

Detection of Unexpected Human Behavior in Human-Robot Interaction in Shared Workspaces

Erkennung von unerwartetem menschlichen Verhalten bei der Mensch-Roboter Interaktion in gemeinsamen Arbeitsbereichen

Master thesis by Jessica Löhr

Date of submission: September 30, 2022

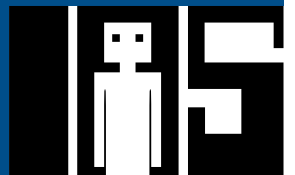
1. Review: Dorothea Koert, Ph.D.

2. Review: Xenija Neufeld, Ph.D.

Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

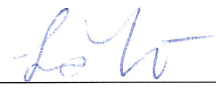
Hiermit versichere ich, Jessica Löhr, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, September 30, 2022



J. Löhr

Abstract

In the last years, the use of robots as support for example in assembly tasks has gained increasing popularity. Unlike traditional industrial robots which worked in a separate environment, such collaborative robots, also called cobots, often work simultaneously with the human worker in a shared workspace. That is why cobots have to fulfill high safety requirements and especially have to implement mechanisms to avoid collisions with the human worker. This is usually done by replanning the trajectory around the human worker or stopping the robot. However, there are applications where collisions mainly appear if the user shows unexpected behavior. Then it can be sufficient to first iteratively decrease the velocity of the robot for the duration of such anomalies and then increase the velocity again. This approach is more efficient than an immediate stop without having to replan the whole trajectory and might also minimize the stress level of the human worker during the interaction with the robot. This thesis aims to develop such an anomaly detection algorithm for a virtual reality training with a limited amount of training data. The proposed method consists of two levels: the first one is based on continuous action recognition and should recognize if the user fulfills the correct task, and the second one utilizes the conformal anomaly detection framework which should detect if the human behavior is anomalous compared to the training data for that task. We were able to apply Gaussian mixture models to the action recognition with an accuracy of 67.72%, which is 19 percentage points weaker than in the reference paper. So the proposed methods for action recognition should still be improved before they can be used in real applications. Our method for anomaly detection can work with a low false alarm rate while still detecting anomalies sufficiently.

Zusammenfassung

In den letzten Jahren hat der Einsatz von Robotern zur Unterstützung z. B. bei Montageaufgaben zunehmend an Popularität gewonnen. Im Gegensatz zu traditionellen Industrierobotern, die in einer separaten Umgebung arbeiten, arbeiten solche kollaborativen Roboter, auch Cobots genannt, oft gleichzeitig mit dem menschlichen Arbeiter in einem gemeinsamen Arbeitsbereich. Aus diesem Grund müssen Cobots hohe Sicherheitsanforderungen erfüllen und insbesondere Mechanismen zur Vermeidung von Kollisionen mit dem Menschlichen. Dies geschieht in der Regel durch Umplanung der Trajektorie um den Menschen oder das Anhalten des Roboters. Es gibt jedoch Anwendungen, bei denen Kollisionen vor allem dann auftreten, wenn der Benutzer ein unerwartetes Verhalten zeigt. In solchen Fällen kann es ausreichen, zunächst die Geschwindigkeit des Roboters für die Dauer der Anomalie iterativ zu verringern und sie dann wieder zu erhöhen. Dieser Ansatz ist effizienter als ein sofortiger Stopp, ohne dass eine neue Trajektorie berechnet werden muss, und kann auch den Stresspegel des Menschen während der Arbeit mit dem Roboter verringern. Ziel dieser Arbeit ist die Entwicklung eines solchen Algorithmus zur Erkennung von Anomalien für eine Virtual-Reality-Schulung, der mit einer begrenzten Menge an Trainingsdaten funktioniert. Die vorgestellte Methode besteht aus zwei Stufen: Die erste basiert auf einer kontinuierlichen Handlungserkennung und soll erkennen, ob der Benutzer die richtige Aufgabe ausführt, und die zweite nutzt das Conformal Anomaly Detection Framework, um zu erkennen, ob das menschliche Verhalten im Vergleich zu den Trainingsdaten für diese Aufgabe anomal ist. Wir konnten mit Gaußsche Mischungsmodellen eine Genauigkeit von 67.72%, was 19 Prozentpunkte unter dem Referenzwert liegt. Die Methode zur kontinuierlichen Handlungserkennung sollte also noch verbessert werden, bevor sie in der echten Anwendung benutzt werden kann. Unsere Methode zur Anomalieerkennung ist in der Lage, mit einer niedrigen Rate an Falschalarmen zu arbeiten, während immer noch genug Anomalien erkannt werden.

Contents

1. Introduction	2
1.1. Motivation	2
1.2. Problem Statement	3
1.3. Content Structure	3
2. Foundations and Related Work	5
2.1. Working with Collaborative Robots	5
2.2. Continuous Human Action Recognition	6
2.3. Anomaly Detection	8
2.4. Related Work	10
3. Approach	13
3.1. Overview	13
3.2. Detection of Unexpected Human Behaviour	14
4. Setup and Evaluation	18
4.1. Setup	18
4.2. Detecting and Handling Unexpected Human Behaviour	26
4.3. Evaluation	28
5. Discussion and Outlook	51
5.1. Framework Performance Discussion	51
5.2. Limitations	51
5.3. Outlook	52
5.4. Conclusion	53
A. Some Appendix	57

Figures and Tables

List of Figures

1.1. Illustration of interaction levels	3
1.2. Sawyer cobot	4
3.1. Anomaly detection framework	16
4.1. Structure diagram	19
4.2. Joint hierarchy for kinect body tracking	20
4.3. Training steps	22
4.4. Simulation screenshot	23
4.5. Hardware components	24
4.6. Azure Kinect	25
4.7. Recording screenshot	26
4.8. Timeline	30
4.9. Durations of actions	31
4.10. Trajectories for the “move gripper station” action for all relevant joints and dimensions . .	32
4.11. Trajectories for the “move gripper station” action for all relevant joints and dimensions with DTW	33
4.12. Confusion matrix for segmented action recognition	35
4.13. Accuracy for different number of components and absolute features	36
4.14. Accuracy for different number of components and relative features	36
4.15. Segmented action recognition with different window sizes	37
4.16. Example: Likelihoods and predictions for one recording with different window sizes and frequency domain features	38
4.17. Segmented- vs. continuous action recognition	39

4.18.Example: Likelihoods and predictions for one recording with different window sizes and time domain features	40
4.19.Frequency domain features vs. time domain features	41
4.20.Anomaly example 1	43
4.21.Anomaly example 2	44
4.22.Anomaly example 3	45
4.23.Anomaly example 4	46
4.24.Anomaly example 5	47
4.25.False positive/ false negative rate for likelihoods with window size 30	48
4.26.False positive/ false negative rate for likelihoods with window size 120	49
4.27.False positive/ false negative rate for CAD	50

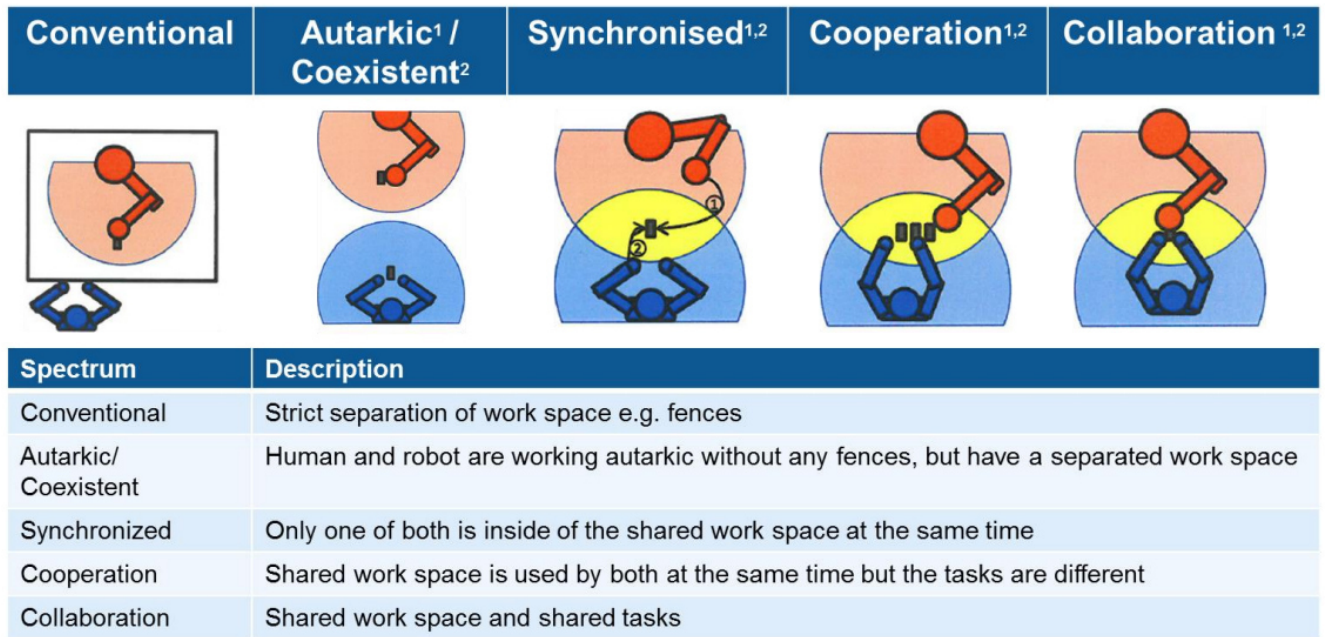
List of Tables

4.1. Segmented- vs. continuous action recognition	39
4.2. Frequency domain features vs. time domain features	41
4.3. Details about anomalies	42

1. Introduction

1.1. Motivation

In many traditional applications with industrial robots, the robot works on its task in a workspace that is separated from the human worker. Nowadays there are many other types of interactions between human workers and robots as shown in figure 1.1. Especially such applications that take place in a workspace that is shared between the human worker and the robot and that require collaboration between those gained increasing popularity over the last few years [24]. Robots that work in a collaborative setting are also called *cobots*. Such cobots can for example be used to assist in training scenarios. In such environments, the robot often must be able to hand over tools or give haptic feedback to the human. However, there are some additional aspects compared to applications with traditional robots: to allow for effective collaboration while minimizing the stress for the human worker, the robot should not only complete its task but also behave predictably and safely so that the human will neither get in dangerous situations nor be unsettled about the movements of the robot [8]. We assume that this is especially important when working in a virtual reality environment, where the human cannot see the robot and thus is not able to react to potentially dangerous movements of the robot. In the first place, collisions must be avoided for safe collaboration. For that, in earlier applications, the robots were provided with simple algorithms that stop the robot if a collision or potentially dangerous situation is detected. The problem here is that this is not a very efficient way since the robot has to wait until the threat is over before it can safely continue its movement. Nowadays, there exist many algorithms for trajectory replanning where the robot just follows an alternative route that avoids obstacles, such as STOMP [12]. However, another goal is that the human worker should not feel unsafe when working with the robot. This goal can not only be achieved by replanning the robot trajectory to avoid collisions but also by adapting the velocity of the trajectory in situations where the user is distracted and thus behaves differently than expected. In our given training scenario, collisions mainly appear in cases, where the tasks and their sequence are predefined and the actual human behavior differs from the expected behavior. Such behavior in these scenarios might arise for example if the user is not familiar with the training, if bugs arise during runtime or if a second person distracts the user during the execution of the training. On one abstraction level higher, this means that collisions only appear if the human action differs from the expected action or if the executed action is correct, but the human trajectory is still anomalous in contrast to the expected trajectory for this behavior. Here no replanning for collision avoidance is needed, but the acceptance towards the robot helper might be increased and the stress-level minimized by adapting the velocity in anomalous situations. Furthermore, such an approach could potentially be used together with replanning algorithms for collision avoidance, so that the new trajectory is executed with a slower velocity. The goal of this thesis is to develop an approach for anomaly detection in human trajectories as well as the evaluation of this approach in a given training scenario.



¹ Thiemann, Diss. Universität Tübingen, 2004
² Fraunhofer IAO, IAT University of Stuttgart

Reference: THIE04, Fraunhofer IAO

Figure 1.1.: Illustration of interaction levels¹

1.2. Problem Statement

Given a virtual reality training, the robot should monitor the motion of the human and be able to detect unexpected behavior. This means that there is a sequence of predefined tasks given, which are located in a shared workspace in which a user and a robot have to solve a task together. One important requirement is that this problem needs to be solved in real time so that the robot can adapt its speed to avoid collisions with the user. Furthermore, the robot should return to its original speed as soon as the human behaves as expected. This approach will then be evaluated in a training scenario, where a gripper station should be maintained. In this case, the task of the robot is to increase the immersion and effectiveness of the training by simulating haptic feedback for the screws and pins of the gripper station. For this, we use a Sawyer cobot², which uses a robot arm with seven degrees of freedom (figure 1.2).

1.3. Content Structure

In chapter two, the necessary foundations and important related work are introduced. Chapter three is about the approach for solving this problem and first gives a brief overview about the approach before

¹<https://ec.europa.eu/research/participants/documents/downloadPublic/K2dwcTZML0NVQVdmTzBUaGpNajRPQkNvWm1YcHlzNGNtZlV2cnArOEhSSH2NWcxSFkzRXhnPT0=/attachment/VFEyQTQ4M3ptUWR3MTVyME1TQzAwbmh4WG00dmxJM3k=>

²<https://www.rethinkrobotics.com/de/sawyer>

³<https://www.rethinkrobotics.com/de/sawyer>

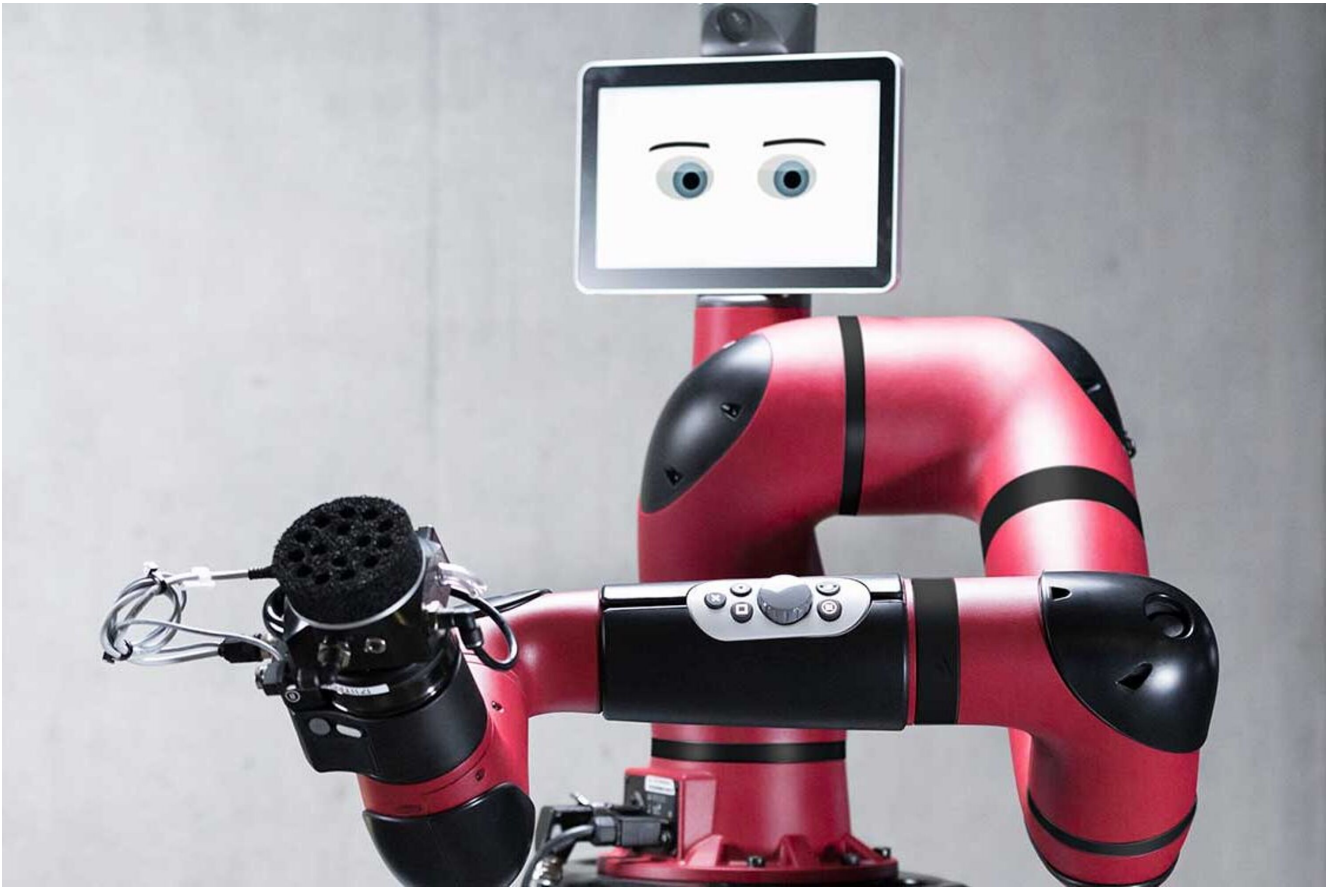


Figure 1.2.: Sawyer cobot³

diving deeper into the single components. Chapter four deals with the experimental setup and evaluation of the approach and its sub-problems in the simulation and the real world. In chapter five, the results are discussed and ideas for possible future work are given.

2. Foundations and Related Work

The approach we are going to investigate in this thesis combines two components: recognition of human action and anomaly detection. For the first topic, we are specifically interested in methods for continuous human action recognition and its main challenges, since we need to predict human actions using the continuous input of the camera. However, since our goal is to detect anomalies we will discuss some basics of anomaly detection after that, followed by methods for anomaly detection specifically for trajectories.

2.1. Working with Collaborative Robots

Before collaborative robots (cobots) became popular, traditional industrial robotic systems were used to perform tasks with high payload and high repeatability [24]. However, the required peripheral safety equipment and thus the increased costs together with lower flexibility made these robots less attractive for smaller enterprises [24]. Furthermore, some tasks are too complex and have too high requirements for flexibility to be solved by an industrial robot [24]. In such cases, cobots can be a suitable solution as they are much cheaper, can be used in a variety of tasks, and can also be easily reprogrammed without dedicated experts. The main difference between cobots and traditional robots used in industry is that cobots, due to their desired ability to collaborate with human beings, should be equipped with e.g. different sensors and force limits to be able to ensure performant and safe collaboration[24][25]. On the other hand, unlike traditional industrial robots, cobots and humans work in a shared space, where a shared space is defined as the intersection of the human- and the robot workspace. This also implies the need for higher safety requirements, which depend on the collaboration level between the human worker and the robot. According to literature, there are three different levels of collaboration in shared spaces: Coexistence, where the robot and the human worker do not share any goals, cooperation, where they work together towards a shared goal, and collaboration, where they work simultaneously on a shared object [1]. Collaboration comes with the highest requirements for safety, as the human worker and the robot can only maintain a small distance while working on their task. There are several different strategies to ensure safe collaboration for the specific tasks [25] According to UNI EN ISO 10218-2:2011 standard, there are four classes of safety requirements for cobots: safety-rated monitored stop, hand-guiding, speed- and separation monitoring, and power and force limiting. However, not only safety requirements should be taken into consideration when working with cobots, but also modularity [1]. This means that it should be easy to adapt a cobot and its task to a different environment.

2.2. Continuous Human Action Recognition

In continuous human action recognition, the goal is to assign action labels to a time-series, where one snapshot represents the human posture at that time. A special challenge here is that one time-series can contain multiple different actions. In our case, we want to use Gaussian mixture models to solve this task.

2.2.1. Dynamic Time Warping

In contrast to many other standard classification tasks, one main challenge in the classification of time series is that these time series often have different lengths. Thus, to improve the performance of classifiers for time series, one can use dynamic time warping as preprocessing step to align two trajectories in time [38]. Dynamic time warping is a method that was originally designed to find and compare patterns in time-dependent data [28] and was first used in speech recognition [3] and other areas such as gesture recognition, computer vision or chemical engineering [35].

Algorithm

Dynamic time warping can be used to map one trajectory to a reference trajectory so that the mapped trajectory has the same length as the reference trajectory. This is done in a way that aligns similar patterns and minimizes the distance between both trajectories. Tomasi et al. [38] also used DTW for preprocessing. For that they first defined a warping path F :

$$F = \langle [m(k), n(k)] | k = 1, \dots, K \rangle$$

which maps one sample trajectory and a reference trajectory to a common time axis of length K . The indices of both trajectories for the k -th position on the common time axis are given by the k -th element of F . The goal of the warping algorithm is to find a warping path that ensures the minimal distance between r and s :

$$\operatorname{argmin}_F D(F) = \frac{\sum_{k=1}^K d_{rs}[m(k), n(k)] w(k)}{\sum_{k=1}^K w(k)}$$

where $d_{rs}[m(k), n(k)]$ can be any dissimilarity measure between $r[n(k)]$ and $s[m(k)]$ and $w(k)$ contains weights.

2.2.2. Gaussian Mixture Models

GMMs are a common method for generative machine learning models. First, we will cover the fundamentals of GMMs and then discuss, how they can be used to predict human actions [22].

