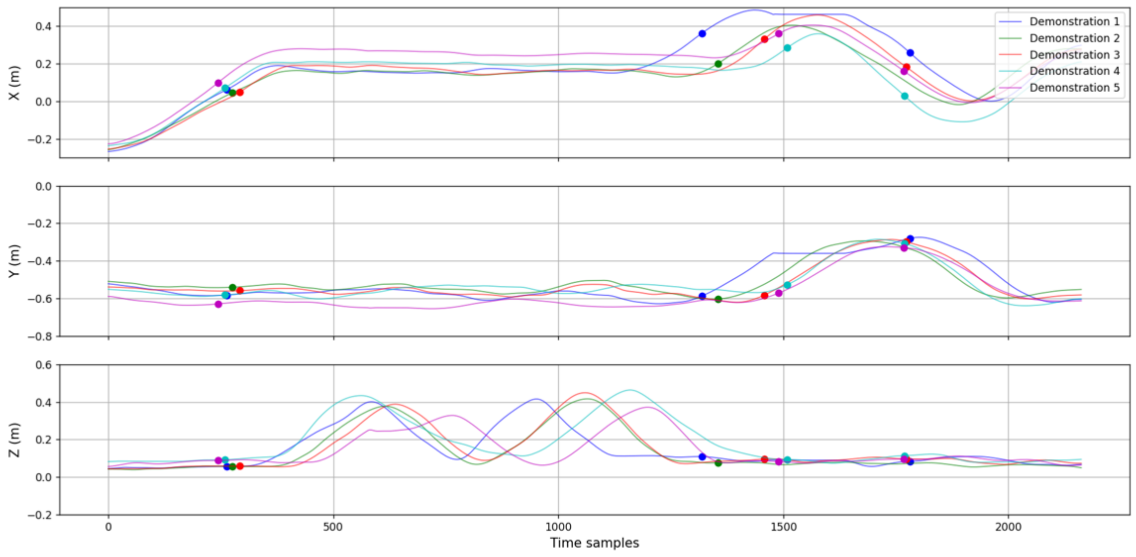


**Figure 5.3:** Position in x-, y-, and z-direction over time of all five demonstrations. The dots indicate change points detected by PELT. The penalty value is set to 100.

Increasing the penalty value promotes a segmentation with fewer change points. Table 5.4 shows at which time instances change points have been detected for penalty values of 100 and 140 respectively. It can be noted that for a higher penalty value, the algorithm tends to split up the sequence into three parts. The two last segments, two semi-circles, are combined into one circle. One of these segmentations is not better than the other, the preferred segmentation depends on how skills in the skill library have been defined and demonstrated.

Table 5.4: Detected change points using PELT with different penalty values. The detected change points are reported in sample numbers in the time series.

| | | Detected change points |
|---|---|---|
| PELT | Demonstration 1 | 264, 1320, 1782 |
| penalty = 100 | Demonstration 2 | 276, 1417, 1764 |
| min_size = 200 | Demonstration 3 | 293, 1459, 1774 |
| downsampled x 10 | Demonstration 4 | 260, 1510, 1770 |
| | Demonstration 5 | 245, 1491, 1769 |
| PELT | Demonstration 1 | 264, 1320, 1782 |
| penalty = 140 | Demonstration 2 | 277, 1356 |
| min_size = 200 | Demonstration 3 | 293, 1435 |
| downsampled x 10 | Demonstration 4 | 260, 1533 |
| | Demonstration 5 | 245, 1508 |

### Binary Segmentation

The segmentation consistency has also been evaluated for binary segmentation. This method allows providing the number of change points as input. The number of change points is set to two. Figure 5.4 shows the segmentation for each demonstration using binary segmentation. The corresponding change points are stated in Table 5.5. The change points are located on quite similar positions across all demonstrations.

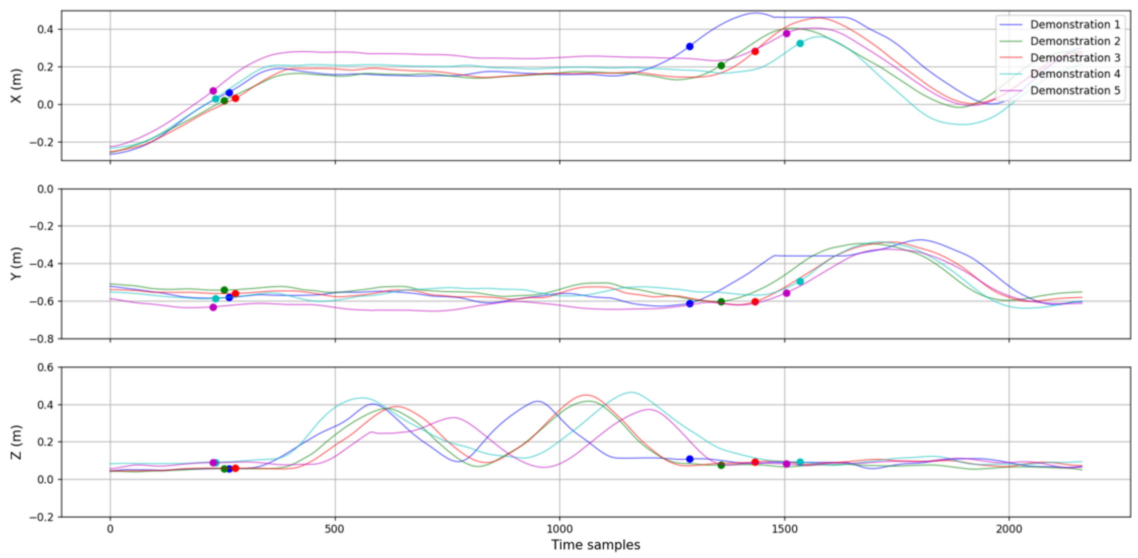


**Figure 5.4:** Position in x-, y-, and z-direction over time of all five demonstrations. The dots indicate change points detected by PELT. The number of change points is set to 2.