Fundamentals of GMMs

A Gaussian mixture model consists of k gaussian distributions, where each one has a mean μ_j and covariances Σ_j , which are weighted by mixing proportions π_j , which must be positive and sum up to one. Let N be a Gaussian distribution and D the number of dimensions. Then the probability for a sample x to be generated by a multivariate Gaussian distribution N is given by:

$$N(x|\mu, \Sigma) = \frac{1}{2\pi^{D/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu))$$

This is also referred to as *Gaussian density function*. Then the probability that a sample x is generated by a Gaussian mixture model is defined by:

$$p(x|\mu_1, \dots, \mu_k, \pi_1, \dots, \pi_k) = \sum_{j=1}^k \pi_j N(x|\mu_j, \Sigma_j)$$

Expectation Maximization

A popular approach for fitting the GMM parameters is the *expectation maximization* algorithm. It consists of two steps:

1. Expectation-Step: The probability r_{ic} for each datapoint x_i to belong to cluster c is calculated. This is done as follows:

$$r_{ic} = \frac{\pi_c N(x_i | \mu_c, \Sigma_c)}{\sum_{k=1}^K \pi_k N(x_i | \mu_k, \Sigma_k)} \quad (2.1)$$

2. Maximization-Step: For each cluster c , the weight m_c is calculated and π_c , μ_c and Σ_c are updated according to the following formulars:

$$m_c = \sum_i r_{ic} \quad (2.2)$$

$$\pi_c = \frac{m_c}{m} \quad (2.3)$$

$$\mu_c = \frac{1}{m_c} \sum_i r_{ic} x_i \quad (2.4)$$

$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c) \quad (2.5)$$

$$(2.6)$$

These steps are repeated until the log-likelihood given by

$$\ln p(X | \pi, \mu, \Sigma) = \sum_{i=1}^N \ln(\sum_{k=1}^K \pi_k N(x_i | \mu_k, \Sigma_k)) \quad (2.7)$$

converges.

GMMs for Action Recognition

As already done by Jim Mainprice et al. [22] and also Sylvain Calinon et al. [4], in order to apply GMMs to human action recognition in general we need to train one GMM for each action and then choose the action for which the respective GMM returns the highest log-likelihood. The GMMs are trained on the single poses of each action. However, for the classification, we want to consider not only the current pose but also the history of all previous poses. For that we sum up the log-likelihoods for each pose:

$$\ln(p(\xi|C_m)) = \sum_t^T \ln(p(\xi_t|C_m))$$

Prateek et al. [36] proposed another method to use GMMs also for continuous action recognition. They chose time domain features that are extracted from the input data within a window of one second to train and test the GMM.

2.3. Anomaly Detection

An anomaly can intuitively be defined as “patterns in data that do not conform to expected behavior”[5]. Anomaly detection is closely related to outlier detection and pattern recognition. It is often used for the detection of anomalous events, for example in intrusion detection or different scenarios in healthcare such as fall detection. In the following sections, we will first go over some foundations and the taxonomy for anomaly detection problems. For our case, we are interested in the specific problem of sequential anomaly detection in trajectories. In the second part, we discuss the CAD framework and some concrete methods to solve this problem.

2.3.1. Foundations of Anomaly Detection

Anomaly detection describes the task of detecting patterns in the dataset, that differ from the expected normal behavior. In his article, Varun Chandola [5] summarizes the main considerations and challenges of anomaly detection. Anomalies can be detected in three types of datasets: sequence data, spatial data, and graph data. For this work, we will focus on sequence data, since this includes time series data. Furthermore, there are different types of anomalies. Point anomalies refer to an individual anomalous sample point, while collective anomalies refer to a set of neighboring anomalous sample points. Both types can be globally anomalous, independent from their context, or only in a certain context. To be able to choose an appropriate anomaly detection technique, one has to be aware of if and how the training dataset is labeled. Datasets that are labeled supervised contain normal classes as well as anomaly classes. One main problem with these types of data labels is that there are usually way more samples with normal classes than anomalous classes which leads to imbalanced distributions. With labels for semi-supervised anomaly detection, these problems can be avoided since they only contain training data of the normal classes. Unsupervised datasets do not contain any separate training data at all. However, such datasets should only be used if there are way more normal instances than anomalous, otherwise, they will lead to a high false alarm rate. The output of anomaly detection can either be a label (normal or anomalous) or a score for each sample that represents how anomalous it is.

2.3.2. Conformal Prediction

In conformal prediction (CP), we are not only interested in the prediction, but also in how certain we are that the prediction equals the true label. The relevant foundations were summarized in [19]. In terms of machine learning, it is a technique where the algorithms do not return a predicted label, but a set of labels for which the confidence for that label exceeds a certain threshold. That means that each possible output label $y \in Y$ for a feature vector x is referred to as a hypothesis. Then the corresponding p -values p_y are calculated for each label. To calculate the p -value, the concept of nonconformity measures must be introduced: The NCM α can be considered as the output of a function $A(B, (X, y))$ which returns a score that measures how different an example (x, y) is from a set of examples B . An NCM can for instance be based on the k -nearest neighbors algorithm [39], random forests[7] or the output neural networks[32]. For the latter one we can use the following definition:

$$\alpha_i = \max_{j=1, \dots, c: j \neq u} o_j^i - o_u^i \quad (2.8)$$

where o_j^i is the output of the neural network for label j of the i -th example. In cases where the distributions for the different classes might overlap, the following NCM is useful[15]:

$$\alpha_i = \sum_{j=1}^k d_{ij}^+ \quad (2.9)$$

where d_{ij}^+ describes the j -th shortest distance between example z_i to the other examples within the same class. By calculating α_i for each training example (x_i, y_i) , the p -value for a new example z_{l+1} can be obtained by:

$$p_y = \frac{|\{j = 1, \dots, l+1 : \alpha_j \geq \alpha_{l+1}\}|}{l+1} \quad (2.10)$$

Given a significance level ϵ , a label y is then added to the prediction set if $p_y > \epsilon$. The prediction set Γ_{l+1}^ϵ for example z_{l+1} with significance level ϵ is then defined as follows:

$$\Gamma = y : y \in Y, p_y \geq \epsilon \quad (2.11)$$

An important assumption that CP makes is that the dataset is independent and identically distributed. Given this assumption, $1 - \epsilon$ can be interpreted as the probability that the prediction set contains the true label.

2.3.3. Conformal Anomaly Detection

Conformal anomaly detection (CAD) is built on the principles of conformal prediction [39]. Just as in CP, we need an NCM and a parameter ϵ .

$$p_i = \frac{|\{j = 1, \dots, n : \alpha_j \geq \alpha_i\}|}{n} \quad (2.12)$$

A label y is then added to the prediction set, if $p_y > \epsilon$.

Data: NCM A , anomaly threshold ϵ , training set (z_1, \dots, z_l) and test example z_{l+1}

Result: Indicator variable $Anom_{l+1}^\epsilon$

```

for  $i \leftarrow 1$  to  $l + 1$  do
  |  $\alpha_i \leftarrow A(\{z_1, \dots, z_{l+1}\} \setminus \{z_i, z_i\});$ 
 $\hat{p}_{l+1} \leftarrow \frac{|i=1, \dots, l+1: \alpha_i \geq \hat{\alpha}_{l+1}|}{l+1};$ 
if  $\hat{p}_{l+1} < \epsilon$  then
  |  $Anom_{l+1}^\epsilon \leftarrow 1;$ 
else
  |  $Anom_{l+1}^\epsilon \leftarrow 0;$ 

```

Algorithm 1: CAD [16]

However, for increasing training set size, CAD becomes computationally inefficient, since, for each incoming test example, it calculates the nonconformity score for each training example again. The inductive conformal anomaly detector (ICAD)[16] is an implementation of CAD, which utilizes principles of ICP: precomputed nonconformity scores of previous examples are used to calculate the p -value for a new example. Furthermore, the training examples are split into a training set (z_1, \dots, z_m) and a calibration set (z_{m+1}, \dots, z_l) . The nonconformity scores are then precomputed for each example in the calibration set. The nonconformity score for a new example is calculated via the training set, while the p -value is calculated using the calibration set.

Data: NCM A , anomaly threshold ϵ , training set (z_1, \dots, z_m) , pre-computed nonconformity scores $(\alpha_{m+1}, \dots, \alpha_l)$ and test example z_{l+1}

Result: Indicator variable $Anom_{l+1}^\epsilon$

```

 $\alpha_{l+1} \leftarrow A(\{z_1, \dots, z_m\}, z_{l+1});$ 
 $\hat{p}_{l+1} \leftarrow \frac{|i=m+1, \dots, l+1: \alpha_i \geq \hat{\alpha}_{l+1}|}{l-m};$ 
if  $\hat{p}_{l+1} < \epsilon$  then
  |  $Anom_{l+1}^\epsilon \leftarrow 1;$ 
else
  |  $Anom_{l+1}^\epsilon \leftarrow 0;$ 

```

Algorithm 2: ICAD [16]

2.4. Related Work

In this section, we will cover related work about human action recognition and anomaly detection. An important factor for both methods is that there are many algorithms with a strong performance for problems that can be solved offline. However, not all of them can be adapted to offline action recognition or anomaly detection. Hence, some algorithms are specifically designed for such cases. Furthermore, we will cover some works about safe human-robot collaboration.

2.4.1. Safe Human Robot Collaboration

To allow for efficient and collision-free replanning for robot motion in human-robot collaboration, many approaches are based on the prediction of human trajectories. This prediction is then used to replan the robot trajectory such that a certain cost function is optimized. This cost function is chosen in a way that the robot still follows an efficient trajectory and also avoids collisions. In many applications, the human worker performs tasks that are already known beforehand. Given this restriction, the problem can be solved by using inverse optimal control to learn a cost function out of the given demonstrations for each task [23]. One appropriate solution for this inverse optimal control problem can be obtained by *path integral inverse reinforcement learning* [11] since it can deal with data in high-dimensional and continuous state-action spaces and only assumes local optimality of the demonstrations. In an earlier work, Jim Mainprice et al. [22] obtained the cost function by first fitting a *gaussian mixture model* (GMM) [34] to a set of motion trajectories for each class and then using *gaussian mixture regression* [37] to predict a trajectory for each class. Then a swept volume is calculated for each class by moving a human model along these trajectories, which are then used in the online phase to estimate the likelihood of workspace occupancy for a voxel. This is done by first obtaining the likelihood for each class for the current partial trajectory using the GMM and then summing the likelihood for all classes that occupy that voxel. These estimations of workspace occupancy can then also be used as input for a motion planner. For the iterative replanning step, stochastic trajectory optimization for motion planning [12] is an often used algorithm, since it does not get stuck in local minima due to its stochastic property and also the analytical gradient of the cost function does not need to be known. However, a challenge for the evaluations of such algorithms is the comparison of trajectories. *Dynamic time warping* [3] is an often used algorithm for this.

2.4.2. Recognition of Human Action

The goal of human action recognition is to predict an action for a sequence of poses, which represents human motion. Models such as LSTMs [40] are capable to solve these tasks with high accuracies. Depending on the nature of the input data, this task can be described as *segmented human action recognition* or *continuous action recognition* [43]. For segmented human action recognition, there is only one action that is demonstrated in the pose sequence. In continuous action recognition, the action sequences are not segmented beforehand, thus there might be several different actions in the motion sequence as well as transitions between them. In general, the task of segmented human action recognition is a bit simpler. Another factor besides classification accuracy that often needs to be taken into consideration for continuous action recognition is latency since it is often needed for online tasks. [29] have shown, that the usage of so-called *action points* that uniquely identify an action can be used to enhance methods for continuous human action recognition. First, they introduced the *firing hidden Markov model*. It is based on two models: a global model with the actions and their state transitions, and a single-action model for each action, which is a chain of forward-linked states and the *firing states* that represent the action points. This architecture makes it possible to detect actions even if they are executed differently. Second, they used a random forest classifier to determine action point classes, which correspond to the detected action. In general, especially for assembly tasks where the sequence of tasks is known, hidden Markov models can be used to model the different states and transition probabilities to improve human action recognition [21].

2.4.3. Anomaly Detection in Trajectories

Anomaly detection in trajectories is often needed for analysis of video sequences, e.g. for surveillance tasks. The first step for solving these anomaly detection tasks in video sequences is tracking the objects in the video sequence. According to Rikard Laxhammar et al.[18], these approaches for anomaly detection can usually be split into two types: Clustering-based approaches and non-clustering-based approaches. In clustering-based approaches, historical trajectories are first clustered, in order to learn cluster models that represent normal behavior. The clustering can for example be performed on a discrete representation of the trajectories via fuzzy C means [27] or hierarchical by first clustering the spacial and then the temporal properties [10]. Incoming trajectories can then be classified as normal or anomalous via the likelihood of the most probable cluster, which is often done by using bayesian inference. Non-clustering-based approaches include for example *support vector machines* [33], *time series discords* [13] or *self organising maps*[30]. For the detection of anomalous sub-trajectories, a *partition-and-detect* approach can be used, where a database of trajectories is first partitioned and then anomaly detection is performed on these segments [20]. Most algorithms for anomaly detection in trajectories classify the whole trajectory and assume that the whole trajectory is known, but especially for cases where anomaly detection is needed in a real-time scenario, we want to be able to detect anomalies for incomplete trajectories that are updated over time. Many of the previously mentioned approaches can be adapted to incremental online anomaly detection. However, Rikard Laxhammar and Göran Falkmar[18] designed an algorithm called *sequential Hausdorff nearest neighbor conformal anomaly detector* (SHNN-CAD), which was specifically designed for online anomaly detection. It is based on the Conformal Anomaly Detector framework [19] and is parameter-light: besides the k -parameter for the nearest neighbor approach it only requires the anomaly threshold, which determines the score a trajectory needs to surpass in order to be detected as an anomaly. Guo et al.[9] proposed SHNN-CAD+ as an improvement to the standard SHNN-CAD algorithm, which does not need a predefined anomaly threshold, but adaptively determines it by itself and also uses an improved Hausdorff distance measure. However, one may not want to label a whole trajectory as normal or anomalous, but detect anomalous sub-trajectories instead. For such cases, the *sliding window local outlier conformal anomaly detector* (SWLO-CAD) [17] and *sub-sequence local outlier inductive conformal anomaly detector* (SSLO-ICAD) [16] were introduced. These previously mentioned algorithms work well with trajectories that just represent positions. For trajectories that need to represent more dimensions like for example velocity, the *sequential multi-factor Hausdorff nearest neighbor inductive conformal anomaly detector*[31] was introduced.

2.4.4. Anomaly Detection based on Human Behavior

An important application for anomaly detection in human behavior is for example fall detection. In an early solution to this problem, Y. Choi used accelerometer- and gyroscope data as an input for the Naive Bayes Algorithm to predict whether the human performs a normal action, such as walking or standing, or a fall[6]. However, such problems can also be used by anomaly detection algorithms. In this area, it is often important that the anomaly detection algorithm can adapt to changing behavioral routines. Salisu Wada Yahaya et al. [41] solved this problem by overwriting the old data with the new normal data depending on two forgetting factors based on data aging and data dissimilarity. In another paper by Salisu Wada Yahaya et al.[42] an intermediary robot was used to retrieve feedback on the detected anomalies and update the algorithm accordingly. However, not only human motions can be observed to detect anomalies. For example, Reuben Aronson et al.[2] use the human gaze to detect anomalies for system failures in human-robot collaboration. This works well since the gaze is highly responsive and also is a strong indicator for anomalous events.

3. Approach

3.1. Overview

Experiments in the past have shown that collisions in our example training scenario are most likely to happen in situations, where the actions of the human differ from the expected behavior. The reason for this is that the robotic support in such schooling scenarios is already designed in a way, that collisions are avoided if the human behaves in an intended way. So our first goal is to create a model that can detect significant deviations between the expected and the actual human behavior. Based on that, if unexpected behavior is detected, we next need to create a suitable method to adapt the velocity of the robot. This should happen in a way that the collision is either avoided or if the collision cannot be avoided, it is weak enough so that no injuries are caused. Furthermore, the robot should return to its original speed as soon as the human behaves as expected again. The designed approach should not only fulfill the safety requirements towards HRC sufficiently but also provide high efficiency in terms of speed. As a precondition for the following approach, we assume that we are in a situation similar to a schooling scenario. This means it should apply to any situation where collisions should not appear if the human only performs the intended actions. We want to solve this task by using a hierarchical approach. First, we train a machine learning model that, given the predefined actions in the schooling scenario, outputs the current action that the human performs. If the detected action deviates from the set of actions that are expected by the training, we consider the human action as an anomaly. Otherwise, given that the user performs the correct action, we perform the actual anomaly detection where we consider the human behavior as anomalous if it deviates too much from that action. For this, we pick the respective pose set for the respective action and use an anomaly detection algorithm to iteratively check for anomalies. Here we first introduce an intuitive method that is directly based on the returned likelihoods of the generative machine learning model, which will serve as a baseline in the evaluation. In a second method, we apply conformal anomaly detection to our problem. As discussed in section 2, there are many solutions for similar problems that utilize methods that require a high amount of training data, for example, LSTMs. The main reason why we are using more statistical approaches instead of neural networks is that we are limited in terms of training data. For many tasks, neural networks require a lot more training data than classical statistical solutions. However, on one hand, recording such a huge amount of training data would not be feasible within the timespan of this thesis. On the other hand, we want to propose an approach that can easily be applied to other scenarios with the same prerequisites. This also means that the training data should be collectible within an acceptable amount of time.

3.2. Detection of Unexpected Human Behaviour

In this section, we first generalize the schooling scenario and then introduce our anomaly detection framework. We discuss the requirements for the framework for the environment, how it works and how it can be customized for alternative environments.

3.2.1. Prerequisites

To be able to formally solve the problem, we first need to describe the context of our problem more abstractly and define the requirements for the dataset.

Generalization of the Environment

The training consists of a series of predefined tasks which must be solved in a certain order. Each task must be solved by a corresponding action of the user. That implies that there is an action sequence that the user has to follow. Let A be the set of all possible actions and $a \in A$ a certain action. Then we can denote $N(a)$ as the set of allowed successor actions. In our case, the length of $N(a)$ equals three for every $a \in A$ since the user is always allowed to continue the current action, wait or progress to the next action. For each frame, the training outputs the joint positions Pos and orientations Or , each of shape $(32, 3)$, which represent the joint positions and angles orientations detected by the camera for 32 different joints.

Dataset Requirements and Processing

We define the recorded raw dataset containing the different training runs with all joint positions and orientations for each timestep as X^{raw} . In order to train models and be able to evaluate them, the dataset must be labeled manually first. In total we need to extract three different types of data:

1. A dataset where one sample represents all joint values for one action of one recording for the action recognition
2. A graph-like structure that represents all states and their possible successor states
3. A dataset where one sample represents all joint values for one recording for the evaluation and each timestep is labeled

The first dataset can be defined as tuple $D = (X, Y)$, where X contains the segmented data for one action and Y the corresponding labels. That means that one sample $d_i \in D$ contains one sample $x_i \in X$ of shape $(timehorizon, \#joints, 6)$ which contains the 3d positions and orientations in eulerian angles for all joints for each timestep, and $y_i \in A$ which corresponds to the action label. This also includes the “anomaly” label for anomalies.

The transitions for the graph-like structure can be extracted by adding all observed pairs of states and successor states in the dataset to the structure.

The last dataset results in a tuple $D = (X, Y)$, where $x_i \in X$ contains the data of one recording and each timestep $x_{i,t}$ of that recording has an action assigned, which is given by $y_{i,t} \in Y$.

3.2.2. Anomaly Detection Framework

Since our scenario provides task sequences that the user has to follow, anomalies are context-dependent. So even if the action of the user is not anomalous itself, we still consider it as anomalous if it belongs to a currently relevant task. This means there are two types of anomalies. First, the *action-level anomaly*, where the recognized goal of the user is not contained in the current set of allowed tasks and second the *context-independent anomaly*, where the user might execute the correct action, but the action itself is anomalous. That is why the proposed anomaly detection framework consists of two stages: First task recognition is executed to perform action-level anomaly detection. If there are no anomalies in this stage, then the training actions of that goal are used to perform anomaly detection within the action.

In that case, we would assume that there is a way to obtain the current task via the simulation. If this is not possible, we are not able to perform context-dependent anomaly detection, which is essential for detecting unexpected behavior in training scenarios, since we also consider a “correct” trajectory at the wrong time as an anomaly. However, we can still use context-independent anomaly detection to detect behavior that is anomalous for any action. In this case, we might also want to use a task recognition step to predict the current task the human is completing and use the corresponding training trajectories for context-independent anomaly detection. An overview of the framework is shown in figure 3.1.

Action-Level Anomaly Detection

This step is equal to continuous action recognition and consists of two phases: an offline training phase and an online testing phase. For both phases, we first need to extract the features Φ for each sample in the training- and test set. For the action-level anomaly detection, we are only interested in the normal actions, so we only use those samples $x_i \in X$ where $y_i \neq anomaly$. As mentioned in section 2.2.2 the feature extraction can either return frequency domain features or time domain features. For frequency domain features, we simply extract the required joint positions and/ or orientations from the dataset. For time domain features, we use a window over the time horizon in addition to the required joint positions and orientations and calculate the mean, variance and root squared mean error for one dimension of the position or orientation of one joint for each dimension of the required joints. Both methods result in a set of features ϕ_i for each training sample x_i .

Then in the offline training phase we fit one GMM G_a using the features ϕ_i of each timestep of those training samples x_i where $y_i = a$ for each action $a \in A$ *anomaly*. The fitting is done using expectation maximization. In the online testing phase, we obtain one update X_t^{raw} of a trajectory for each timestep and want to assign it to an action label $a \in A$. The required method here depends on the previously used feature extraction method. If frequency domain features were used, then we need to sum up the log-likelihoods for all incoming updates as done in Jim Mainprice et al. [22]. In contrast to the original paper, we need to use a window-based approach here to use this method for continuous action recognition. This means we define a window W of size n , to which the extracted frequency domain features of all updates are added until it reaches size n . Before adding a new incoming update to the window, we first have to pop the oldest element from the window. Let ϕ_t^{test} be the extracted feature set for X_t^{raw} . Then the window for any timestep $t \geq n$ has the following form:

$$W_t = \{\phi_{t-n}^{test}, \phi_{t-n+1}^{test}, \dots, \phi_t^{test}\}$$

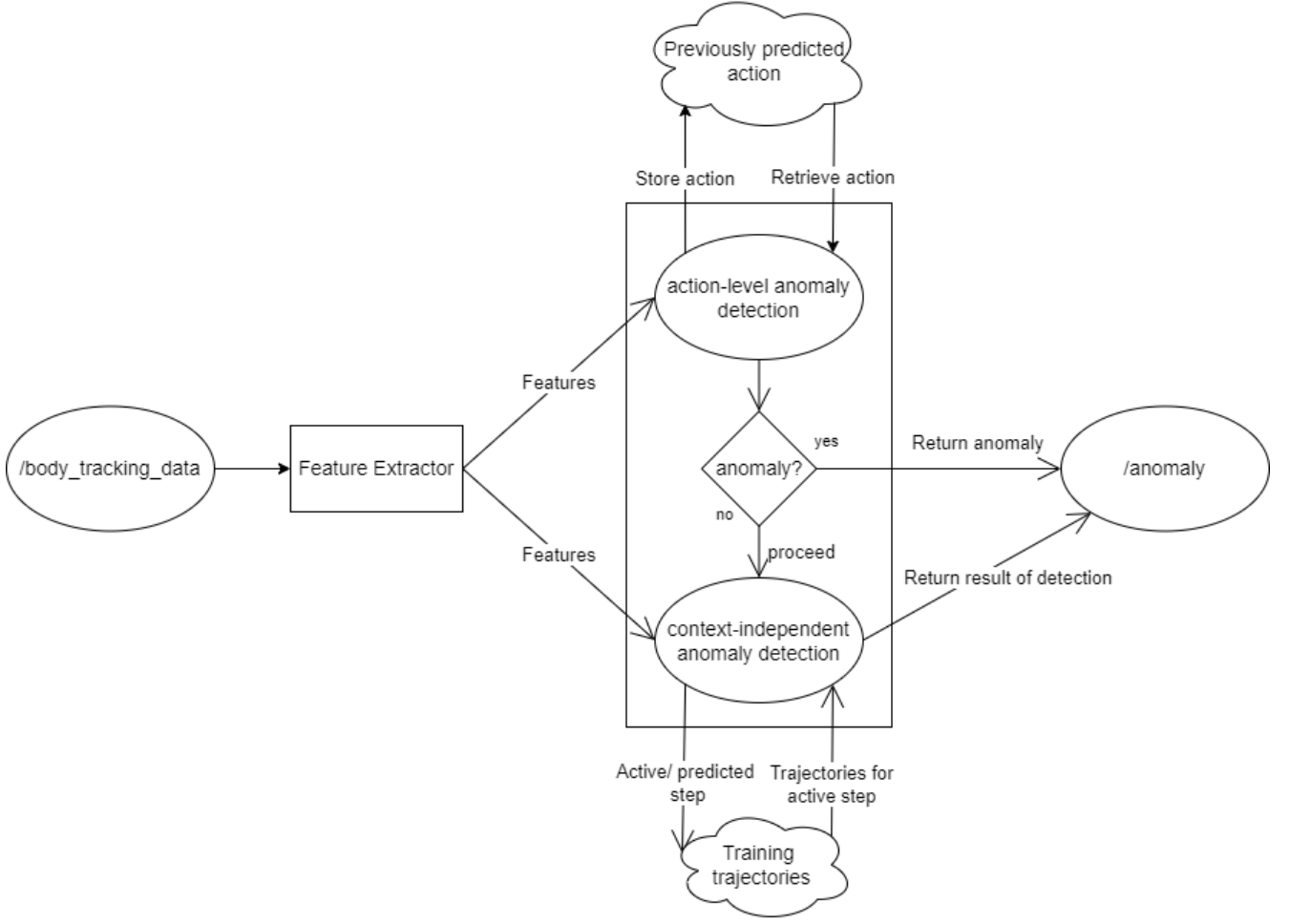


Figure 3.1.: Anomaly detection framework

For the classification of one timestep t we evaluate the window W_t . Let $G_a(\phi_t^{test})$ be the log-likelihood for update ϕ_t^{test} and G_a . Then the predicted action for W_t is given by:

$$pred_t = \operatorname{argmax}_{a \in A} \left(\sum_{\phi_i^{test} \in W_t} G_a(\phi_i^{test}) \right)$$

On the other hand, if we used time domain features we add the raw data to the window so that it has the form:

$$W_t = \{X_{t-n}^{raw}, X_{t-n+1}^{raw}, \dots, X_t^{raw}\}$$

The features for that window are then calculated in the same way as in the training phase. So we obtain the feature ϕ_t^{test} for every window W_t . We then only use this feature for the classification:

$$pred_t = \operatorname{argmax}_{a \in A} (G_a(\phi_t^{test}))$$

With that prediction, we can check for anomalies in timestep t . Given the current state s of the training, we query the set of the valid successor states $N(s)$ and check whether it contains the prediction $pred_t$. If it contains the prediction we consider the behavior in timestep t as normal and anomalous otherwise..

Context-Independent Anomaly Detection

For the first method, we use the outputs of the GMMs immediately to predict, whether the current behavior conforms to any of the possible actions. We can formalize this by using some terms of conformal prediction, especially the prediction set. The prediction set for the timestep t is then given by all action labels for which the summed log-likelihoods over window W_t for action a takes a value above ϵ . So for frequency domain features we define P_t as:

$$P_t = \{a \in A \mid \sum_{\phi_i^{test} \in W_t} G_a(\phi_i^{test}) > \epsilon\}$$

It is important to mention that this definition of a prediction set differs from the definition used for conformal prediction since it does not utilize any nonconformity measures. We only use this term here to give a better formalization of this method. According to the intuitive definition of anomalies as "patterns that do not conform to the expected behavior" [5], we consider the behavior in the current timestep as anomalous if there is no action that can be predicted with sufficiently high certainty, which means that the prediction set P is empty. Let $Anom_t$ be a boolean variable, that shows whether there is an anomaly in timestep t . Then it takes the following values:

$$Anom_t = \begin{cases} 1 & P_t = \emptyset \\ 0 & \text{else} \end{cases}$$

The sensitivity of the anomaly detection can be configured via the parameter ϵ . Alternatively, we could immediately calculate $Anom_t$ without the use of prediction sets via:

$$Anom_t = \begin{cases} 1 & \max_a (\sum_{\phi_i^{test} \in W_t} G_a(\phi_i^{test})) \leq \epsilon \\ 0 & \text{else} \end{cases}$$

As a second method, we want to apply the CAD framework to our case. In contrast to the first method, this method does not work window based, but only uses the respective poses directly. Here we first need to define an anomaly threshold ϵ , a training set $\{z_1, \dots, z_n\}$ for each action, and an NCM A to calculate the α values for the training set. We obtain the training set by drawing n samples from the corresponding GMM. To calculate the NCMs, we use the same method as given in section 2.3.2 by Papadopoulos et al. [32]. The only difference is that instead of the output of a neural network, we use the likelihood returned by $G_a(\phi_i^{test})$:

$$\alpha_i = \max_{j=1, \dots, c: j \neq u} G_j(\phi_i^{test}) - G_u(\phi_i^{test})$$

Then we can calculate the p-values for the current action:

$$p_{test} = \frac{|\{j = 1, \dots, l+1 : \alpha_j \geq \alpha_{l+1}\}|}{l+1}$$

Here, we base the anomaly detection on how much the pose of the current timestep differs from other examples of only that action. So another difference to the first method is, that to execute the anomaly detection on timestep t , we need the predicted action from the current timestep t . Now we only calculate the p-values for that action a_t . This leads to the following definition of $Anom_t$:

$$Anom_t = \begin{cases} 0 & p_{a_t} > \epsilon \\ 1 & \text{else} \end{cases}$$

4. Setup and Evaluation

4.1. Setup

There are many different components on which this work is based. In this section, we go over the basics of the simulation and the hardware that are necessary to understand the implementation and evaluation.

For this work, we are using a Sawyer cobot which has one arm and seven degrees of freedom. To control the robot, we are using the Itera SDK on ROS Noetic together with Ubuntu 20.04. To obtain information about the human joint positions and angles, we capture the human motions with an Azure Kinect, which uses a depth camera and an RGB camera for the recording. The recorded depth- and RGB images are stored in .mkv files and then processed using the offline processor by Azure ¹. Since the manufacturer of the camera recommends more advanced hardware for body tracking, we use a laptop with an Nvidia GeForce GTX 3060 Laptop GPU, where we installed CUDA to run the body tracking on the GPU. The virtual reality simulation is implemented in Unity and uses an HTC Vive and two trackers, which are tracked by two base stations that are oriented towards each other on the corner of the workspace.

4.1.1. Architecture Overview

The training consists of two main components: Unity for the VR simulation and ROS for the communication with the robot arm. First, the simulation retrieves the robot's position via the two trackers. One tracker is placed on the end effector of the robot and the other one is first placed on the head of the robot for the calibration, and then used for the actual schooling as a wrench. After the robot is calibrated, the coordinates of the interfaces of the gripper station in the VR simulation are calculated and then published to the `\interface_positions` topic. Then the training starts and the first goal for the user is activated and marked in the simulation. In this training, the robot is needed for the first goal. So the ID of the first interface is published to the `\active_interface` topic and the trajectory to the corresponding interface position is planned and executed by MoveIt. It is important to mention that in the meantime the user might complete other goals where the robot is not needed. During the whole training, the Kinect camera captures the human movement and publishes the body tracking results to `\body_tracking_data`. The body tracking results contain the detected position and orientation in quaternions of each of the 32 joints. The mapping of the joints to the human body is shown in figure 4.2. The anomaly detection script uses this data to update the human trajectory and executes the anomaly detection algorithm for the updated trajectory. The results of the anomaly detection algorithm are published to the `\anomaly` topic. The structure is shown in figure 4.1.

¹<https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples>

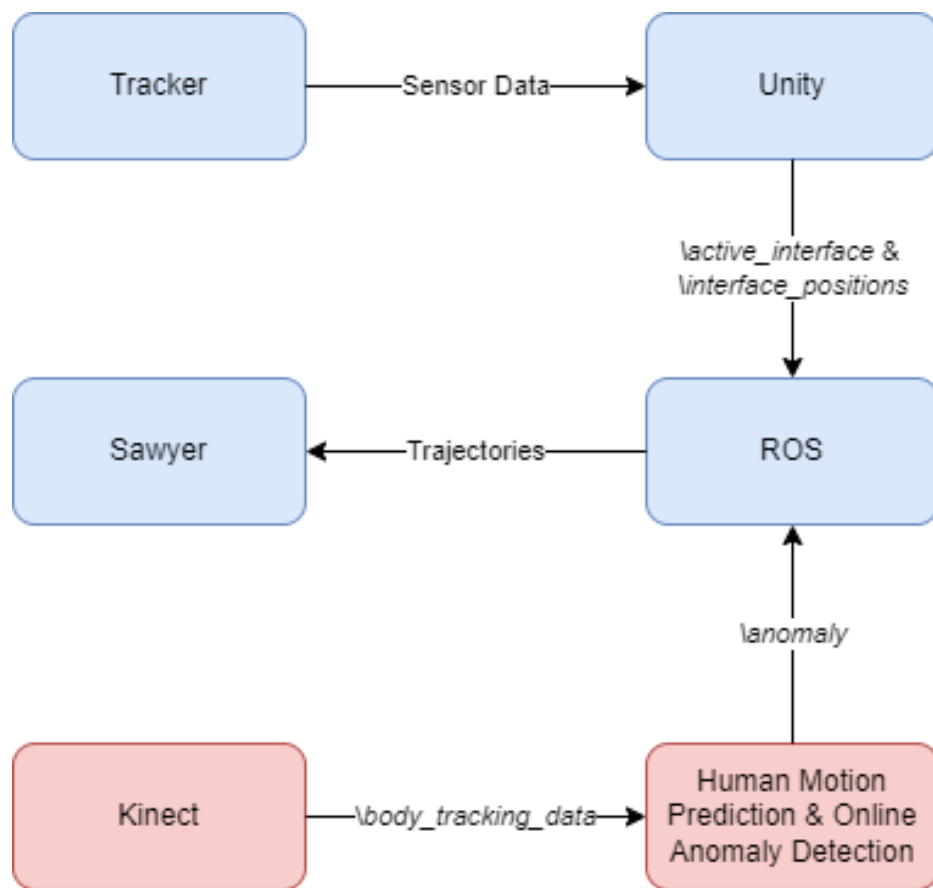


Figure 4.1.: Structure diagram: the blue components already existed before and the red components were added and implemented for this thesis

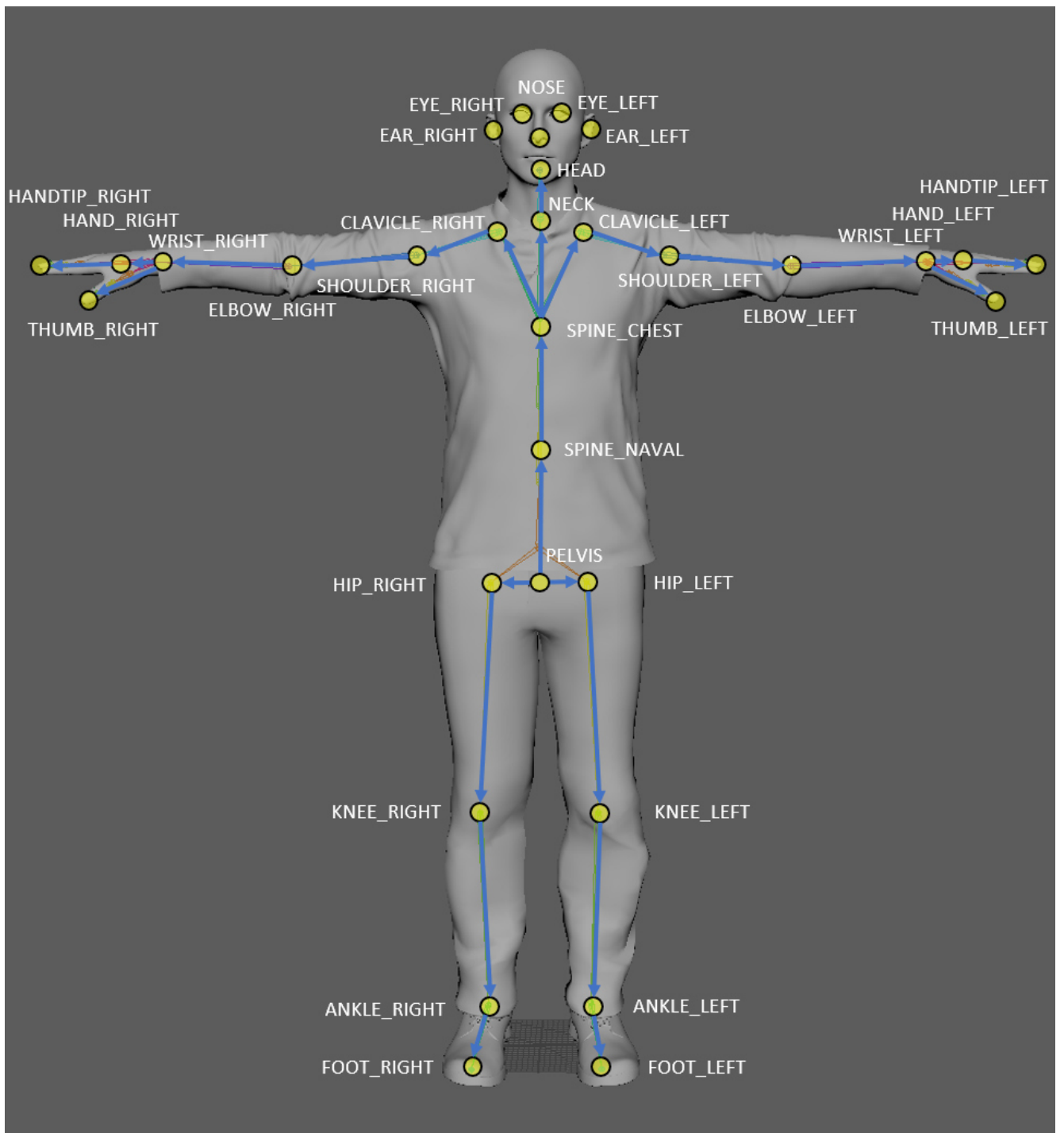


Figure 4.2.: Joint hierarchy for kinect body tracking²

²<https://docs.microsoft.com/de-de/azure/kinect-dk/body-joints>

4.1.2. Components

As already shown in the overview, the training utilizes some special hardware. In the following, we will briefly discuss them and explain the software in further detail.

Virtual Reality Training in Unity

In the virtual reality training for which we test this method, the user has to maintain a gripper station. The whole training consists of seven steps which have to be completed in the following order:

1. loosen the upper screw
2. loosen the lower screw
3. remove the gripper station
4. insert the gripper station
5. push the pin
6. adjust the gripper station
7. push the pin again

Between the removal and the insertion of the gripper station, the user has to carry the gripper station to another position. That means that there is an additional action between these steps which one could call “move gripper station”. Since, out of these three actions, carrying the gripper station is the part that is most relevant for anomaly detection, we will just summarize this into a single action “move gripper station”. So for the rest of this thesis, we will work with the following six steps:

1. loosen the upper screw
2. loosen the lower screw
3. move the gripper station
4. push the pin
5. adjust the gripper station
6. push the pin again

It is important to mention, that the human can complete step three while the robot is moving. The gripper station and the different steps are shown in figure 4.3. We also decided to include a *wait* step for our anomaly detection method, which can be entered after any other action. The reason for this is that the user often has to wait until the robot arm reaches its target or needs some time to identify the next objective. All of these tasks require an interaction with the corresponding object within its bounding box. For the first two actions, the tracker must be moved within the bounding box of the respective screw and for the other actions, the user has to use the hand to interact with the objects within their bounding boxes. As soon as the simulation detects that the user has completed the current task, it marks the next one and, if needed, publishes the next active interface to `\active_interface`. This is done via the game object “ROS connector”. To synchronize the position of the robot in the simulation to its real position, the user has to place the loose tracker at the top of the robot. The other one is fixed at the end-effector of the robot. Unity

then uses the positions of both trackers to correctly calibrate the position of the robot. The user can then take the tracker on top of the robot again to run the simulation. Figure 4.4 shows what the simulation looks like from user perspective.

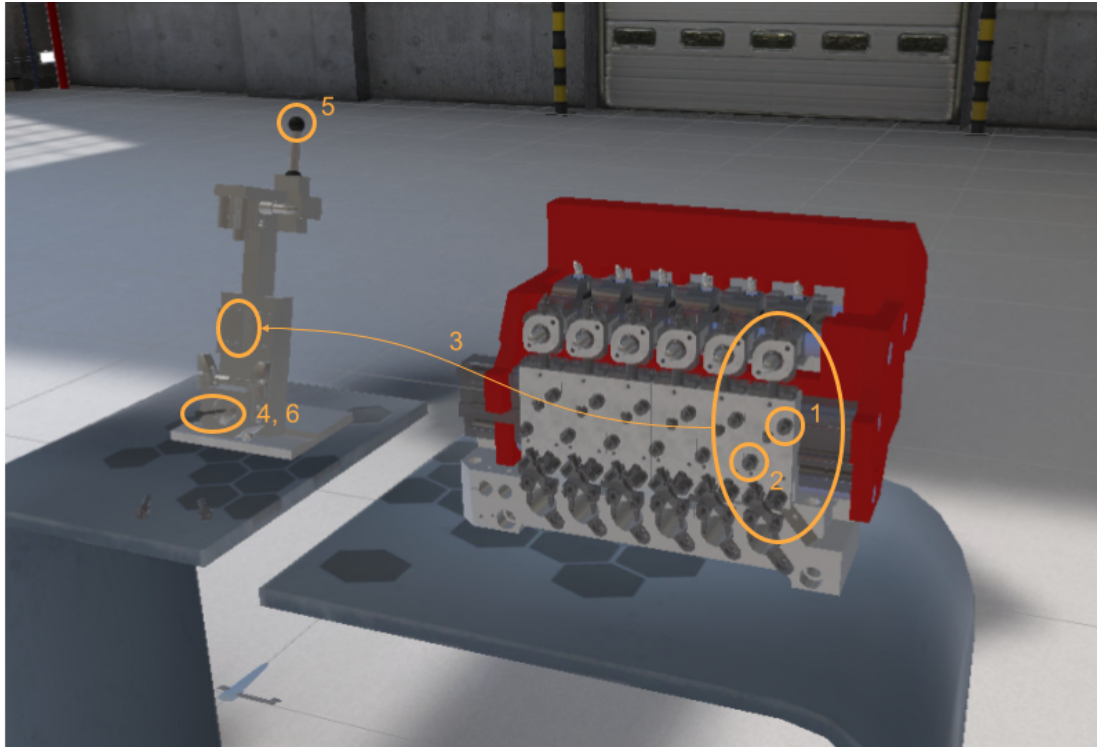


Figure 4.3.: The gripper station with orange circles around the targets of the different actions and numbers which display the order in which the interfaces are active.