When changing the number of change points to be detected to 3, the sequence will be split into four segments. An extra change point is added in the last segment, resulting in two semi-circles instead of a full circle, similar to the PELT segmentation. An overview of the detected change points is given in Table 5.5.

Table 5.5: Detected change points using binary segmentation with different values for n_changepoints. The detected change points are reported in sample numbers in the time series.

| | | Detected change points |
|---|---|---|
| Binary segmentation<br>n_changepoints = 2<br>min_size = 200<br>downsampled x 10 | Demonstration 1 | 265, 1290 |
| | Demonstration 2 | 255, 1360 |
| | Demonstration 3 | 280, 1435 |
| | Demonstration 4 | 235, 1535 |
| | Demonstration 5 | 230, 1505 |
| Binary segmentation<br>n_changepoints = 3<br>min_size = 200<br>downsampled x 10 | Demonstration 1 | 265, 1290, 1780 |
| | Demonstration 2 | 255, 1360, 1760 |
| | Demonstration 3 | 280, 1435, 1770 |
| | Demonstration 4 | 235, 1535, 1780 |
| | Demonstration 5 | 230, 1505, 1775 |

### 5.2.2 The Effect of Downsampling on Segmentation Consistency

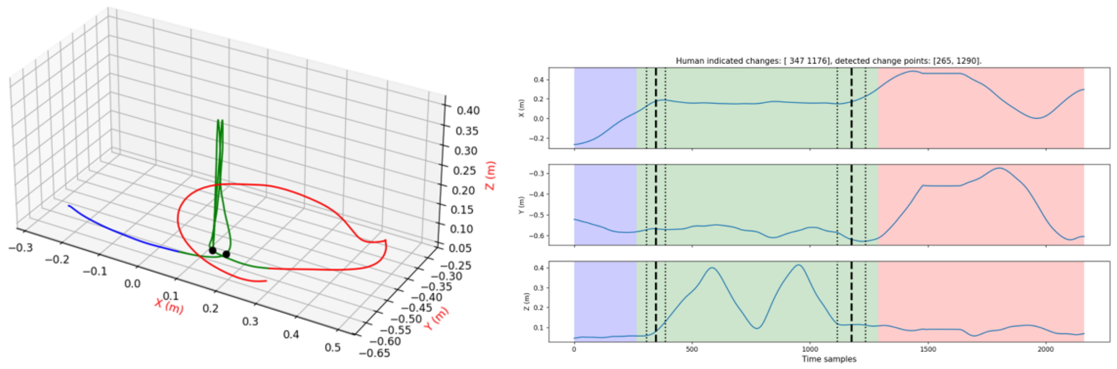
To further reduce the computational effort, the time series could be downsampled even more. To check how this influences the change point detection, the PELT and binary segmentation have been performed on the sequence with different degrees of downsampling. Table 5.6 shows the change points for the first demonstration of the sequence when different downsample factors are used. Note that the hyperparameters, min_size, and the penalty for PELT, have been adjusted accordingly. Comparing the change point's time samples does not work in this case since the time series do not have the same amount of time samples after downsampling. Therefore, the location of the change points has been expressed as a percentage of the timeline to be able to compare these for the different downsample factors. It can be noticed that downsampling the time series barely influences the location of the detected change points. Similar results were found for the other four demonstrations.

Table 5.6: Detected change points using PELT and binary segmentation for different downsampling factors. The detected change points are reported in sample numbers in the time series. The location of the change points has also been expressed as a percentage along the timeline of the time series. Demonstration 1 has been used for all scenarios in this table.

| | Detected change points | Percentage (%) |
|---|---|---|
| PELT (penalty = 100, min_size = 200, downsampled x 10) | 264, 1320, 1782 | 12.21, 61.03, 82.39 |
| PELT (penalty = 20, min_size = 40, downsampled x 50) | 53, 264, 357 | 12.24, 60.97, 82.45 |
| PELT (penalty = 10, min_size = 20, downsampled x 100) | 27, 132, 179 | 12.44, 60.83, 82.49 |
| Binary segmentation (n_changepoints = 2, min_size = 200, downsampled x 10) | 265, 1290 | 12.25, 59.64 |
| Binary segmentation (n_changepoints = 2, min_size = 40, downsampled x 50) | 55, 260 | 12.70, 60.04 |
| Binary segmentation (n_changepoints = 2, min_size = 20) | 25, 130 | 11.52, 59.91 |

### 5.2.3 User vs Algorithm Segmentation

Each sequence was annotated by human annotators to see how people would intuitively split up sequences of motions. This human annotation is compared to the outputs of the segmentation algorithms. A total of 14 individuals have been asked to manually segment the sequence. For each segmented demonstration, change points indicated by the different users have been grouped.



(a) (b)

**Figure 5.5:** Human change point detection vs binary segmentation (n_changepoints is 2). The segments found by the algorithm are indicated by the colored segments. The black dots and black dashed lines indicate the average change points indicated by people. The black dotted lines in (b) indicated the standard deviation of the human change points.

Change points within a range of 120 time samples are considered to belong to the same cluster. For each cluster consisting of at least 8 points, the mean time sample has been calculated. This results in a list of average change points for each segmented demonstration, as shown in Table 5.7. One of the segmented demonstrations is shown in Figure 5.5.

Table 5.7: Average detected change points using human segmentation. The detected change points are reported in sample numbers in the time series.

| | | Detected change points |
|---|---|---|
| Human | Demonstration 1 | 347, 1176 |
| | Demonstration 2 | 379,1279 |
| | Demonstration 3 | 415, 1321 |
| | Demonstration 4 | 374, 1465 |
| | Demonstration 5 | 452, 1374 |

The segmentation of a sequence can highly influence the classification of individual skills. Therefore, the classification accuracy is analyzed for the different segmentations performed by people, the PELT method, and the binary segmentation method. All segments resulting from the segmentation of the three different methods have been classified using Detach-ROCKET with 10,000 MINIROCKET kernels. The training set, representing the skill library, contained all motions stated in Table 5.1. The classification accuracy of each method is stated in Table 5.8, as well as the average deviation from the change points detected by humans expressed in time points.