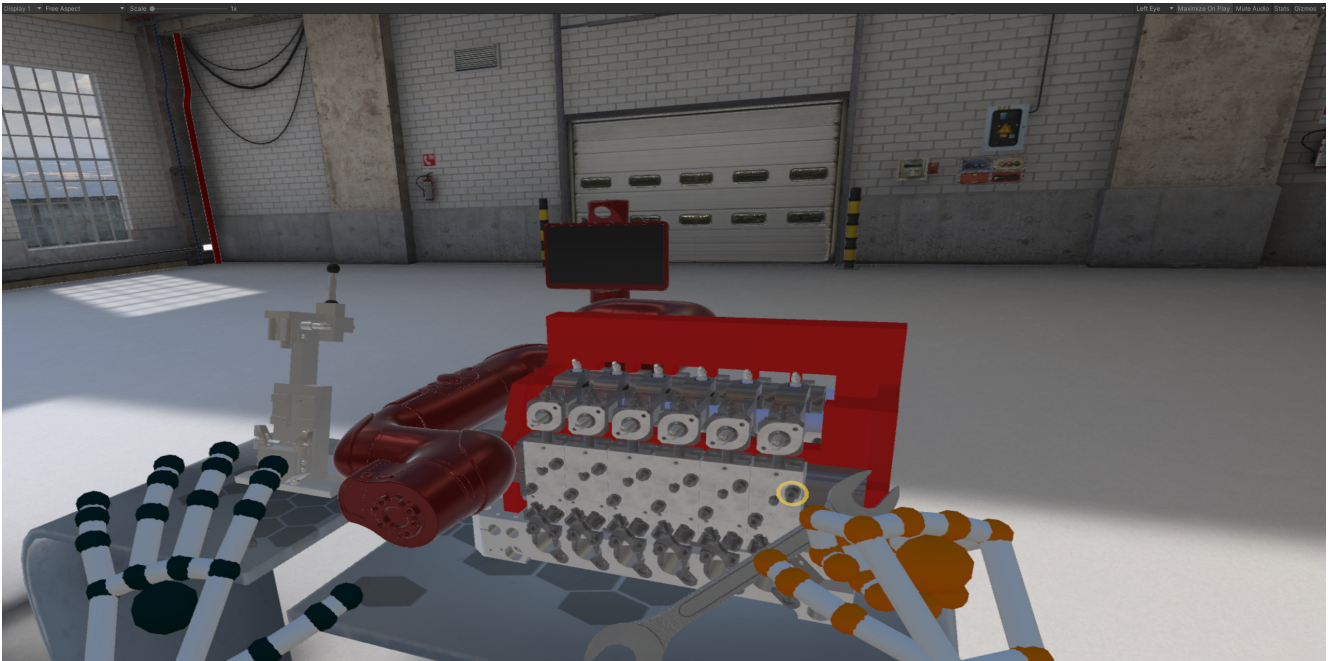


Figure 4.4.: Screenshot of the virtual reality training.

VR Headset and Tracker

As VR headset we use a Valve Index (figure 4.5a) with HTC Vive trackers, which communicate to the training via the SteamVR framework. The headset and the trackers are detected via two base stations in the corner of the workspace. One of the trackers is located at a wrench which the user needs to complete the training (figure 4.5c and 4.5d), and the other one is located on the end effector of the robot (figure 4.5b). However, not only the trackers are important to interact with the simulation, but the hands of the user are also essential for some actions. To track the position of the hands of the user, a Leap Motion controller is attached to the VR headset.



(a) Vale Index with Leap Motion controller attached



(b) Haptic interface at the end effector of the robot



(c) HTC Vive tracker with wrench attached - front



(d) HTC Vive tracker with wrench attached - back

Figure 4.5.: Hardware components

ROS

The robot has to move between the obtained active interface positions, depending on which interface is currently active. To plan and execute the trajectory, MoveIt is used. One restriction towards that trajectory is that it has to plan a trajectory behind a collision plan, so that the robot does not hit the user. After reaching the interface position, the robot provides haptic feedback for the task corresponding to the interface position. This was implemented in a previous master's thesis by Ben Kirsche [14]. In the context of our work, we added a subscriber to the `\anomaly` topic, which contains information about detected anomalies. Depending on that results, the speed is adapted as explained in section 4.2.2.

Camera



Figure 4.6.: Azure Kinect

Three main aspects influenced our decision about the camera placement. First, we could observe that in some scenarios where the robot arm partially covered up the human body, it was recognized as part of the human joints by the body tracking algorithm. Second, the other option in these scenarios is that the human was not detected at all. We could observe that this is most likely to happen if parts of the upper body, especially the head, are hidden. Last but not least, the whole method should be applicable for the virtual reality training, which means that both trackers should be visible for the base stations all the time. This leads us to a camera position, where the camera focuses the workspace sideways and slightly at an angle so that the human is on the left and the robot is on the right side of the picture. Furthermore, the camera is positioned around two meters high. It should also be noted that, according to Carlos Morato et al. [26], it would have been beneficial to use more Kinect cameras to achieve more precise body tracking results. The final camera view as well as the outputs of the infrared-, the depth- and the RGB camera, are shown in figure 4.7.

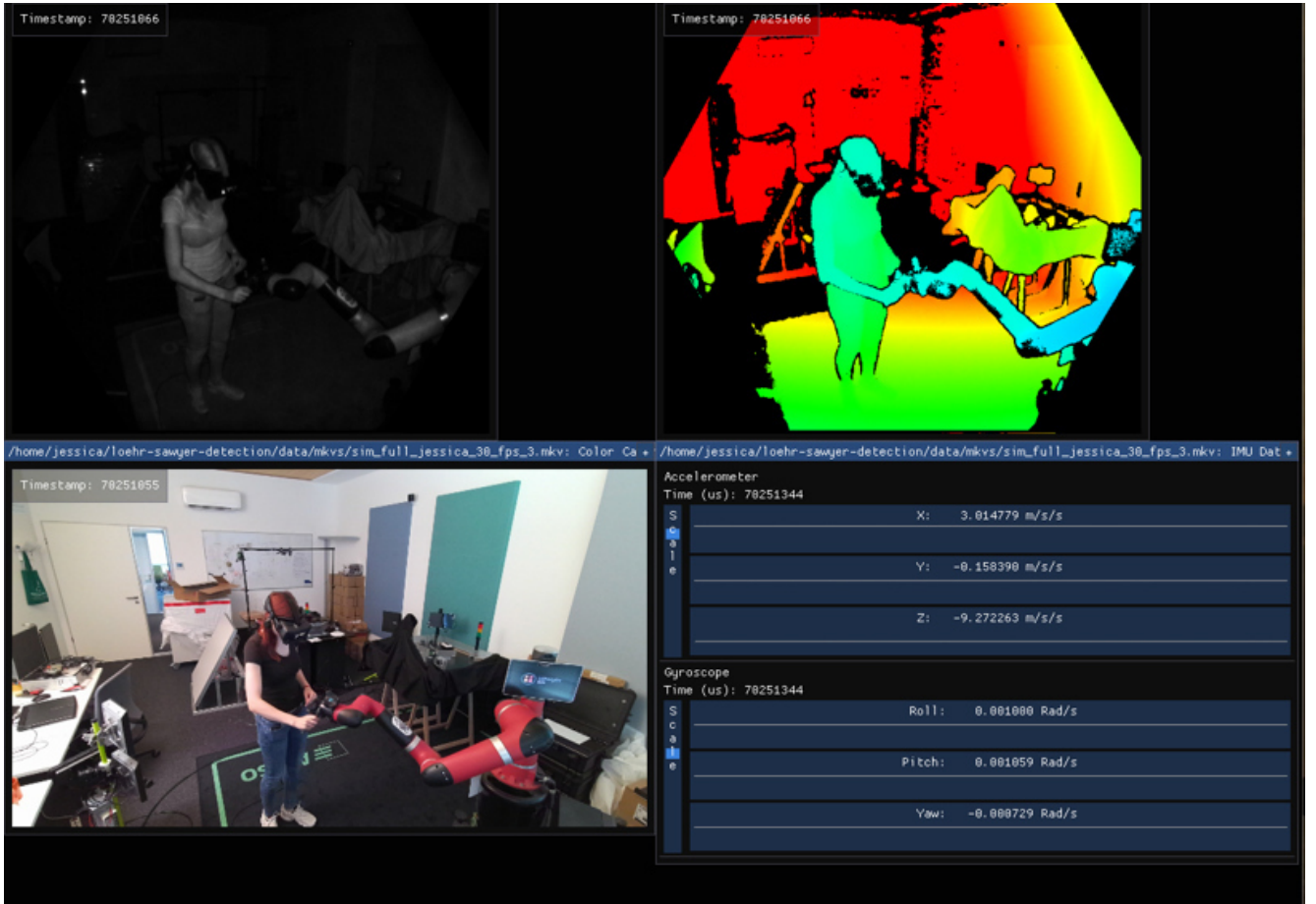


Figure 4.7.: Screenshot of a recording with picture of infrared camera in the upper left, depth camera in the upper right and rgb camera in the lower left corner

4.2. Detecting and Handling Unexpected Human Behaviour

In this section, we first describe how the training works from a technical point of view. After that, we explain how our approach proposed in section 3 was implemented and how it is included in the existing codebase.

4.2.1. Anomaly Detection Implementation

The anomaly detection is implemented as a separate ROS node, which subscribes to the topic `\body_tracking_data` and publishes to `\anomaly`. Each time a new message is published to `\body_tracking_data` the wrapper (algorithm 3) for both anomaly types is called. First the function `actionLevelAnomalyDetection()` evaluates the Gaussian mixture model for the current pose sequence, checks whether the predicted pose is in the set of allowed poses, and accordingly returns true for anomalies or false for normal data. If an anomaly is detected in this step, the anomaly is published and the function terminates. Otherwise the function

contextIndependentAnomalyDetection(), is called and its result published. This function just calls a CAD algorithm and returns its result.

```

Data: body tracking data
actionLevelAnomaly = actionLevelAnomalyDetection();
if actionLevelAnomaly = true then
    | publish(true);
else
    | contextIndependentAnomaly = contextIndependentAnomalyDetection();
    | publish(contextIndependentAnomaly);

```

Algorithm 3: Anomaly Detection Algorithm

For the Gaussian mixture model in the action level anomaly detection, we use scikit learns GMM class.³

4.2.2. Velocity Adaption Implementation

We use the MoveIt framework⁴ to plan and execute trajectories. In MoveIt, a trajectory is represented as *robot trajectory message*, which can either contain a *joint trajectory* or a *multi DOF joint trajectory*. For our case, we decided to use joint trajectories, but the other option would be fine as well. In a joint trajectory, a trajectory is represented as an array of points. A point contains the positions, velocities, and accelerations for each joint and the time at which this point should be reached in relation to the start time of the trajectory. Furthermore, the message contains a header with metadata and an array of strings where each element corresponds to the joint name represented by the respective index. Currently, MoveIt itself does not provide a function to reparametrize a trajectory during execution. This means the reparametrization has to be done manually. To do this, we replace the current trajectory with a new one by scaling the time, velocities, and accelerations for each remaining point of the current trajectory while keeping the positions. Furthermore, we drop all points that the robot has already reached. The rescaling for each attribute by the factor s is done by the following simple formulas:

$$\begin{aligned}
 t_{new} &= \frac{t_{old} - exec}{s} \\
 pos_{new} &= pos_{old} \\
 v_{new} &= v_{old} * s \\
 a_{new} &= v_{old} * s^2
 \end{aligned}$$

where t refers to the time, pos to the position, v to the velocity and a to the acceleration of the arm, and $exec$ represents the time since the beginning of the execution of the original trajectory and is needed to make the new trajectory starting at 0 seconds again. The variable s lies in the interval $(0, 1)$ and is, if the new value matches this interval, increased by 0.1 if the anomaly detection returns 0, and decreased by 0.1 if it returns 1. By that, the robot will stop if the user behaves unexpectedly for too long, and will not change its speed at all if the user follows the training correctly. However, problems may occur when replacing the old trajectory with the reparametrized one, if the current pose of the robot differs by more than 0.01 from the starting point given in the new trajectory. The reason for this is that a trajectory in ROS is not represented continuously, for example as a curve, but as a discrete set of waypoints. So when replacing

³<https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>

⁴<https://moveit.ros.org/>

the trajectories, the actual joint states of the robot might lie exactly between the last joint states of the old trajectory and the first of the new one, which can cause huge deviations. To avoid this problem, we implemented the following workaround: While re-parametrizing the trajectory, the robot continues the execution of the old trajectory. We then remember the time when the first point of the reparametrized trajectory should be reached according to the original trajectory. Now the algorithm waits until the old trajectory is executed for that amount of time and then gives the command to execute the new trajectory. By that, we ensure better continuity and a smaller deviation between the actual joint states and the first joint states of the new trajectory. This problem might also be solved by setting the granularity of waypoints sufficiently high, but this requires a lot of memory.

4.3. Evaluation

In this section, we will first discuss how the data was collected, as well as cover some statistics for the dataset, which is later used for the evaluation. Then we will first evaluate performance of each component from a technical perspective before evaluating it in the real application.

4.3.1. Dataset

For this work, we want to test our method proposed in section 3 with a special VR training. This also means that we have to collect our dataset ourselves. In the following, we give some insights into how the data was collected and which observations we made. Furthermore, we explain how we preprocess the data to achieve good results.

Dataset Collection

Since our framework requires that the context-independent anomaly detection works with semi-supervised methods, we need to label our dataset. There are two ways in which the training data can be labeled, which differ in the step where the segmentation is done. The first option is to segment the steps of the training beforehand and then record each step on its own. Here the human would come from a resting pose, perform the action, and return to the resting pose. The second option is to record the training as a whole and then segment the training steps afterwards. This means that the starting- and end point of one action depends on the previous and the following step. However, when the training is used for a real application, the action of the user also highly depends on the previous and the next step. So to retrieve training data that represents the training for the real application as closely as possible, we segment the data into single actions after recording the whole training. We assumed that the training data resulting from 7 recordings should be sufficient. The reason for this is that Jim Mainprice et al. [22] used 7 training samples for a classification where they also used GMMs to differentiate between seven actions, which is close to our setting. Another advantage of this approach to collecting the data is that the sequence of actions can be extracted directly from the training data. In total, we recorded 5 training runs with 6 different persons. By recording the training with several different users, we obtain a higher variance in our dataset and thus can provide a more realistic evaluation. Before each session with a new person, we explained the whole procedure of the training with each step in detail, especially when the robot is needed for that step.

It is important to mention that the collected dataset is quite noisy. The reason for that is that in many cases either the haptic feedback does not work very well or the calibration is too much off. In detail, this affects the actions *loosen upper screw*, *loosen lower screw* and *push pin*. When loosening a screw, sometimes less than half a rotation is already sufficient to loosen the screw, while in other cases the user rotates the screw two times before the robot blocks any further rotation. In the latter case, the simulation will not mark the next target as it is supposed to, since the calibration was so much off that the rotations did not take place within the collider of the screw. That means the haptic interface in ROS registered the full rotation and assumes that the screw is completely loosened, while the simulation in Unity did not register any rotation or just parts of it. The same holds for pushing the pin. Sometimes the robot does not allow the user to push the pin in as much as the simulation needs it to register the push. In many cases, the user tried to push the pin with so much weight that the robot gave in, and hence the user stumbled. Furthermore, we could observe that the trajectory controller sometimes also failed during the execution for unknown reasons. This was mostly observed for trajectories where the arm went straight to the top. To summarize this, we could only record a few training runs where the robot provided flawless support. This also means that to complete the training, the user has to perform actions that should actually be considered as anomalous if the behavior of the robot was deterministic and thus make the dataset noisy. Also, since we wanted to record 5 complete training runs with 6 different persons for testing purposes, executing the training until we reach a total of five on the robot side perfectly working training runs was no option since a perfect run is rather an exception and thus this would take too much time. Furthermore, it would most likely lead to bad results for the real application if we only trained the model with human behavior for perfect examples while the appearance of bug and thus different behavior is actually the normal case. Another important aspect is that due to incomprehensible robot movement, for example, if the trajectory seemed unnecessarily complicated and long, the human also reacted with actually anomalous behavior. However, we could record 6 additional runs in total, where anomalous behavior occurred due to actual anomalous human behavior on which we want to test our algorithm.

There were also some factors considering human behavior when recording the data that should be mentioned beforehand. First, many users had problems with hitting the screw at the beginning due to poor calibration. However, in later recordings, the same users began to touch the haptic interface with their other hand first, before loosening the screw. Most of the users are right-handed, so they took the screwdriver with their right hand. For other steps like moving the gripper station, they also took their left hand in some cases. We could also observe that the user sometimes is not sure about the next step if the simulation showed unexpected behavior. Hence the users sometimes asked some questions during the simulation, including some demonstrations. We decided to cut these parts out since this would not happen if the user would execute the training as expected and thus we want to consider this as an anomaly, which we do not want to include in our training data.

Furthermore, even though we took many factors into account for the camera placement, the robot arm still covers the human body sometimes. In this case, the body tracking either could not track the human reliably or detected the robot arm as part of the human. That means we also had to cut these parts out. Since in such cases the body tracking algorithm estimates the positions and orientations of the joints, we cannot generate any statistics about this behavior.

Statistical Dataset Exploration

As already mentioned, the training consists of five different actions plus an additional waiting action. Figure 4.8 shows the sequence and durations of these actions for ten recordings that were labeled manually.

We can see that the durations of these actions differ a lot. The *waiting* action takes the longest time and appears between the steps *loosen lower screw* and *loosen upper screw*, and the steps *move gripper station* and *push pin*. The reason for this is that between these steps, the user has to wait for the robot to reach the new interface position. As we can see in figure 4.9, there is also a high variance in the duration of the single actions. In other words, the same action might be completed quickly in one run and takes more time in another one. This is especially the case for the “wait” action, where the duration completely depends on the trajectory that was calculated by the trajectory planner. However, in our training, the planner uses the first valid trajectory that it can find, which can be efficient in some cases, but in other cases, it can lead to unnecessarily long trajectories. This leads to a high variance in this action.

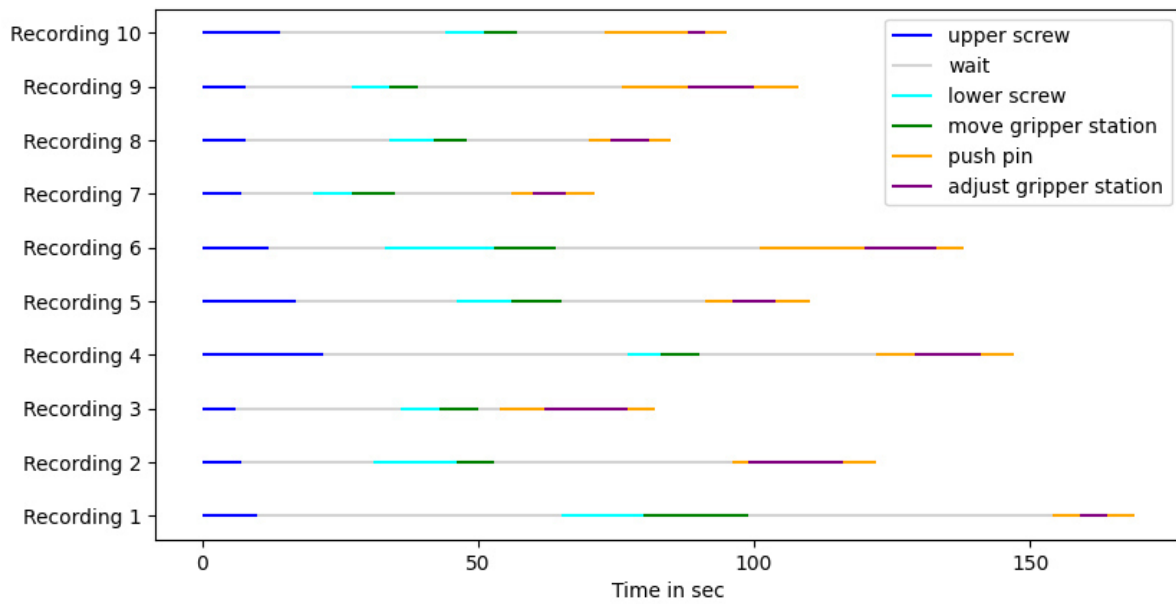


Figure 4.8.: Timeline

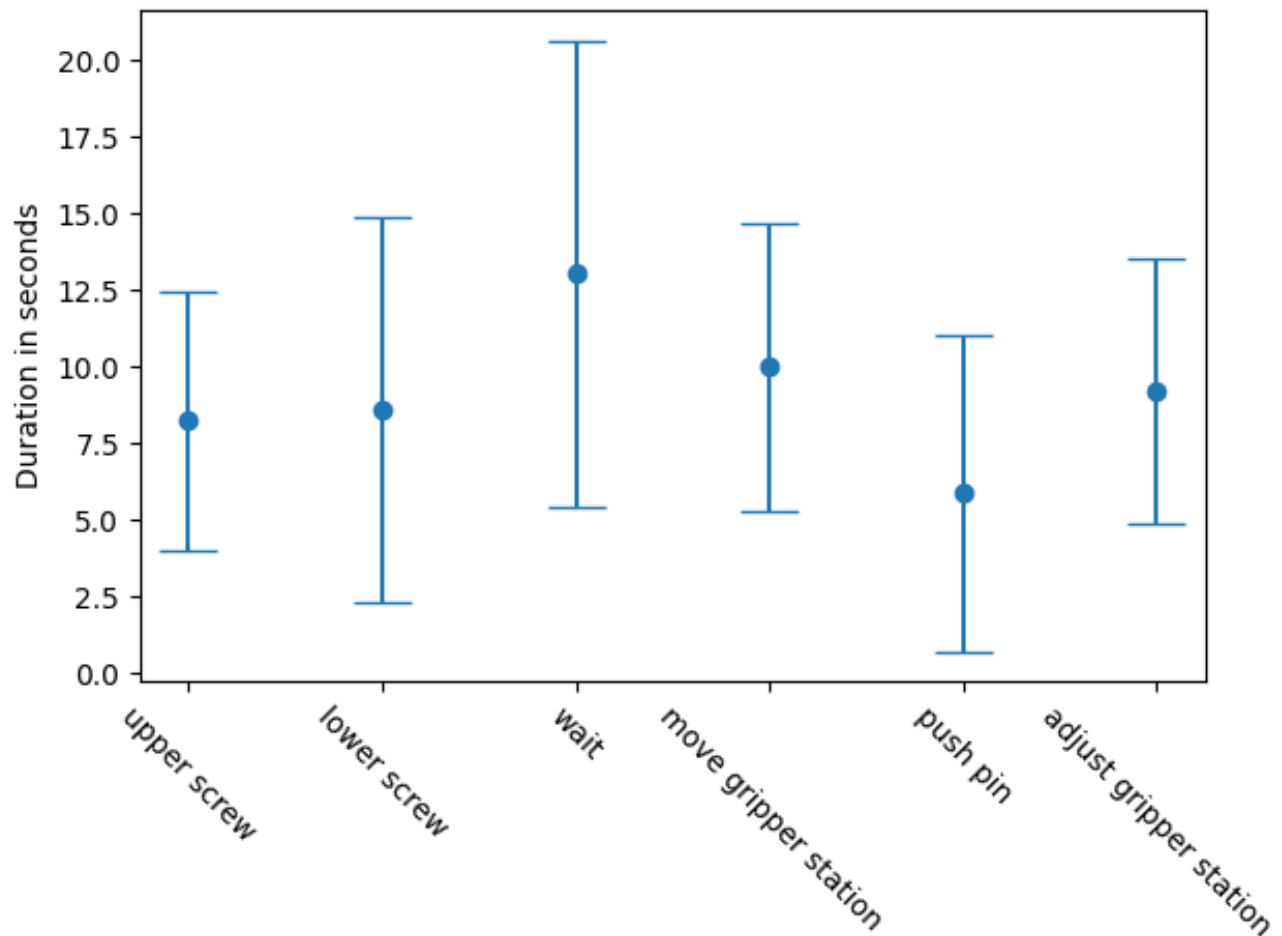


Figure 4.9.: Durations of actions

Preprocessing

As already mentioned in section two, one main challenge in the classification of time series is that different time series are of different lengths. Furthermore, users might complete the same actions with different velocities. As already shown in the previous section, we can assume that this is also the case for our data, since we have a high variance in the durations of the actions. Depending on the used classifier, this could lead to problems. To minimize this effect, we use dynamic time warping (DTW) to align the training trajectories in time. To do this, we have to select a reference trajectory to which we align all of the other training trajectories. Figure 4.10 shows the relevant trajectories for the “move gripper station” action without DTW. We can see that all of these trajectories have a similar shape, but they all have a significantly different lengths. Figure 4.11 shows the same trajectories, but they were warped to the same trajectory beforehand. We can see that the trajectories not only have a more similar shape but they are also aligned in time.

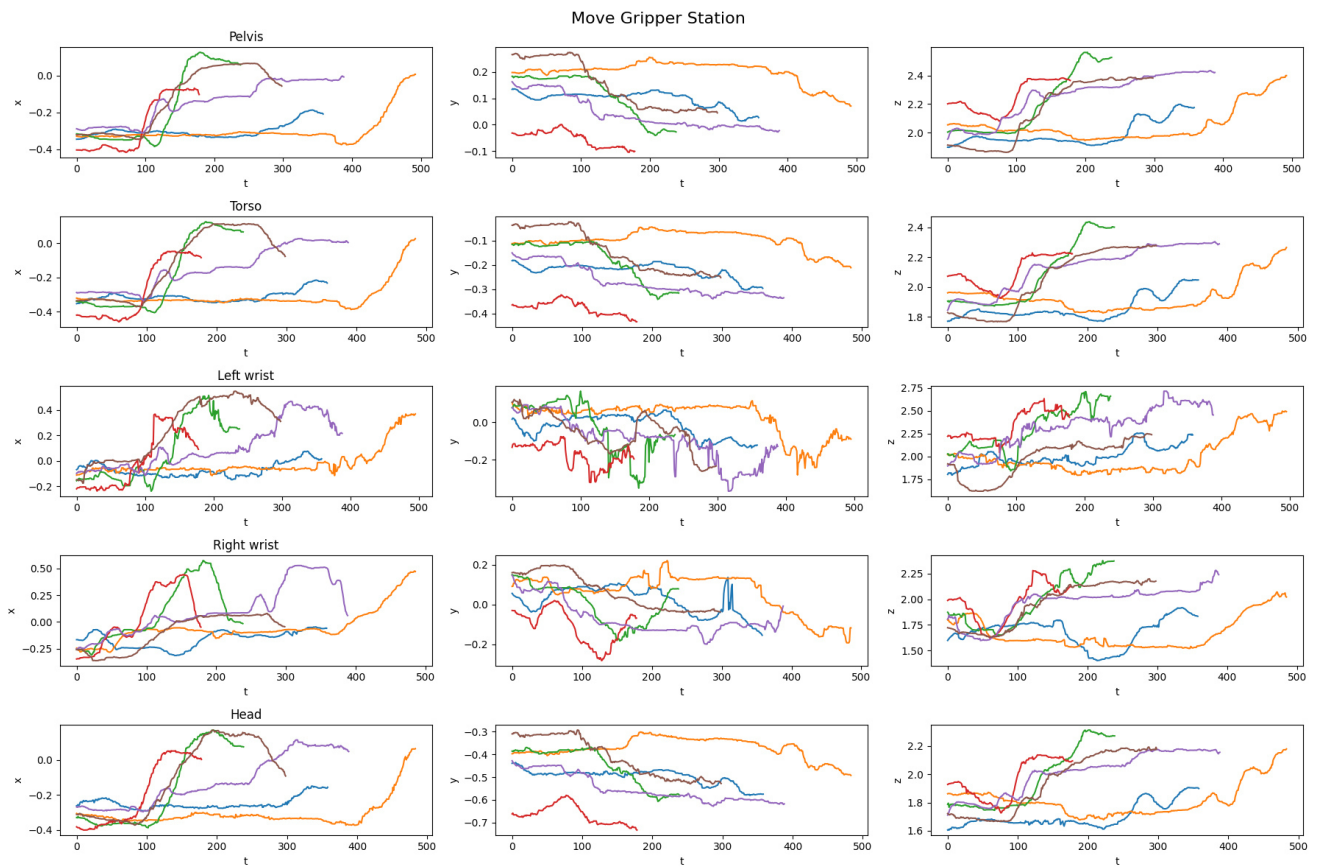


Figure 4.10.: Trajectories for the “move gripper station” action for all relevant joints and dimensions

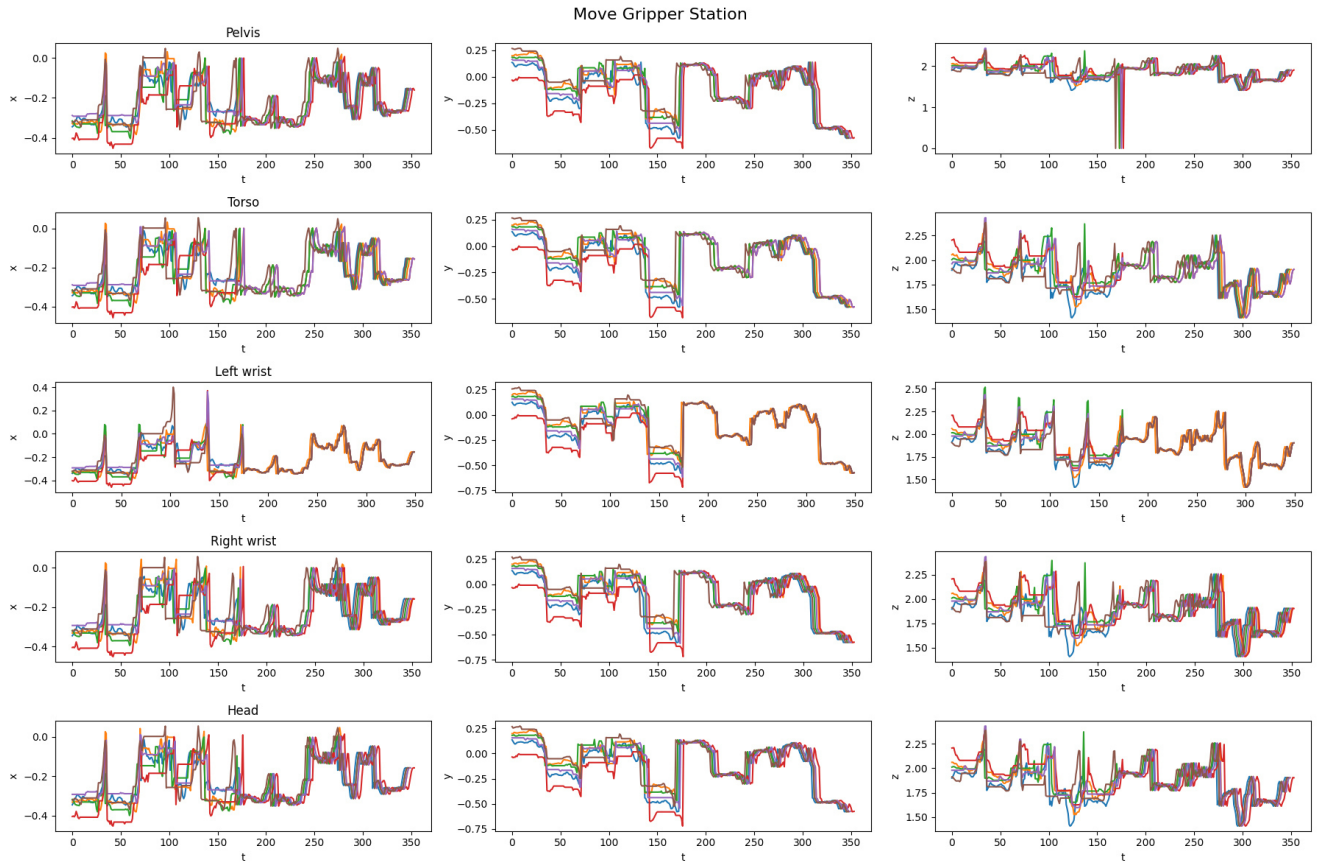


Figure 4.11.: Trajectories for the “move gripper station” action for all relevant joints and dimensions with DTW

4.3.2. Action Classification

In this section we will evaluate the action recognition on which the anomaly detection is based on. The success of such classification tasks heavily relies on the choice of features. Thus we will first discuss from which joint values we want to extract the features and which combination of joint values should be considered. Then we want to prove that GMMs are capable of representing the different actions in our dataset, which is done by evaluating the performance of the GMMs on segmented test sets where one test sample contains exactly one action. Since the actual application requires continuous action recognition, we also investigate how well this approach still works, if we use the window based approach proposed in section 3.2.2 first as a validation with the same test set and then with a test set where the samples contain a complete recording with multiple different actions. There we also compare the performance of the algorithms with time domain features to frequency domain features.

Feature Selection

In order to find the best features, we tested different combinations of joint positions of different length using cross validation among the users. That means we fit the model on the recordings of nine out of

ten persons and test it on the recordings for person ten. By that we want to achieve that the features represent the actions in such a way that the classification will work for any person despite the exact anatomy. Furthermore, we have to exclude some joints from the set of potential features since the camera is not able to track them reliably. This is the case for all joints below the knees, for all joints that are successors of the head, as well as as successors of the wrists. Due to promising results in the literature, we tested the following feature combinations:

1. Pelvis position and pelvis, torso and right wrist orientation: These are the features that were used in Mainprice et al.[22].
2. Pelvis position and pelvis, torso, right wrist and left wrist orientation: This extends the previous features with the left wrist orientation, which might be needed in our case since many users also used the left wrist in the training.
3. Pelvis position and pelvis, torso, right wrist, left wrist and head orientation: This extends the previous features with the head orientation. This might be useful to detect anomalies, since Aronson et al.[2] have shown that the human gaze is very sensible towards anomalies.

Mainprice et al. used this combination of position and orientation to be able to reconstruct the full human posture out of the features without the use of inverse kinematics. As we can see later, these combinations did not perform very well in our tests. Reasons for this might be that the orientations returned by the body tracking are not that accurate. That is why we decided to also test feature combinations that use the same joints as above, but only their positions instead of orientations.

However, there are also some problems with using absolute positions. First, the result might be dependent on the size of the user and second, for approaches such as GMMs, we are not able to cover temporal dependencies. As a result, we also calculated the differences between the positions of two timesteps and also tested them as features. This is done for the same feature combinations as above.

Performance per Action

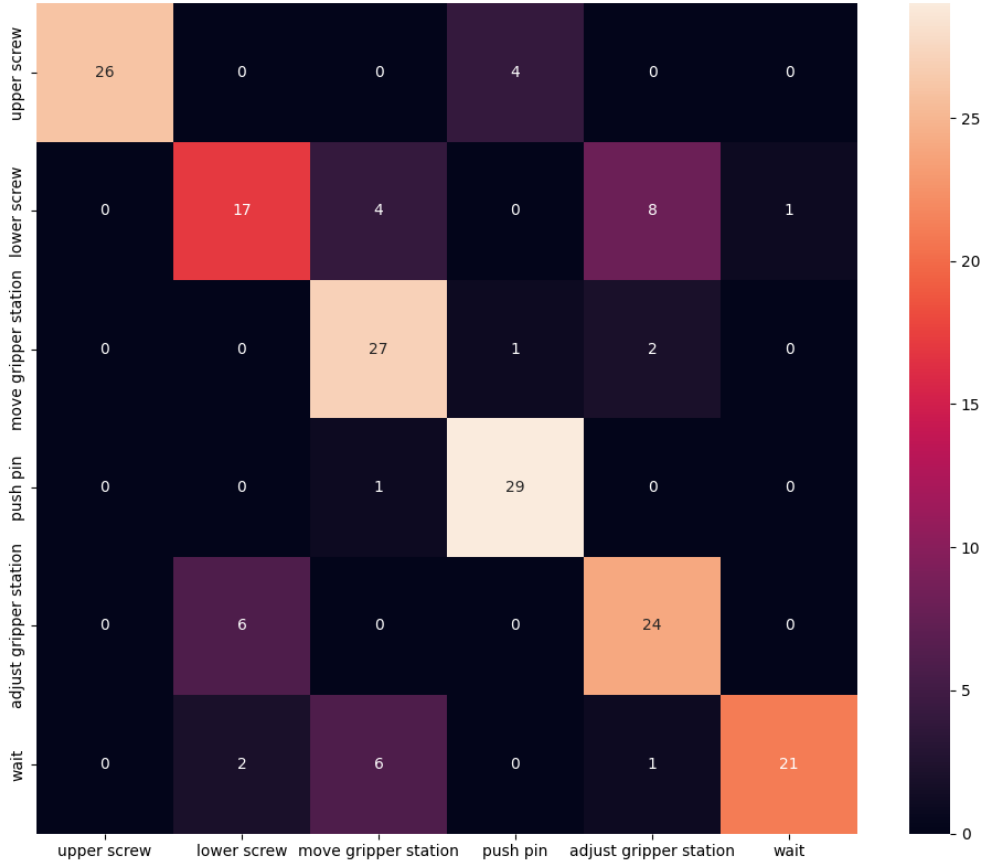


Figure 4.12.: Confusion matrix where the columns show the true label and the rows represent the predicted action

Here we evaluate the performance of the classifier on segmented actions in order to verify that GMMs are capable of representing our problem. To do this, we evaluate the method by Mainprice et al.[22] on our dataset. The only difference between this approach and our method proposed in section 3.2.2 is that we eliminate the window size and sum up the likelihoods of all poses of that action. Fig 4.12 shows the confusion matrix for the classification task. This evaluation was done with cross-validation, where the classifier is tested with the recordings of one person and trained on the remaining ones. The columns show the true label, while the rows represent the predicted action. The cells in the diagonal of the matrix display the amount of correct classified samples. Interestingly, even if Mainprice et al. explicitly tested this method only for actions that have a clear target, which is not the case for waiting, we can observe a high accuracy for this action. Furthermore, if we take a look at fig 4.9, we can see that the average time that the

user stays in this action is quite high and that it also has a high variance. Since there is no real goal that the user can accomplish in that time and he also has no influence on the progress of that action, there are a lot of ways to perform this action in a way that can be considered as normal.

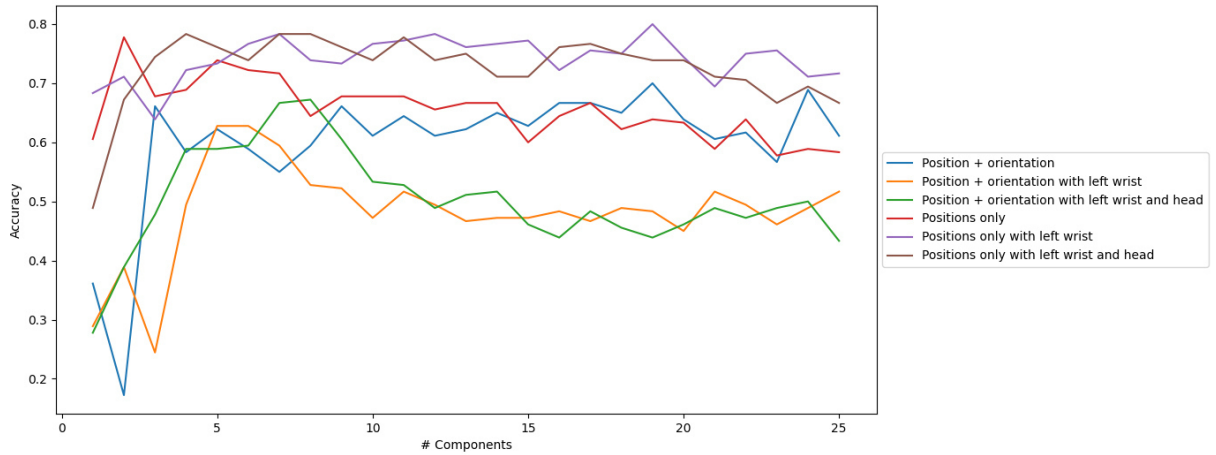


Figure 4.13.: Accuracy for different number of components and absolute features

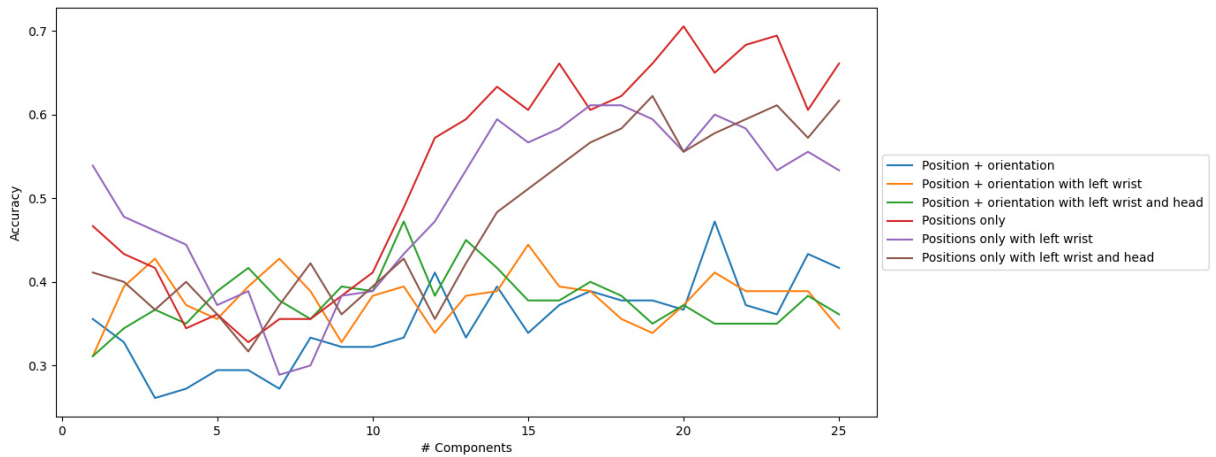


Figure 4.14.: Accuracy for different number of components and relative features

Figure 4.13 shows the accuracies for the different feature combinations with absolute positions and figure 4.14 with relative positions for different number of components. We can see that the usage of relative or absolute positions does not lead to an improvement in performance. The best result for the relative lies at 70%, while it is around 10 percentage points higher for the absolute features. Furthermore, as already mentioned above, we can see that all feature combinations that utilize orientations tend to perform weaker than the ones with positions only. Another important insight is that, as expected, most classifiers tend to perform better as the number of GMM components increases. However, this effect is even more

significant for the relative features. For the absolute features, the performance increases significantly at around three to five components and then remains quite stable, while for the relative features we can observe a very weak performance until around ten components. Overall, we obtain the best accuracy of 80% for the feature combination that uses the absolute positions of the pelvis, torso, and right and left wrist with a GMM that uses 19 components. For the following evaluation, we will use these features and the number of components. As already expected above, the combinations that only use the right wrist perform overall weaker than the others.