**Table 5.8:** Classification accuracy for segments of the sequence. The sequence has been segmented using different segmentation methods.

| Segmentation method | Classification accuracy | Average deviation from human segmentation |
|---|---|---|
| Human segmentation | 100% | - |
| PELT | 85% | 121.2 |
| Binary segmentation | 84% | 121.1 |

## 5.3 Novelty Detection Evaluation

To assess novelty detection using the LOF, the algorithm's performance in identifying previously unseen skills transformed by MINIROCKET kernels is evaluated. The recorded dataset is split up into a training set and a test set. The training set represents the skill library. The test set represents recorded motions which will be labeled as known or new by the novelty detector. The datasets were preprocessed in the same way as for the classification evaluation. When a recording of a motion is in both the training set and the test set, it should be predicted as known, meaning that a similar motion has been seen before, i.e. it is in the skill library. The novelty detection performance has been also tested for different combinations of the motion categories stated in Table 5.1.

The performance is expressed in terms of precision, recall, and F1-score. The precision is a measure of how many detected novel motions are actually novel. Recall says how many actual novel points were correctly detected. The F1-score combines the precision and recall. These measures can be calculated as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} \times \text{FP}}, \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.1}$$

with TP, FP, and FN being the true positives, the false positives, and the false negatives respectively. An overview of the precision, recall, and F1-score using different training sets is shown in Table 5.9.

When all motions are in both the training set and the test set, everything is predicted to be known, as expected. In order to test if new motions can be detected as well, part of the samples in the training data is removed. The corresponding motions in the test set should then be detected as 'new' by the algorithm. When all motions in the category 'lines' are removed from the training set, they are detected as new motions. In a similar fashion, when the up-down motions are left out of the novelty detection, they are detected as novelties. Hence, the novelty detection works correctly for the lines and up-down motions.

When leaving the three circle-shaped motions out of the training set, only one of these motions was detected as a novelty. This happens because the curves of the circles and the semi-circle show much resemblance to the curves of the S-shape. When leaving the circle-shapes in the training set and taking out the S,Z-shapes, similar conclusions can be drawn. The S-shape is not predicted as a new motion due to its resembling characteristic to the circles. However, the mirrored Z does not have these curvy characteristics, so it is detected as a novelty, as expected.

Similarly, when only one or a few motions of the same category are removed from the training set, the novelty detector will predict the unknown motions as known. For example, when leaving the line in $x$-direction out of the training set, it will not be detected as new. It shows too much similarity to a line in $y$-direction, especially after all motions have been spatially aligned using SVD. For this reason, entire categories have been left out of the training set during the novelty detection evaluation.

**Table 5.9:** Precision, recall, and F1-score for novelty detection using Local Outlier Factor (LOF). For each trial, a specific motion category was removed from the training set, and the performance metrics were computed based on the ability to detect these removed categories as novel in the test set. The test set includes all motion categories listed in Table 5.1, unless noted otherwise.

| Removed from training set | Precision | Recall | F1-score |
|---|---|---|---|
| - | 1.00 | 1.00 | 1.00 |
| Lines | 1.00 | 1.00 | 1.00 |
| Circle-shaped | 1.00 | 0.33 | 0.50 |
| S-Z shapes | 1.00 | 0.50 | 0.67 |
| Up-down-up-down shapes | 1.00 | 1.00 | 1.00 |

Chapter 6

# Discussion

## 6.1 General Discussion

This project aimed to develop a framework that can segment sequences of skills, detect if the skills are already present in a skill library and, if so, recognize the skills correctly. Furthermore, the user is kept in the loop by being able to confirm or modify decisions made by the algorithm.

Detach-ROCKET with MINIROCKET kernels and pooling operators was selected as the most suitable TSC method. Compared to ROCKET and MULTIROCKET, MINIROCKET demonstrated significantly faster processing times while maintaining high performance. Consequently, all further evaluations were conducted using MINIROCKET.

The few demonstrations used to fit the classifier were augmented by adding drift and time warping to create more training samples and to account for imperfections in demonstrations performed by non-experts during classification. The classification evaluation showed that Detach-ROCKET performed well in classifying skills based on position data of the end-effector of the Franka Research 3. However, it is required that the number of time instances does not differ too much between the data to be classified, the data used to generate the kernels, and the data used to fit the classifier. Manipulation of the data by means of downsampling and interpolation should be done with care because excessive downsampling of the time series data changes temporal relations and results in information loss. Motions with abrupt changes, like the Z-shapes, will lose their characteristics when downsampled excessively.

Segmenting multiple demonstrations of the same sequence showed consistent behavior for the PELT method and binary segmentation. Changing the penalty parameter from the PELT method influences how many change points are detected in the sequence. The binary segmentation method requires to know the number of segments in a sequence beforehand. This requirement means that prior knowledge or an additional step needs to be added to the framework before applying binary segmentation. If the estimated number of segments is incorrect, it may lead to over- or under-segmentation, affecting the accuracy of subsequent classification and novelty detection.

Both PELT and binary segmentation showed comparable classification performance, achieving accuracies of 85% and 84%, respectively. The human segmentation yielded the highest classification performance, as these segments closely reflect the skills in the skill library. Changing the downsampling factor, and hence the number of time instances in the time series, had no significant impact on segmentation outcomes. Change points are detected at similar locations along the timeline of the time series for different downsampling factors.

Motions that exhibit partial similarities in movements can be challenging to distinguish as separate skills. This is because ROCKET aims to identify patterns in the data and uses these to differentiate motions during classification by convolving random kernels with time series and extracting features. As a result, motions with similar patterns, such as curves, tend to end up close together in feature space.