Since our method proposed in section 3.2.2 only uses the frames within a certain window instead of all frames in the past, we next want to verify that similar results can be achieved by only using the frames within the corresponding window. As the example in figure 4.15 shows, we can approximately reach the reference accuracy of 80% with window sizes of 125 frames with 75% accuracy. Since our dataset was collected with 30 FPS, a window size of 120 frames corresponds to 4 seconds. We observe that the prediction is very inaccurate for lower window sizes and increases significantly for higher sizes. It is important to mention that this evaluation was done on the segmented dataset. For the continuous dataset, this tendency might only hold until a certain window size. The reason for this is that the continuous dataset contains multiple different actions in one recording. That means if we choose a window size that is too large, the predictions around the timesteps where the user transitions from one action to another might be inaccurate since the poses from the previous action will dominate the window for a longer duration.

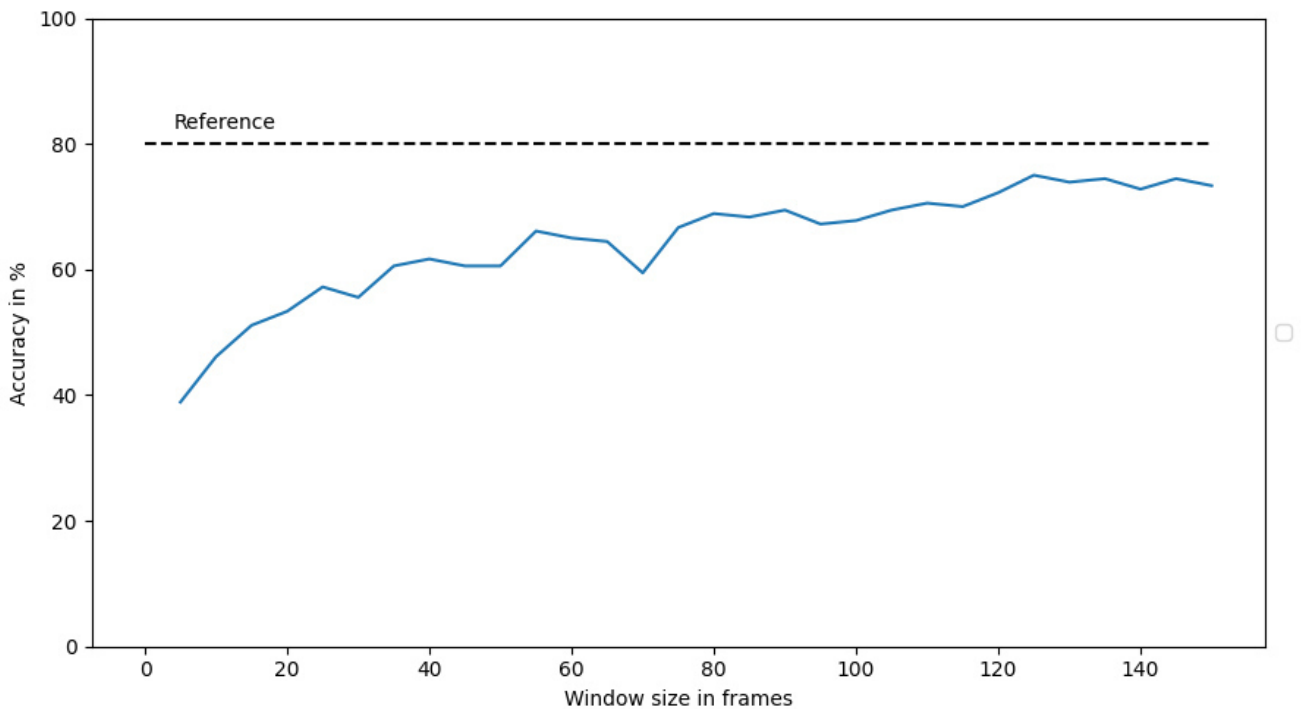


Figure 4.15.: Segmented action recognition for different window sizes: the horizontal line represents the reference accuracy which was achieved when using complete recordings for the prediction.

Performance per Recording

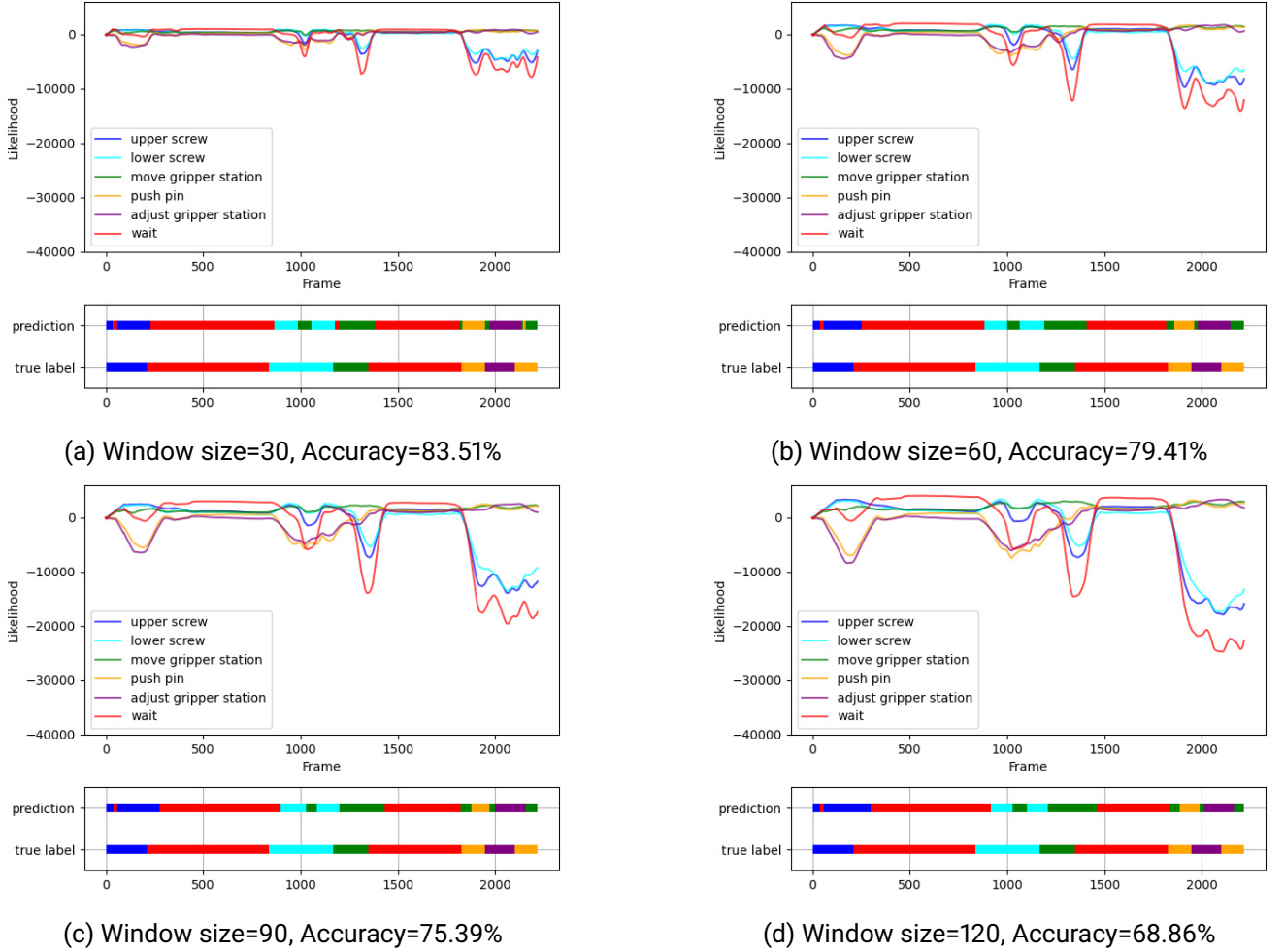


Figure 4.16.: Example: Likelihoods and predictions for one recording with different window sizes and frequency domain features

In this section we want to compare two methods: First, the method where we used frequency domain features and for the testing summed up the log-likelihoods of all sequential poses within a window, and second the method where we used time domain features. Both were tested with relative and absolute features. In contrast to the evaluation in the previous section, we evaluate these algorithms on the complete recordings, for continuous action recognition respectively. In the previous section, we already found out that the window-based approach is very inaccurate for low window sizes, and for increasingly window sizes performs only a bit weaker than when using all poses of an action. There we already assumed that this trend only holds until a certain window size since after transitions, poses from the previous action influence the new action longer if the window size is higher. Figure 4.16 illustrates this on one example with different window sizes and thus underlines this assumption. We observe that the sequence of predictions is approximately equal and thus the prediction looks similar for all window sizes. The main difference between these predictions lies in the timestep, at which an action is first predicted. For higher window sizes, the algorithm indeed needs more time to predict the correct action after a transition. As the plots

illustrate, this has a huge impact on accuracy. While in the segmented action recognition, the algorithm with a window size of 30 performed around 15 percentage points weaker than with a window size of 120, we can observe a reversed effect for the continuous action recognition for this chosen example. As shown in figure 4.9, one action in our training endures around 10 seconds on average. This means that for a window size of 120 frames, or 4 seconds respectively, 40% of the poses that belong to this action are influenced by previous poses. That is why we assume that this effect should be less significant for cases where the duration of an action is longer than in our case. The average accuracies for the complete dataset are shown in figure 4.17 and table 4.1. Figure 4.16 also illustrates that the likelihoods are smoother for larger window sizes. We will discuss this later in section 4.3.3 since this gives the impression that lower window sizes are more sensitive toward anomalies.

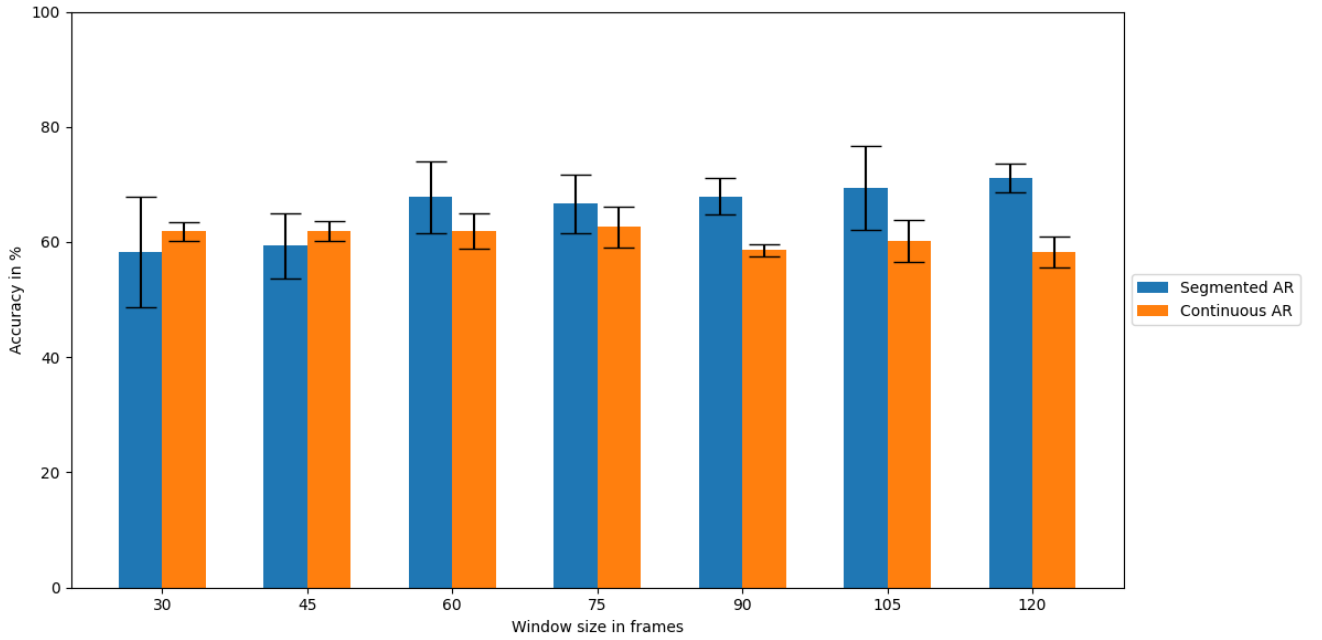


Figure 4.17.: Comparison of segmented- and continuous action recognition for different window sizes for frequency domain features

		Window size													
		30		45		60		75		90		105		120	
AR type		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Segmented		58.33	9.57	59.33	5.69	67.78	6.28	66.67	5.09	67.9	3.14	69.44	7.31	71.11	2.49
Continuous		61.85	1.65	61.9	1.69	61.92	3.13	62.59	3.52	58.58	1.08	60.12	3.63	58.31	2.7

Table 4.1.: Comparison of segmented- and continuous action recognition for different window sizes for frequency domain features

Next, we want to evaluate the method that utilizes time domain features and compare the results with the previous method which uses frequency domain features. Figure 4.18 shows the results for the same

example as figure 4.16. For this example, we can see that with time domain features, the window size is less important than with frequency domain features. Figure 4.19 and table ?? prove this assumption. It also shows that in general the time domain features outperform frequency domain features in terms of accuracy.

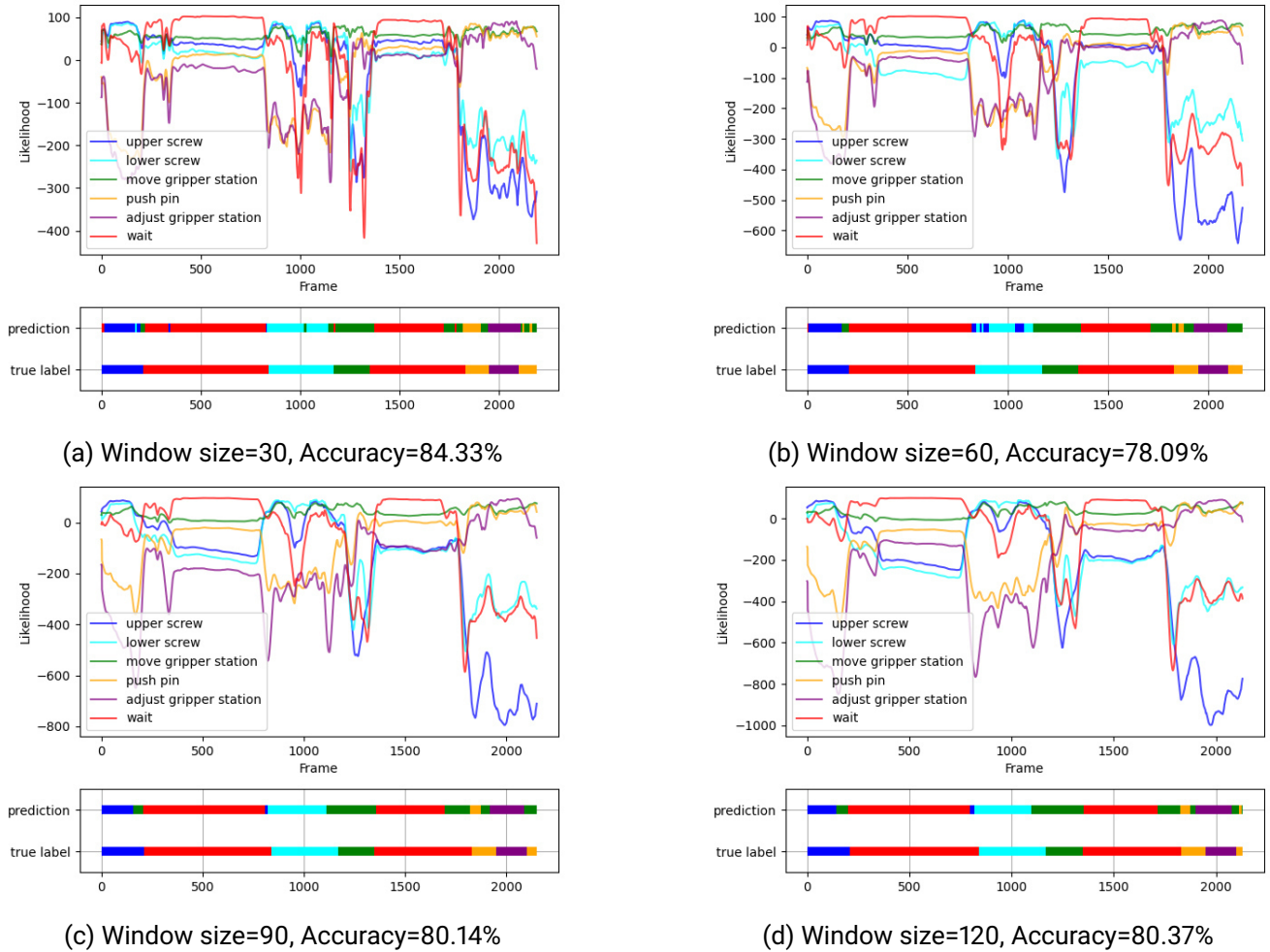


Figure 4.18.: Example: Likelihoods and predictions for one recording with different window sizes and time domain features

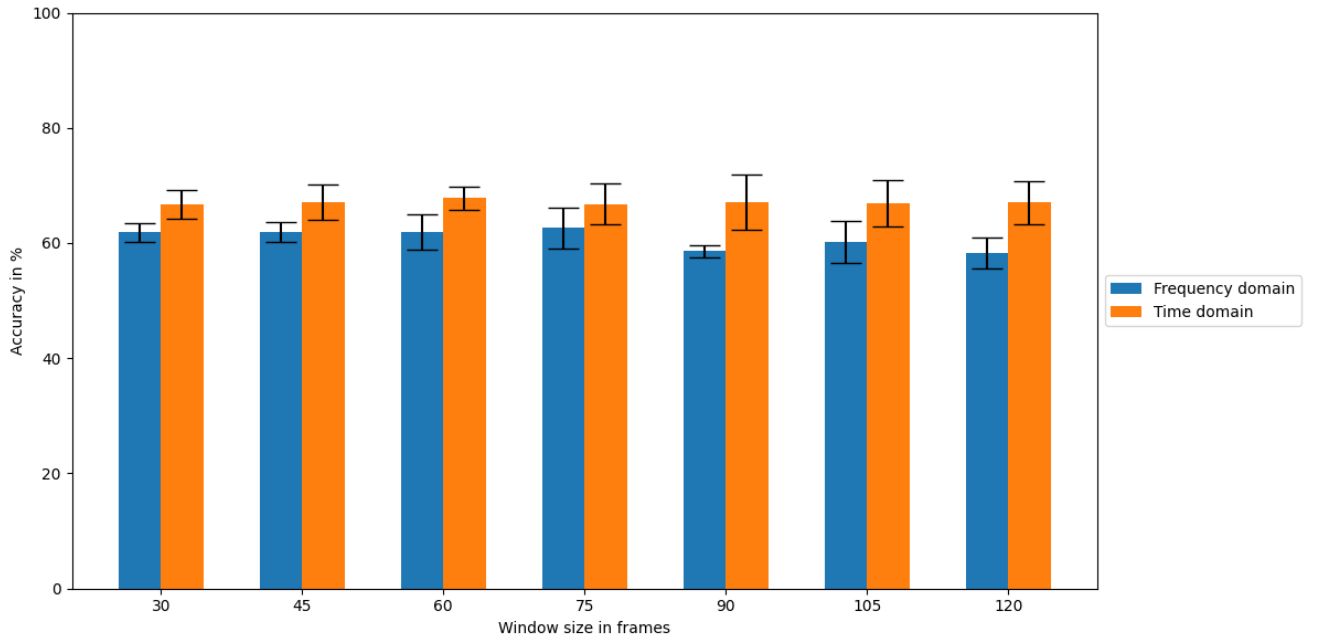


Figure 4.19.: Comparison of frequency- and time domain features for different window sizes

Window size														
30		45		60		75		90		105		120		
Features	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Frequency	61.85	1.65	61.9	1.69	61.92	3.13	62.59	3.52	58.58	1.08	60.12	3.63	58.31	2.7
Time	66.64	2.5	67.07	3.01	67.82	2.02	66.79	3.49	67.12	4.73	66.93	4.02	67.04	3.69

Table 4.2.: Comparison of frequency- and time domain features for different window sizes

4.3.3. Anomaly Detection in Motion Trajectories

In this section we will evaluate the actual context-independent anomaly detection. For this we first give some details about the anomalies that appeared during the recordings. We will then analyze for both methods mentioned in section 3.2.2, if the method is even capable of representing anomalies. That is done by taking a look at the maximum likelihood and p-values and finding an own suitable anomaly threshold for each of these recordings. Next, in order to evaluate if these approaches can be used in the real application, the analysis will be extended by finding one anomaly threshold for all recordings and analyzing the false positive and false negative rates for both methods.