The ROCKET features used for classification are also used for novelty detection. To the best of the author's knowledge, the use of ROCKET features for novelty detection with LOF has not been previously explored. The challenge of distinguishing similar motion patterns also arises in this context: previously unseen motions were not recognized as novelties if a resembling motion was already present in the skill library. However, for clearly distinct motions, such as the lines and up-down motions, the novelty detection performed well.

Unsupervised segmentation remains challenging for complex tasks, as models struggle to infer patterns relevant for humans without explicit input [1], [42]. Various studies, such as [43], [44], and [45], aim to perform segmentation fully unsupervised. While this approach is useful in scenarios where humans play no role, it introduces unnecessary complexity in situations where humans interact with robots. Therefore, the proposed framework incorporates a user confirmation step, allowing human feedback to refine segmentation and classification, thereby enhancing robustness in real-world scenarios. By allowing users to manually adjust segmentation or reject novelty detection predictions, the confirmation step mitigates challenges associated with these parts of the framework.

The modular design of the framework enables easy replacement or adjustment of individual components. The modularity allows for adaptations or extensions of the framework. Various methods could be investigated for segmentation, feature extraction, classification, and novelty detection. For instance, integrating a segmentation model that updates after user confirmation could improve the change point detection algorithm over time. Similarly, updating the novelty detection model after the confirmation step could promote continuous learning. Furthermore, alternative feature extraction techniques may improve classification and novelty detection for specific skills and motions.

Another relevant topic to investigate is the scalability of the skill library. New motions can easily be added to allow continuous learning. However, as the skill library grows, it is important to manage memory and classification efficiently. A key question would be how to manage memory efficiently while maintaining classification efficiency. Potential solutions include periodic pruning of skills or organizing them hierarchically to optimize storage and retrieval.

## 6.2 Directional and orientation invariance

Section 4.2.3 shortly discussed the challenge regarding directional and orientational invariance. One could argue that the importance of the orientation and direction of motions is task-specific. In some situations, the orientation/direction of motions does not have any meaning while in another context, it might have. For example, consider the action of stirring in a pan. It does not matter whether the motion is clockwise or counterclockwise. On the other hand, rotating a key in a lock can either lock or unlock a door depending on the direction in which the key is turned. Moving clockwise or counter-clockwise has a distinctively different meaning in this case. Therefore, the significance of invariance is highly context-dependent, as the relevance of motion orientation and direction varies based on the specific task.

In the novelty detection evaluation, it was observed that motions with highly similar movement patterns are not always recognized as ”new”. Instead, the algorithm may classify them as previously seen motions, even if they have not been encountered before, especially when SVD is used to align them. Whether it is desirable to detect a line in one direction as a similar motion as a line in another direction, depends a lot on the context of the motions. Also for motions like circles and curves, the starting position of the motion and the movement direction greatly impact where the motion will end up in feature space.

The relevance of orientation/direction could also depend on preceding or subsequent actions. The importance of orientation/direction could be determined by pre- or post-conditions of a certain action. In these cases, the importance of orientation/direction is not intrinsic to the single motion but arises from the larger sequence or context, with pre- or post-actions determining its significance.

The extent to which similar motions performed in different directions are considered distinct also impacts motion generalization. Allowing for directional or orientation invariance can reduce the number of unique motion classes by grouping similar motions. On the other hand, too much invariance can lead to over-generalization, where motions with subtle but meaningful directional differences are misclassified as the same action. Therefore, it is desirable to get more insights into the context in which a skill has to be performed. Based on the situation, a decision can be made about the degree of invariance needed. This prevents over-generalization in cases where the direction or orientation is relevant.

## 6.3 Recommendations for Future Work

### 6.3.1 Including more contextual information

This study relied solely on end-effector position data for motion segmentation, classification, and novelty detection. Using only position data limits skill differentiation to movement patterns. Future work could incorporate additional data streams, such as gripper data, force sensor data, end-effector orientation, and vision data to provide greater contextual awareness. Integrating additional features could enhance classification specificity by capturing other distinguishing skill characteristics than just motion patterns. This added context could also help determine the necessary degree of invariance for different tasks. However, meaningful information might be overruled by information from less important channels as the number of data channels increases.

The authors of [25] and [26] developed a way to identify the most relevant channels for classification when using an ensemble of Detach-ROCKET instances. When the number of input channels is high, more convolutional kernels are needed to find relevant patterns. However, increasing the number of kernels is computationally inefficient. Instead, an ensemble model can be used which combines several standard instances of Detach-ROCKET. After the pruning step, the model retrieves the data channels used by the kernels that have not been eliminated. This information is used to give an estimation of the relevance of each channel.

Additionally, during this project, code was developed to analyze the proportion of MAX values and PPV values in the remaining features after pruning. This provides insights into the relevance of specific pooling operators when classifying certain data. These findings could be used to design new pooling operators for feature extraction. This 'feature shaping' could contribute to versions of ROCKET that are more specialized for specific data types.

### 6.3.2 Using textual input instead of physical demonstrations

Instead of letting a non-expert user physically demonstrate the skill or task that needs to be executed, the user could give instructions via prompts that will serve as input for a large language model. The language model can select skills from the skill library that are suitable for the execution of the explained skill or task. Similar work has been investigated in [46]. The segmentation in the current framework will then be replaced by task interpretation. Depending on the interpretation, suitable skills can be chosen from the skill library.

Chapter 7

# Conclusion

This project developed a framework for task segmentation, classification, and novelty detection that allows non-expert users to leverage a skill library containing motions recorded by experts. By integrating user feedback, the framework ensures that segmentation and classification remain adaptable and interpretable, addressing challenges associated with unsupervised motion analysis.

The evaluation demonstrated that Detach-ROCKET, particularly its MINIROCKET variant, provides an efficient and accurate method for classifying motion sequences. Despite its strong performance, the difference in sample rate between training and test data should be minimized for the best results. Some degree of downsampling is desired because it significantly decreases computational load and increases time efficiency. However, excessive downsampling results in too much information loss to still obtain accurate performance in classification and novelty detection. Segmentation methods, including PELT and binary segmentation, exhibited consistent performance, but user-defined segmentation yielded the highest classification accuracy. Therefore, the user-in-the-loop approach proved valuable in refining results, particularly in mitigating errors in segmentation and novelty detection.