One Threshold for each Recording

When we recorded the training data, we also recorded five trainings where the user showed anomalous behavior without provoking it. Table 4.3 shows the relevant information about these anomalies. We decided that anomalies three and four are crucial since they heavily prevent the user from progressing in the training.

Number	Description	Action	Time in sec.	Priority
1	Take gripper station not recognized	Move g. s.	62 - 64	low
2	Insert gripper station not recognized	Move g. s.	20 - 31	low
3	User dropped screwdriver	Upper screw	7 - 10	high
4	User dropped gripper station	Move g. s.	58 - 61	high
5	User was distracted by surrounding	Wait	44 - 47	low

Table 4.3.: Details about anomalies

The respective figures 4.20, 4.21, 4.22, 4.23 and 4.24 show the highest likelihood over all actions for each frame, which is normalized for each example for a better overview. The horizontal line was drawn at $y = \epsilon$ and is green for each normal frame and red for each anomalous frame. If the likelihood is below that line, the algorithm detects an anomaly. When evaluating these recordings on their own, we can find a suitable value for ϵ for all of the anomalies, so that we can separate the normal from the anomalous data. However, we can observe that there are a lot of minima besides the anomalous frames and that it is impossible to find an ϵ that does not lead to false alarms at all. After manually reviewing these recordings, we found out that some of those minima indeed occur for behavior that could also be labeled as low priority anomaly, for example when the user hesitates. However, when labeling the dataset, we decided that we do not want to consider this behavior anomalous. Another factor that led to such minima was the error-prone body tracking. As already mentioned in section 4.3.1, the robot arm was for example detected as part of the human sometimes.

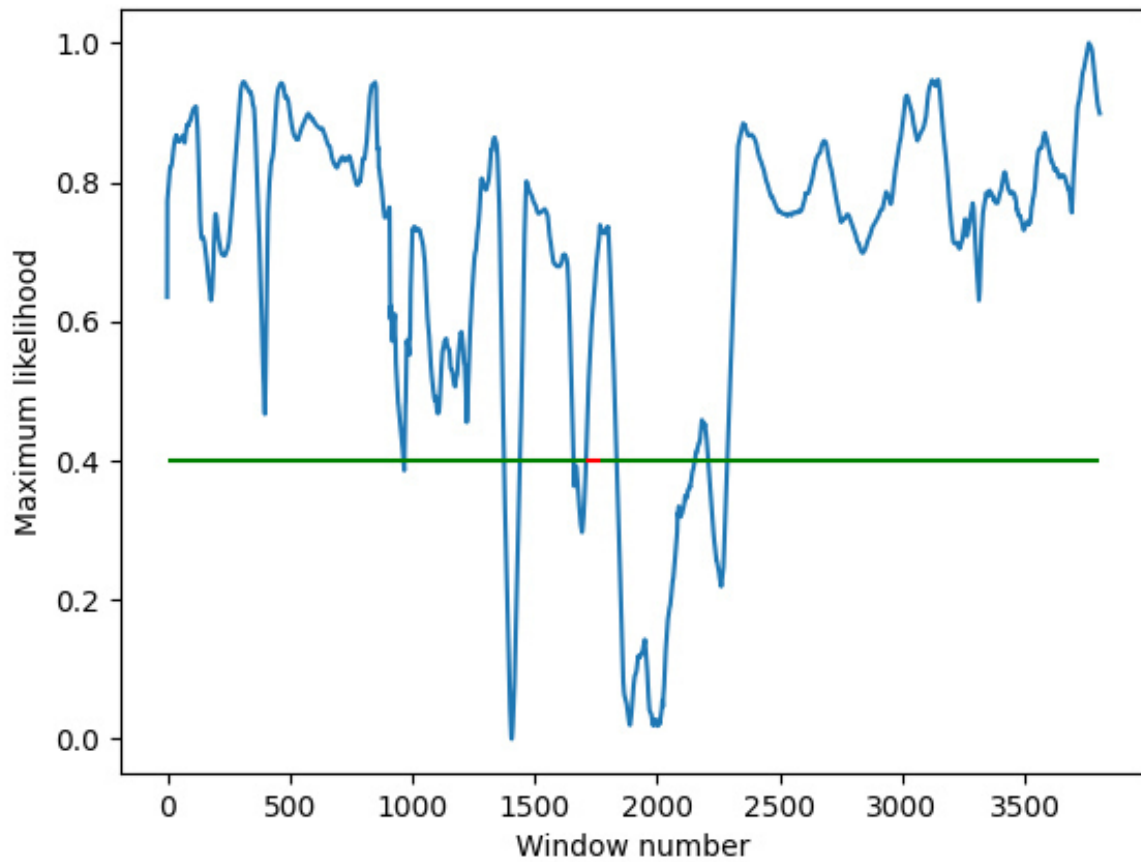


Figure 4.20.: Anomaly detection results for anomaly 1: the plot shows the maximum likelihood over all actions for every window. In this case, the window number equals the frame number since the prediction is done for every frame. The likelihoods were normalized. The green and red horizontal line displays the anomaly threshold at $y = \epsilon$ and is green if the true label for that frame is normal and red for anomalies. The algorithm detects an anomaly for the corresponding window if the likelihood is below that line.

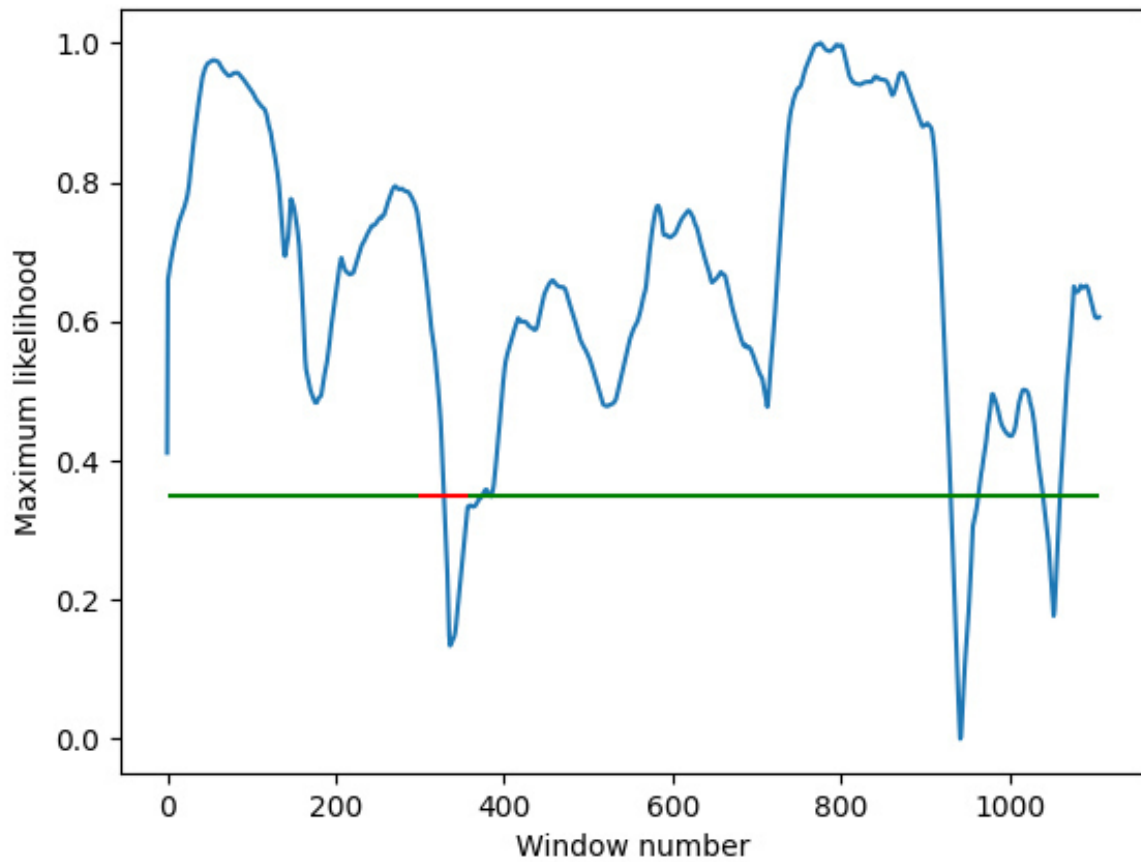


Figure 4.21.: Anomaly detection results for anomaly 2: the plot shows the maximum likelihood over all actions for every window. In this case, the window number equals the frame number since the prediction is done for every frame. The likelihoods were normalized. The green and red horizontal line displays the anomaly threshold at $y = \epsilon$ and is green if the true label for that frame is normal and red for anomalies. The algorithm detects an anomaly for the corresponding window if the likelihood is below that line.

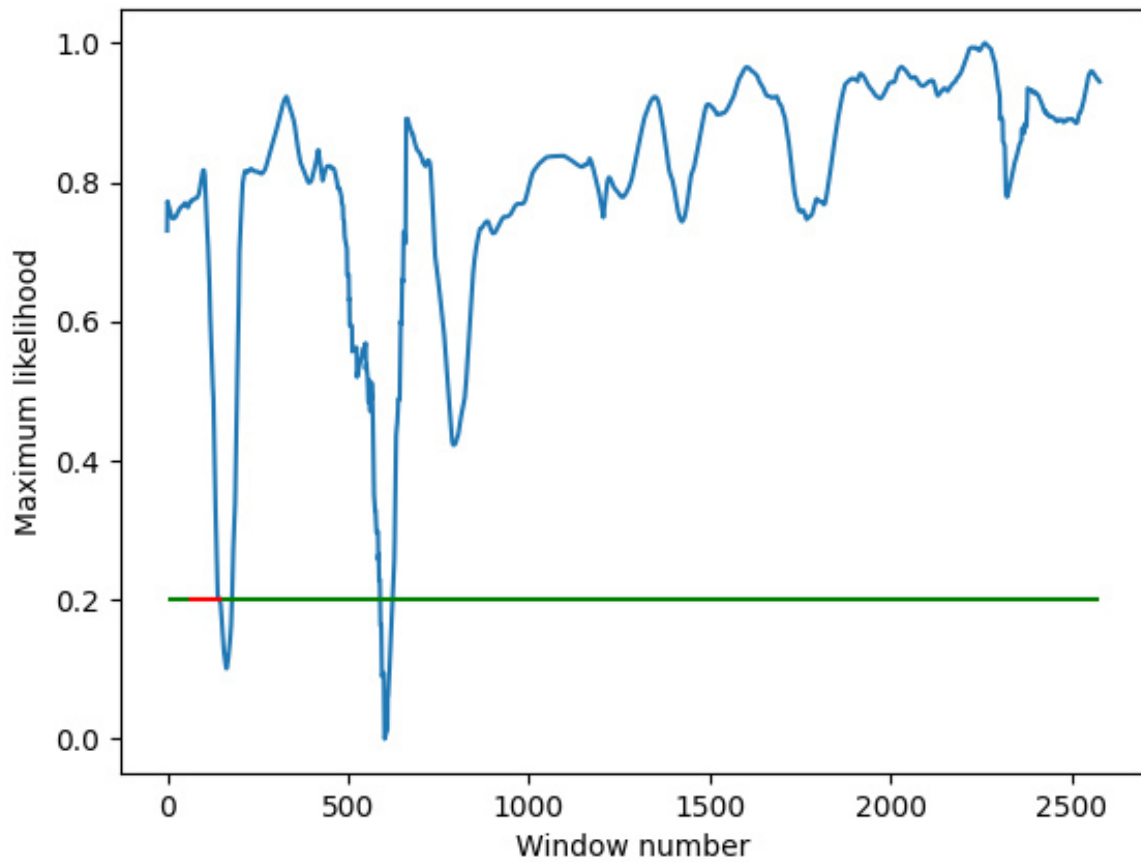


Figure 4.22.: Anomaly detection results for anomaly 3: the plot shows the maximum likelihood over all actions for every window. In this case, the window number equals the frame number since the prediction is done for every frame. The likelihoods were normalized. The green and red horizontal line displays the anomaly threshold at $y = \epsilon$ and is green if the true label for that frame is normal and red for anomalies. The algorithm detects an anomaly for the corresponding window if the likelihood is below that line.

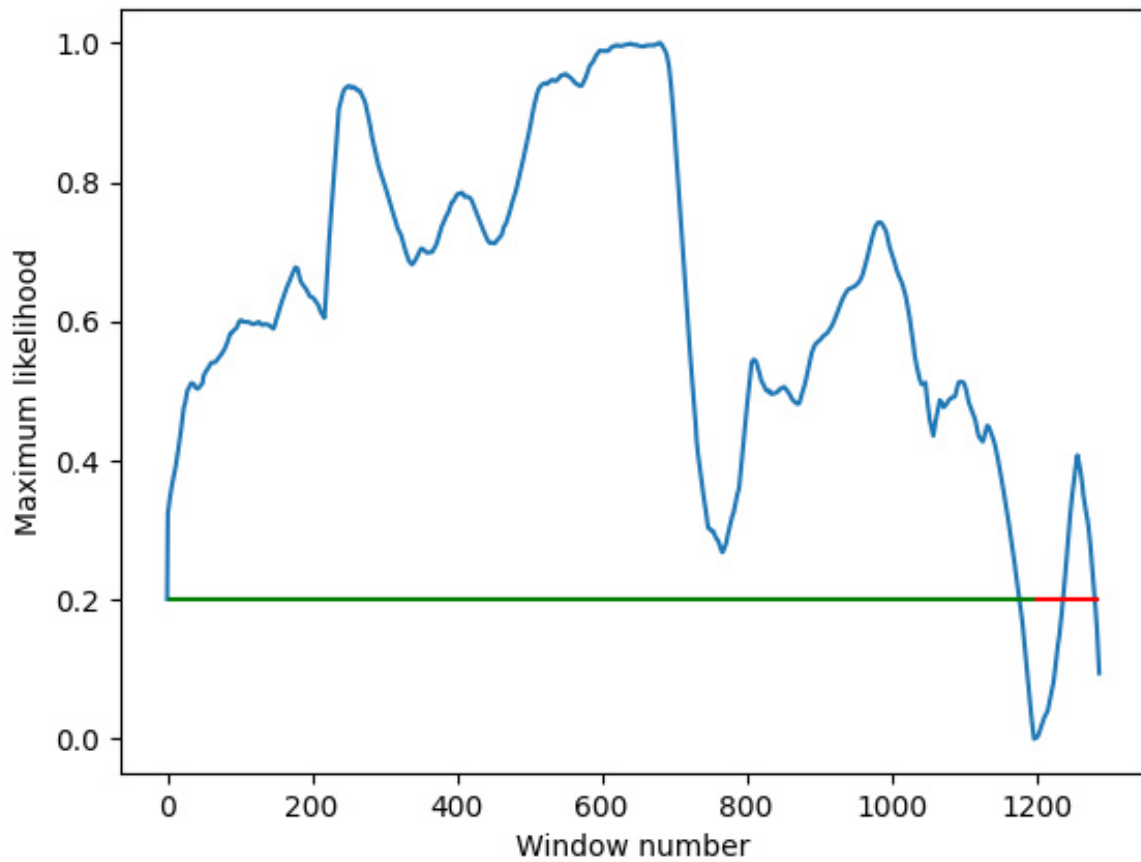


Figure 4.23.: Anomaly detection results for anomaly 4: the plot shows the maximum likelihood over all actions for every window. In this case, the window number equals the frame number since the prediction is done for every frame. The likelihoods were normalized. The green and red horizontal line displays the anomaly threshold at $y = \epsilon$ and is green if the true label for that frame is normal and red for anomalies. The algorithm detects an anomaly for the corresponding window if the likelihood is below that line.

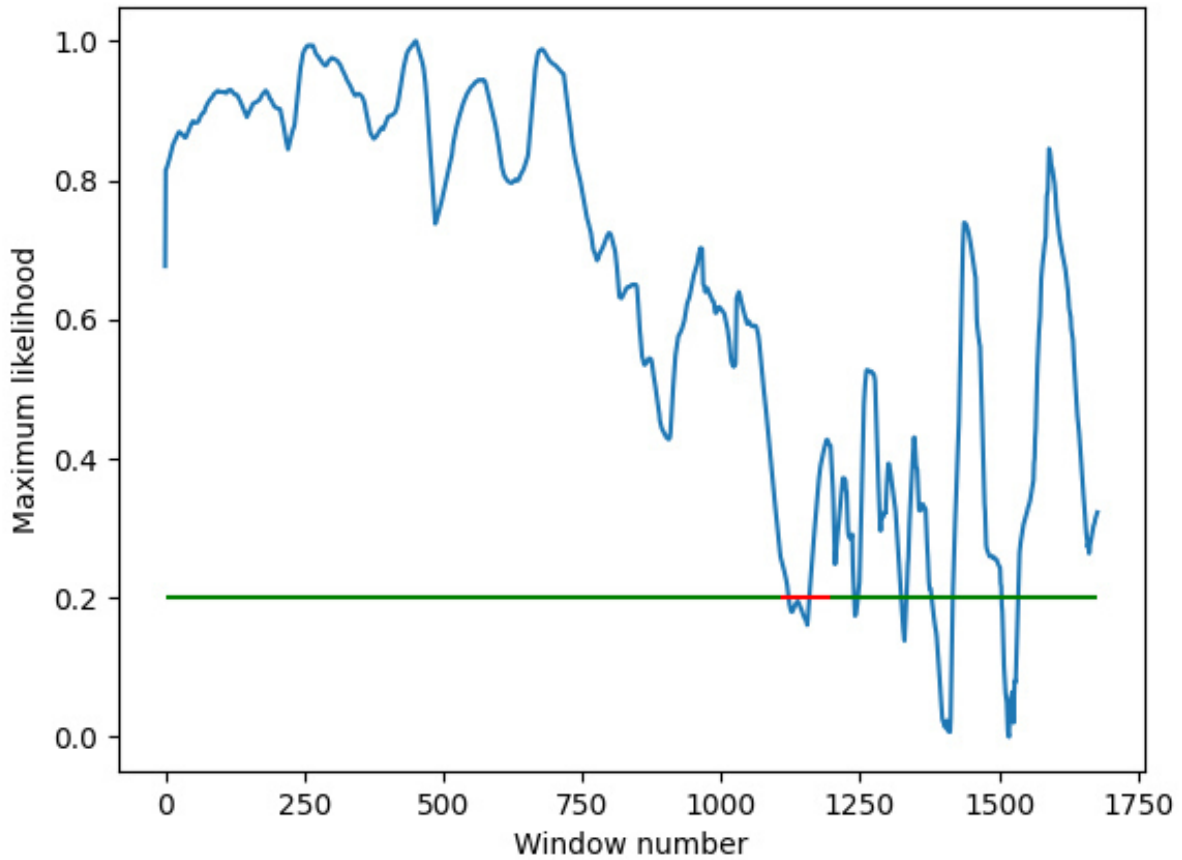


Figure 4.24.: Anomaly detection results for anomaly 5: the plot shows the maximum likelihood over all actions for every window. In this case, the window number equals the frame number since the prediction is done for every frame. The likelihoods were normalized. The green and red horizontal line displays the anomaly threshold at $y = \epsilon$ and is green if the true label for that frame is normal and red for anomalies. The algorithm detects an anomaly for the corresponding window if the likelihood is below that line.)