A key challenge identified was distinguishing similar motion patterns, particularly when movements shared overlapping features in the feature space, which is the case for i.e. circles and semi-circles. The extent to which directional and orientation invariance should be applied depends on task-specific requirements, as some motions may be interchangeable while others require precise directional control. Future work could explore context-aware methods to determine when invariance is appropriate.

The framework's modular design enables easy adaptation, allowing for improvements in segmentation, feature extraction, classification, and novelty detection. Enhancing novelty detection through adaptive learning could improve the system's ability to recognize previously unseen skills. Additionally, incorporating contextual data such as force, vision, and gripper data could further refine classification performance since in this research, only end-effector position data was used.

Finally, the scalability of the skill library presents an important avenue for future research. As more motions are added, efficient memory management and hierarchical organization will be necessary to maintain classification speed and accuracy.

Overall, this framework provides a strong foundation for skill-based learning, enabling non-expert users to efficiently apply expert-recorded skills in diverse applications without the need for hours of training or complex coding. It advances LfD by offering a computationally efficient, and human-guided framework for skill acquisition and task programming in robotic learning.

# Bibliography

[1] A. D. Sosa-Ceron, H. G. Gonzalez-Hernandez, and J. A. Reyes-Avendaño, "Learning from demonstrations in human–robot collaborative scenarios: A survey," *Robotics*, vol. 11, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2218-6581/11/6/126

[2] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, "Skill-based instruction of collaborative robots in industrial settings," *Robotics and Computer-Integrated Manufacturing*, vol. 53, pp. 72–80, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584516301910

[3] Y. S. Liang, D. Pellier, H. Fiorino, and S. Pesty, "Evaluation of a robot programming framework for non-experts using symbolic planning representations," in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 1121–1126.

[4] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. Volume 3, 2020, pp. 297–330, 2020. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-control-100819-063206

[5] A. Barekatain, H. Habibi, and H. Voos, "A practical roadmap to learning from demonstration for robotic manipulators in manufacturing," *Robotics*, vol. 13, no. 7, 2024. [Online]. Available: https://www.mdpi.com/2218-6581/13/7/100

[6] N. Gavenski, O. Rodrigues, and M. Luck, "Imitation learning: A survey of learning methods, environments and metrics," 2024. [Online]. Available: https://arxiv.org/abs/2404.19456

[7] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, 12 2019.

[8] M. Pedersen, L. Nalpantidis, R. Andersen, C. Schou, S. Bøgh, V. Krueger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, 02 2016.

[9] B. Akgun, K. Subramanian, and A. Thomaz, "Novel interaction strategies for learning from teleoperation," *AAAI Fall Symposium - Technical Report*, pp. 2–9, 01 2012.

[10] F. Steinmetz, V. Nitsch, and F. Stulp, "Intuitive task-level programming by demonstration through semantic skill recognition," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3742–3749, 2019.

[11] Franka Emika DESK. [Online]. Available: https://franka.de/interaction

[12] ArtiMinds RPS. Robot programming suite for universal programming of industrial robots. [Online]. Available: https://www.artiminds.com/robotics-software-and-services/robot-programming-suite-basic/

[13] F. Steinmetz, A. Wollschläger, and R. Weitschat, "Razer–a hri for visual task-level programming and intuitive skill parametrization," *IEEE Robotics and Automation Letters*, vol. 3, 2018.

[14] G. Neumann, C. Daniel, A. Paraschos, A. G. Kupcsik, and J. Peters, "Learning modular policies for robotics," *Frontiers in Computational Neuroscience*, vol. 8, 2014.

[15] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters, "Learning movement primitive libraries through probabilistic segmentation," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 879–894, 2017. [Online]. Available: https://doi.org/10.1177/0278364917713116

[16] T. Eiband, F. Lay, K. Nottensteiner, and D. Lee, "Unifying skill-based programming and programming by demonstration through ontologies," *Procedia Computer Science*, vol. 232, pp. 595–605, 2024, 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924000590

[17] M. Middlehurst, P. Schäfer, and A. Bagnall, "Bake off redux: a review and experimental evaluation of recent time series classification algorithms," *Data Mining and Knowledge Discovery*, vol. 38, no. 4, p. 1958–2031, Apr. 2024. [Online]. Available: http://dx.doi.org/10.1007/s10618-024-01022-1

[18] A. J. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," *CoRR*, vol. abs/1602.01711, 2016. [Online]. Available: http://arxiv.org/abs/1602.01711

[19] C. W. Tan, F. Petitjean, E. J. Keogh, and G. I. Webb, "Time series classification for varying length series," *CoRR*, vol. abs/1910.04341, 2019. [Online]. Available: http://arxiv.org/abs/1910.04341

[20] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels," *CoRR*, vol. abs/1910.13051, 2019. [Online]. Available: http://arxiv.org/abs/1910.13051

[21] A. Dempster, D. F. Schmidt, and G. I. Webb, "MINIROCKET: A very fast (almost) deterministic transform for time series classification," *CoRR*, vol. abs/2012.08791, 2020. [Online]. Available: https://arxiv.org/abs/2012.08791

[22] C. W. Tan, A. Dempster, C. Bergmeir, and G. I. Webb, "Multirocket: Effective summary statistics for convolutional outputs in time series classification," *CoRR*, vol. abs/2102.00457, 2021. [Online]. Available: https://arxiv.org/abs/2102.00457

[23] H. Salehinejad, Y. Wang, Y. Yu, T. Jin, and S. Valaee, "S-rocket: Selective random convolution kernels for time series classification," 2022. [Online]. Available: https://arxiv.org/abs/2203.03445

[24] S. Chen, W. Sun, L. Huang, X. P. Li, Q. Wang, and D. John, "Pocket: Pruning random convolution kernels for time series classification from a feature selection perspective," *Know.-Based Syst.*, vol. 300, no. C, Nov. 2024. [Online]. Available: https://doi.org/10.1016/j.knosys.2024.112253

[25] G. Uribarri, F. Barone, A. Ansuini, and E. Fransén, "Detach-rocket: Sequential feature selection for time series classification with random convolutional kernels," *Data Mining and Knowledge Discovery*, pp. 1–26, 2024.

[26] A. Solana, E. Fransén, and G. Uribarri, "Classification of raw meg/eeg data with detach-rocket ensemble: An improved rocket algorithm for multivariate time series analysis," 2024. [Online]. Available: https://arxiv.org/abs/2408.02760

[27] P. Yang, G. Dumont, and M. Ansermino, "Adaptive change detection in heart rate trend monitoring in anesthetized children," *IEEE transactions on bio-medical engineering*, vol. 53, pp. 2211–9, 12 2006.

[28] J. Verbesselt, R. Hyndman, G. Newnham, and D. Culvenor, "Detecting trend and seasonal changes in satellite image time series," *Remote Sensing of Environment*, vol. 114, no. 1, pp. 106–115, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003442570900265X

[29] I. Naoki and J. Kurths, "Change-point detection of climate time series by nonparametric method," *Lecture Notes in Engineering and Computer Science*, vol. 2186, 10 2010.

[30] L. Oudre, R. Barrois-Müller, T. Moreau, C. Truong, A. Vienne-Jumeau, D. Ricard, N. Vayatis, and P.-P. Vidal, "Template-based step detection with inertial measurement units," *Sensors*, vol. 18, no. 11, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/11/4033

[31] C. Truong, L. Oudre, and N. Vayatis, "A review of change point detection methods," *CoRR*, vol. abs/1801.00718, 2018. [Online]. Available: http://arxiv.org/abs/1801.00718

[32] ——, “ruptures: change point detection in python,” 2018. [Online]. Available: https://arxiv.org/abs/1801.00826

[33] R. Killick, P. Fearnhead, and I. A. Eckley, “Optimal detection of changepoints with a linear computational cost,” *Journal of the American Statistical Association*, vol. 107, no. 500, p. 1590–1598, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1080/01621459.2012.737745

[34] P. Fryzlewicz, “Wild binary segmentation for multiple change-point detection,” *The Annals of Statistics*, vol. 42, no. 6, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1214/14-AOS1245

[35] scikit-learn developers, *Novelty and Outlier Detection*, Scikit-learn, 2024. [Online]. Available: https://scikit-learn.org/1.5/modules/outlier_detection.html#novelty-and-outlier-detection

[36] E. R. de Faria, I. R. Goncalves, A. C. P. de Leon Ferreira de Carvalho, and J. Gama, “Novelty detection in data streams,” *Artificial Intelligence Review*, vol. 45, pp. 235 – 269, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:17012701

[37] J.-G. Gaudreault and P. Branco, “A systematic literature review of novelty detection in data streams: Challenges and opportunities,” *ACM Comput. Surv.*, vol. 56, no. 10, May 2024. [Online]. Available: https://doi.org/10.1145/3657286

[38] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” *SIGMOD Rec.*, vol. 29, no. 2, p. 93–104, May 2000. [Online]. Available: https://doi.org/10.1145/335191.335388

[39] J. Hong, Z. Zhang, A. M. S. Enayati, and H. Najjaran, “Human-robot skill transfer with enhanced compliance via dynamic movement primitives,” 2023. [Online]. Available: https://arxiv.org/abs/2304.05703

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, and G. Louppe, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, 01 2012.

[41] D. Gachomo, “The power of the pruned exact linear time(pelt) test in multiple changepoint detection,” *American Journal of Theoretical and Applied Statistics*, vol. 4, p. 581, 12 2015.

[42] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg, *Transition State Clustering: Unsupervised Surgical Trajectory Segmentation for Robot Learning*. Cham: Springer International Publishing, 2018, pp. 91–110. [Online]. Available: https://doi.org/10.1007/978-3-319-60916-4_6

[43] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters, “Learning movement primitive libraries through probabilistic segmentation,” *Int. J. Rob. Res.*, vol. 36, no. 8, p. 879–894, Jul. 2017. [Online]. Available: https://doi.org/10.1177/0278364917713116

[44] T. Shankar, S. Tulsiani, L. Pinto, and A. Gupta, “Discovering motor programs by recomposing demonstrations,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rkgHY0NYwr

[45] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia, “Compile: Compositional imitation learning and execution,” 2019. [Online]. Available: https://arxiv.org/abs/1812.01483

[46] N. Ingelhag, J. Munkeby, J. van Haastregt, A. Varava, M. C. Welle, and D. Kragic, “A robotic skill learning system built upon diffusion policies and foundation models,” 2024. [Online]. Available: https://arxiv.org/abs/2403.16730

# Appendix A

# Usage of AI tools

Conforming the guidelines of the University of Twente concerning ”Use of AI in Education at the University of Twente”, the following statement can be made:

During the preparation of this work, the author used the following tools for the following reasons.

- **ChatGPT**: ChatGPT has been used for debugging purposes and served as source of inspiration to help improve the academic tone of certain sentences.

After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

# Appendix B

# ROCKET Details

## B.1 Kernel characteristics

The kernel specifications of the ROCKET algorithm and its variants are stated in Table B.1.

**Table B.1:** Kernel specifications for ROCKET, MINIROCKET, and MULTIROCKET.

| | ROCKET | MINIROCKET | MULTIROCKET |
|---|---|---|---|
| Kernel length $l_k$ | 7, 9 or 11 | 9 | 9 |
| Weights $W$ | Normal distribution $\mathcal{N}(0,1)$ | {-1,2} | {-1,2} |
| Bias $b$ | Uniform distribution $\mathcal{U}(-1,1)$ | From convolution input | From convolution input |
| Dilation $d$ | Random | Fixed (relative to input length) | Fixed (relative to input length) |
| Padding | Random decision whether or not padding is used | Fixed | Fixed |
| Pooling operators | MAX<br>PPV | PPV | PPV<br>MPV<br>MIPV<br>LSPV |

The dilation $d$ can be defined as $d = [2^{U_d}], \quad U_d \sim \mathcal{U}\left(0, \frac{\log_2(l_{\text{input}}-1)}{\log_2(l_k-1)}\right)$ for the standard ROCKET algorithm, where $l_{\text{input}}$ is the length of the input time series. For MINIROCKET and MULTI-ROCKET, the dilation $d$ is sampled from set $D = [2^0, ..., 2^A]$, where the exponents are uniformly spaced between 0 and $A = \frac{\log_2(l_{\text{input}}-1)}{\log_2(l_k-1)}$.

## B.2 ROCKET Transform

ROCKET transforms a time series into a feature space using a large number of kernels. The transformation happens by convolving time series $X$ with kernel $k$ which has length $l_k$, weights $[w_1, w_2, ..., w_l]$, bias $b$, and dilation factor $d$. Each element of convolution output $Z$ can be written as:

$$z_i = \sum_{j=0}^{l_k-1} w_j \cdot x_{i+j\cdot d} + b \tag{B.1}$$

For multivariate time series, the output is the sum of the convolutions of the kernel with random subsets of the input channels. The number of selected channels $C_k$ for each kernel $k$ is determined probabilistically by:

$$C_k = 2^{U_k}, \quad U_k \sim \mathcal{U}\left(0, \log_2(\min(n_{\text{channels}}, l_k) + 1)\right) \tag{B.2}$$

where:

- $U_k$ is sampled from a uniform distribution.
- The exponentiation ensures that $C_k$ follows an exponential distribution, favoring smaller values.
- The upper bound $\min(n_{\text{channels}}, l_k) + 1$ ensures $C_k$ does not exceed the number of available channels or kernel length.

After the transform, features will be extracted from the convolution output $Z$. There are several pooling operators that are used to extract the features.

## B.3 Pooling Operators

The definitions of the pooling operators used in the various ROCKET algorithms are given below.

**Proportion of positive values:**

$$\mathrm{PPV}(Z) = \frac{1}{n}\sum_{i=1}^{n}[z_i > 0] \tag{B.3}$$

in which $Z$ is the convolution output with length $n$.

**Global max pooling**

$$\mathrm{MAX}(Z) = max(Z) \tag{B.4}$$

**Mean of positive values:**

$$\mathrm{MPV}(Z) = \frac{1}{m}\sum_{i=1}^{m} z_i^+ \tag{B.5}$$

in which $Z^+$ is the vector with length $m$ containing the positive values of $Z$.

**Mean of indices of positive values:**

$$\mathrm{MIPV}(Z) = \begin{cases} \frac{1}{m}\sum_{j=1}^{m} i_j^+ & \text{if } m > 0, \\ -1 & \text{otherwise.} \end{cases} \tag{B.6}$$

in which $I^+$ represents the vector with the indices of all positive values of $Z$.

**Longest stretch of positive values:**

$$\mathrm{LSPV}(Z) = \max[j - i]\forall_{i \leq k \leq j z_k} > 0] \tag{B.7}$$

Depending on which ROCKET version is used, the features extracted from all convolution outputs are placed in a vector. The set of all extracted features is then denoted as $\mathcal{F}$.

## B.4 Sequential Feature Detachment (SFD)

Detach-ROCKET uses the SFD algorithm to perform feature selection. SFD is an iterative process. At each detachment step, a fixed percentage of features is removed from the set of active features $\mathcal{S}$, where $\mathcal{S} \subseteq \mathcal{F}$. Parameter $p$ is the proportion of eliminated features at each step. The default value for $p$ is 0.05.

At each iteration $t$, a ridge regression classifier is applied to the current set of selected features, denoted as $\mathcal{S}_t$. This involves solving the following optimization problem:

$$\hat{\theta}_t^{\text{ridge}} = \arg\min_{\theta} \left\{ \sum_{i=1}^{N} \left( y_i - \theta_0 - \sum_{k \in \mathcal{S}_t} x_{ik}\theta_k \right)^2 + \lambda \sum_{k \in \mathcal{S}_t} \theta_k^2 \right\} \tag{B.8}$$

The resulting coefficients $\hat{\theta}_t^{\text{ridge}} = \{\hat{\theta}_k\}$ indicate the contribution of each feature to the classifier's decision. Features are then ranked based on the absolute values of these coefficients, $|\hat{\theta}_k|$, with the least significant $100 \cdot p$ percent being eliminated. The remaining $100 \cdot (1 - p)$ percent features are preserved for the next step, forming $\mathcal{S}_{t+1}$. The algorithm is explained step by step in Algorithm 1.

**Algorithm 1** Sequential Feature Detachment

```
Require: Number of iterations $M$, initial number of features $N$, number of kernels $K$, elimination fraction $p$
 1: Initialize: Generate feature set using ROCKET with $\mathcal{F} = 2K$ features
 2: Train ridge regression model using LOOCV to determine $\lambda$
 3: for $t = 1$ to $M$ do
 4:     Train ridge classifier on active feature set $\mathcal{S}$
 5:     Compute optimal coefficients $\hat{\theta}_k$ for each feature
 6:     Rank features based on $|\hat{\theta}_k|$
 7:     Eliminate the lowest $p$ fraction of ranked features
 8:     Update $\mathcal{S}$ with the remaining features
 9: end for
10: Return Selected feature subsets $\mathcal{S}_t$ at each step
```

The accuracy of the model can be computed and plotted for each $\mathcal{S}_t$, as shown in Figure B.1. The optimal number of retained features $Q_c$ is determined by the following optimization problem:

$$Q_c = \arg\max_q \{\alpha(q) + c \cdot q\} \tag{B.9}$$

$q$ represents the proportion of eliminated features and $\alpha(q)$ is the model accuracy for a certain number of eliminated features. $c$ influences the trade-off between accuracy and number of features.

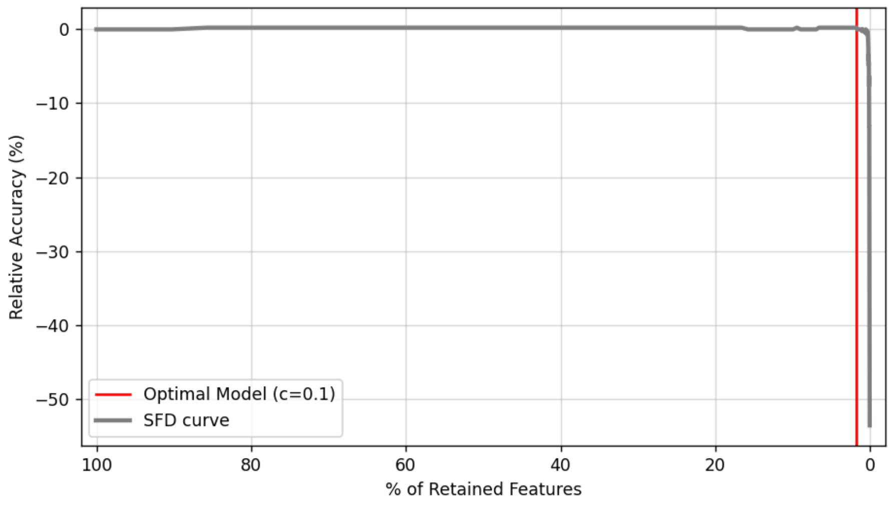


**Figure B.1:** Example of a SFD curve; The model accuracy relative to the accuracy of the initial model is plotted with respect to the proportion of retained features.

When $Q_c$ is known, the ridge classifier is retrained on the training data using the selected set of features. The reduced model can then also be used to classify the test data.

# Appendix C

# Recorded demonstrations

All recorded skills are visualized below. The red dot marks the start of a motion.

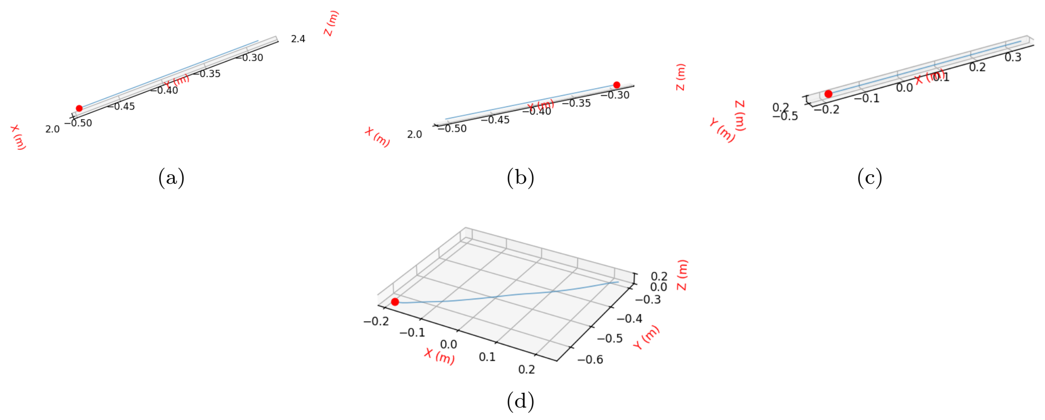


**Figure C.1:** Demonstrations of line motions: (a) Line +y (b) Line -y (c) Line +x (d) Diagonal line

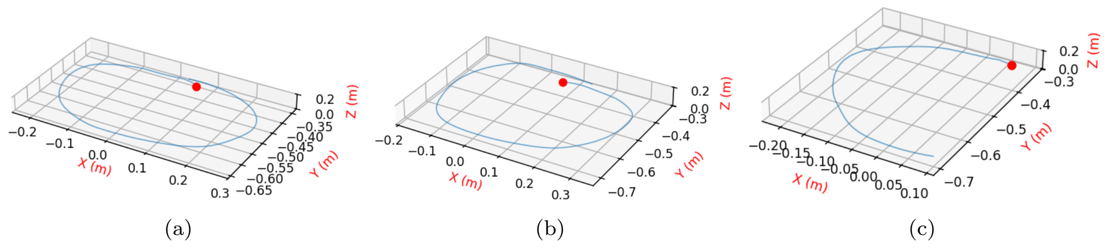


**Figure C.2:** Demonstrations of circle-shapes: (a) CCW circle (b) CW circle (c) CCW Semi-circle

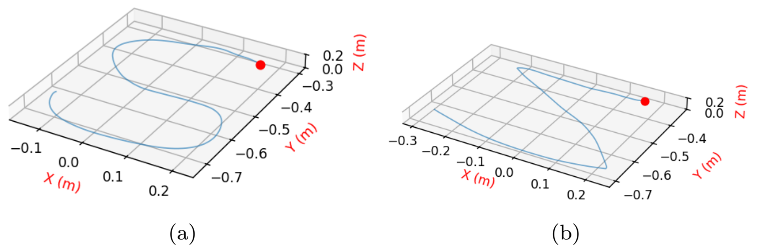
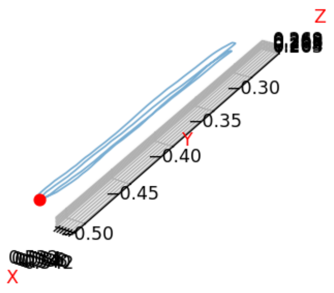
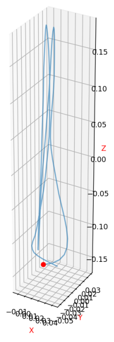


**Figure C.3:** Demonstrations of S- and Z-shapes: (a) S-shape (b) Mirrored Z-shape



(a)



(b)

**Figure C.4:** Demonstrations of up-down-up-down motions: (a) Up-down-up-down y-direction (b) Up-down-up-down z-direction