One Threshold for all Recordings

For the evaluation of the methods in the real scenario, we want to know how well the algorithms perform if we have to choose one value for ϵ for the whole dataset. Then we want to compare the false positive rate (also called false alarm rate) and false positive rate of the different algorithms. The false positive rate is calculated by dividing the number of samples that are labeled as normal and classified as anomalous by the amount of all samples that are labeled as normal. The false negative rate on the other hand is given by the division of the number of samples that are labeled as anomalous and classified as normal by the amount of all samples that are labeled as an anomaly. The rates for the likelihood method are shown in figure 4.25 with a window size of 30 and figure 4.26 with a window size of 120. In that case, we used the frequency domain features again as they returned more meaningful results and thus serve as a better baseline than

time domain features. As we can see, the false alarm rate stays very high until ϵ reaches 0.98 and then drops significantly. The rate where the false positive- and the false negative rate intersect lies at 0.18. Interestingly, this holds for both tested window sizes. However, for window size 120 the false negative rate drops faster than for window size 30. As shown in the plot, there is a tradeoff between the false positive- and the false negative rate. Choosing an ϵ with a high false positive rate leads to a less efficient robot movement since the robot moves with a decreased velocity due to the high amount of detected anomalies. For higher false negative rates, on the other hand, the algorithms are more likely to miss anomalous human behavior, which makes the application more dangerous. In cases with higher safety requirements for the human-robot interaction, we would choose a higher ϵ and accept a higher false alarm rate to keep the false negative rate as low as possible.

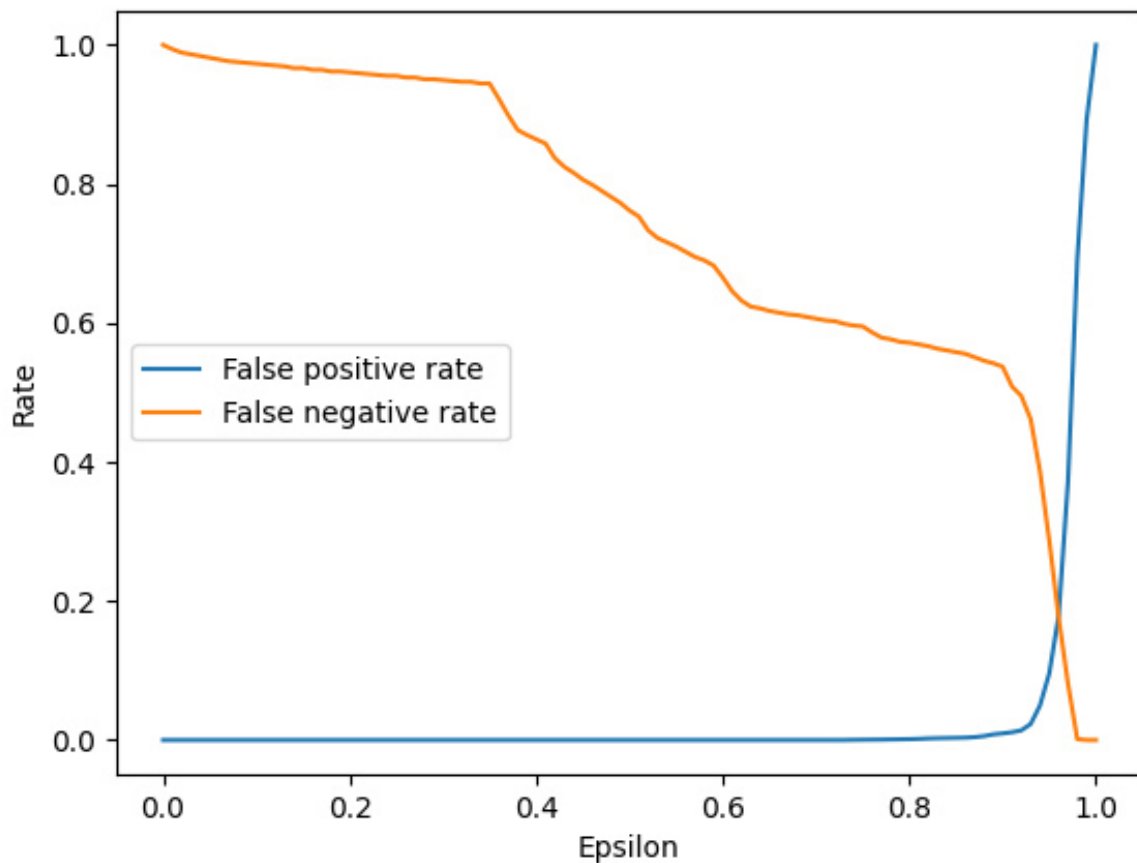


Figure 4.25.: False negative- and false positive rate over all recordings for the likelihood method

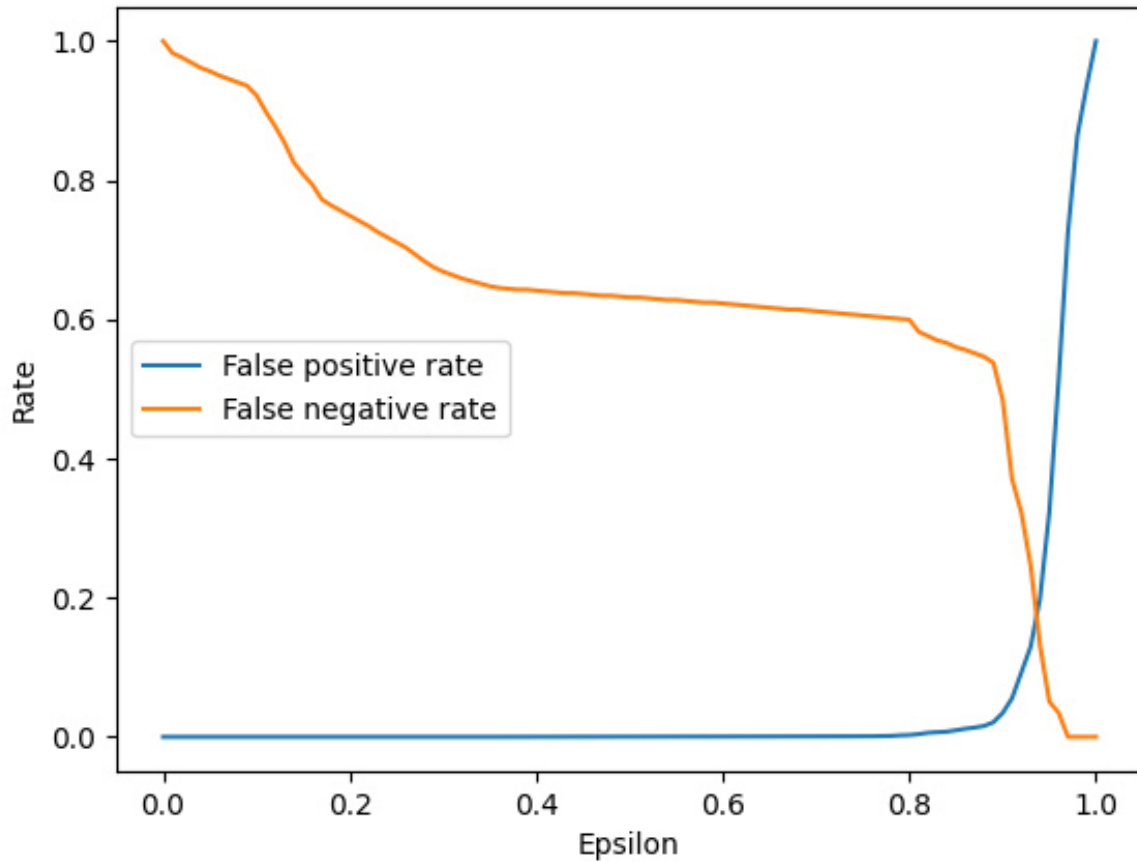


Figure 4.26.: False negative- and false positive rate over all recordings for the likelihood method

The results for the CAD method are shown in figure 4.27. We can observe that the intersection between the false negative and the false positive rate lies at $\epsilon = 0.01$ which is way earlier than for the likelihood method. However, the rate at which they intersect lies at 0.65 and thus is much higher compared to the baseline method. We can also observe that the false positive rate immediately increases to approximately 1, while still detecting false negatives. For the real application, the plot shows that the robot would either work with a decreased velocity for most of the time for $\epsilon > 0.01$, even if there are no actual anomalies, or otherwise not detect any anomalies. So we can say, that the CAD approach with the NCM proposed in section 3.2.2 is not suitable for this dataset and application. It is also important to mention that, as opposed to the likelihood method, we work with single poses instead of windows of poses here. This means that single noisy data points (see section 4.3.1) have a much higher impact here than on the baseline method because we do not care about the surrounding data points.

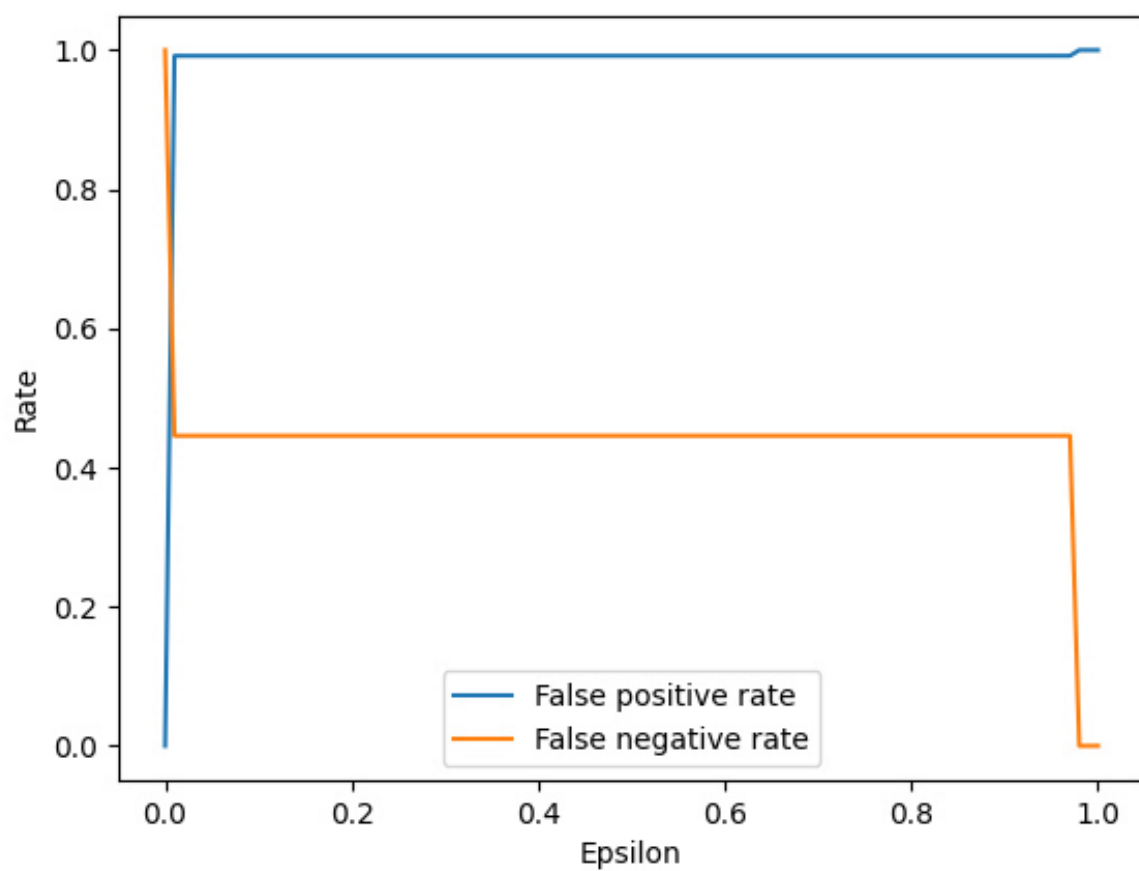


Figure 4.27.: False negative- and false positive rate over all recordings for CAD

5. Discussion and Outlook

In this chapter, we want to evaluate our approach, especially, concerning the following research questions:

1. Is our approach suitable to detect anomalies in human behavior?
2. Is the robot able to perform its task as quickly as before, given that there is no actual anomaly?

Furthermore, we give ideas for further work as well as a conclusion about this thesis and the results.

5.1. Framework Performance Discussion

Our action recognition could not achieve similar accuracies as was given in the reference papers (86.92% for time domain features [36], 92% for frequency domain features [22]). For the frequency domain features, we achieved 80% accuracy when using all poses of one action of a segmented recording which is 12% percentage points lower than in the reference paper. When extending this method to a window-based approach for continuous action recognition, the accuracy decreased even more to 62.59% for the best window size. With a maximum of 67.72%, the time domain features performed better than the frequency domain features but was still around 19 percentage points weaker than in the reference paper. The reason for this might be the noisy dataset (see section 4.3.1) or a higher complexity of the problem compared to the datasets in the reference papers since there are many ways to correctly complete an action.

For anomaly detection, we could observe that the method which uses the maximum achieved likelihood over all actions as an indication for anomalies already achieved good results. That means we can select an anomaly threshold parameter ϵ , such that we achieve a tradeoff between the false positive- and the false negative rate which can be tested in a real application. The CAD method, on the other hand, could not hold up to this. The reason for this is that our selected nonconformity measure is not capable of representing the different actions such that the NCM is different for all actions. That means the robot will either always act with a decreased velocity due to detected anomalies or not detect any anomalies.

5.2. Limitations

One problem with this approach is that anomaly detection algorithms assume that anomalies appear much less frequently than normal data. However, there are way more and significantly diverse opportunities to perform an action than the training data can cover. For example, when loosening the screw, it can be approached from many different directions. The action itself should then still be considered normal, but our algorithm will detect this as an anomaly since it has not seen this trajectory before. This means that with our approach, as opposed to the assumption of anomaly detection algorithms, there are many more

ways to cause an anomaly than to perform a normal action. One solution that might decrease this impact might be to include more guidance during the training so that the user is more likely to perform not only a certain action but to perform it also in an expected way. Depending on the context where the training is used, online learning could also be included to train the algorithm while it is used. In this case, all normal data will be added to the training data. This requires that a second person observes the training and is notified when the algorithm detects an anomaly. The observer can then confirm the anomaly or not. In case he does not confirm it, the algorithm can add that trajectory to the training set.

In general one of the biggest limitations, we had to deal with was the amount of training data. Other methods like neural networks are known to work well for human action recognition and thus could also be used for this task. However, since they usually also require a lot of training data, it would exceed the scope of this thesis to record such a huge amount of training data. For future work, it might still be interesting to use a neural network for anomaly detection in human motion trajectories.

5.3. Outlook

The CAD algorithm did not achieve good results for our case. For future work, one could evaluate if there are more suitable NCMs that lead to a better performance of the CAD algorithm. For example, as mentioned in section 2.4.3, Falkmar et al. introduced the sequential Hausdorff nearest neighbor conformal anomaly detector[18] and the sliding window local outlier conformal anomaly detector [17], which are designed for incomplete trajectories. However, we did not evaluate this approach in our work for several reasons. For example, the algorithms were evaluated for trajectories in a 3D space and the used NCMs are more complex than the ones we tested for our approach. We first wanted to evaluate, whether an algorithm with such a high complexity is needed for our case. Furthermore, there are a lot of factors that need to be taken into consideration when applying the algorithms of those papers to our problem, which would exceed the scope of this thesis.

Our approach aims to detect anomalies in human motion after they happened. However, another interesting question would be if anomalies can be predicted beforehand. To do this, one might predict the future human motion trajectory based on the current trajectory information and then apply anomaly detection algorithms to the predicted trajectory. For the prediction, one could for instance use the next step of Jim Mainprice et al.[22], where they predict the human motion by performing gaussian mixture regression.

In this work, we tested the algorithm in a scenario where the robot can avoid collisions by generating a collision plane and staying behind this plane. However, the robot could work much more efficiently if it moved farther through the human workspace. This requires a replanning algorithm for avoiding collisions. It would be interesting to combine this approach with the trajectories generated by replanning algorithms so that in case of anomalies the robot slows down while executing the replanned trajectory. Since this can decrease the waiting times, it would be interesting to perform a user study to see how the robot moving farther through the human workspace influences the well-being of the user and the overall experience with the simulation.

5.4. Conclusion

In this work, we developed a method for anomaly detection in human motion trajectories to allow for safer human-robot collaboration by slowing down the robot in case of anomalies. We noticed that there are two types of anomalies: Anomalies on the action level, where the user performs a valid action at the wrong time, and anomalies within the action, where the behavior of the user can not be identified with sufficient certainty. To solve this problem, we created a method based on continuous action recognition and conformal anomaly detection. Our proposed framework first recognizes the current action to detect action level anomalies and then also checks if anomalies occur within that action. We tested this method with a VR training where the user has to maintain a gripper station and a sawyer cobot provides haptic feedback for certain tasks. We collected the data ourselves using an Azure Kinect and ended up with five successful recordings of five different persons, as well as six recordings where anomalies occurred. Our best approach for continuous action recognition that uses GMMs and time domain features should be improved before testing it in the real application. The anomaly detection which is based on the returned likelihoods returns good results so that this part could be evaluated on the real robot.

Bibliography

- [1] Janis Arents et al. “Human–Robot Collaboration Trends and Safety Aspects: A Systematic Review”. In: *Journal of Sensor and Actuator Networks* 10.3 (2021), p. 48.
- [2] Reuben M Aronson and Henny Admoni. “Gaze for error detection during human-robot shared manipulation”. In: *Fundamentals of Joint Action workshop, Robotics: Science and Systems*. 2018.
- [3] Donald J Berndt and James Clifford. “Using dynamic time warping to find patterns in time series.” In: *KDD workshop*. Vol. 10. 16. Seattle, WA, USA: 1994, pp. 359–370.
- [4] Sylvain Calinon, Florent Guenter, and Aude Billard. “On learning, representing, and generalizing a task in a humanoid robot”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298.
- [5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.
- [6] Y Choi, AS Ralhan, and S Ko. “A study on machine learning algorithms for fall detection and movement classification”. In: *2011 International Conference on Information Science and Applications*. IEEE. 2011, pp. 1–8.
- [7] Dmitry Devetyarov and Ilia Nouretdinov. “Prediction with confidence based on a random forest classifier”. In: *IFIP international conference on artificial intelligence applications and innovations*. Springer. 2010, pp. 37–44.
- [8] Riccardo Gervasi, Luca Mastrogiacomo, and Fiorenzo Franceschini. “A conceptual framework to evaluate human-robot collaboration”. In: *The International Journal of Advanced Manufacturing Technology* 108.3 (2020), pp. 841–865.
- [9] Yuejun Guo and Anton Bardera. “SHNN-CAD+: An improvement on SHNN-CAD for adaptive online trajectory anomaly detection”. In: *Sensors* 19.1 (2018), p. 84.
- [10] Weiming Hu et al. “A system for learning statistical motion patterns”. In: *IEEE transactions on pattern analysis and machine intelligence* 28.9 (2006), pp. 1450–1464.
- [11] Mrinal Kalakrishnan et al. “Learning objective functions for manipulation”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1331–1336.
- [12] Mrinal Kalakrishnan et al. “STOMP: Stochastic trajectory optimization for motion planning”. In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.
- [13] Eamonn Keogh, Jessica Lin, and Ada Fu. “Hot sax: Efficiently finding the most unusual time series subsequence”. In: *Fifth IEEE International Conference on Data Mining (ICDM’05)*. Ieee. 2005, 8–pp.
- [14] Ben Kirsche. “Simulation von haptischem Feedback in VR mittels eines Cobots”. MA thesis. University Bonn-Rhein-Sieg, July 2021.
- [15] Rikard Laxhammar. “Conformal anomaly detection: Detecting abnormal trajectories in surveillance applications”. PhD thesis. University of Skövde, 2014.

-
- [16] Rikard Laxhammar and Göran Falkman. "Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories". In: *Annals of Mathematics and Artificial Intelligence* 74.1 (2015), pp. 67–94.
- [17] Rikard Laxhammar and Göran Falkman. "Online detection of anomalous sub-trajectories: A sliding window approach based on conformal anomaly detection and local outlier factor". In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2012, pp. 192–202.
- [18] Rikard Laxhammar and Göran Falkman. "Online learning and sequential anomaly detection in trajectories". In: *IEEE transactions on pattern analysis and machine intelligence* 36.6 (2013), pp. 1158–1173.
- [19] Rikard Laxhammar and Göran Falkman. "Sequential conformal anomaly detection in trajectories based on hausdorff distance". In: *14th International Conference on Information Fusion*. IEEE. 2011, pp. 1–8.
- [20] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. "Trajectory outlier detection: A partition-and-detect framework". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 140–149.
- [21] Hongyi Liu and Lihui Wang. "Human motion prediction for human-robot collaboration". In: *Journal of Manufacturing Systems* 44 (2017), pp. 287–294.
- [22] Jim Mainprice and Dmitry Berenson. "Human-robot collaborative manipulation planning using early prediction of human motion". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 299–306.
- [23] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. "Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 885–892.
- [24] Eloise Matheson et al. "Human-robot collaboration in manufacturing applications: a review". In: *Robotics* 8.4 (2019), p. 100.
- [25] George Michalos et al. "Design considerations for safe human-robot collaborative workplaces". In: *Procedia CIRP* 37 (2015), pp. 248–253.
- [26] Carlos Morato et al. "Toward safe human robot collaboration by using multiple kinects based real-time human tracking". In: *Journal of Computing and Information Science in Engineering* 14.1 (2014).
- [27] Brendan T Morris and Mohan M Trivedi. "Learning and classification of trajectories in dynamic scenes: A general framework for live video analysis". In: *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*. IEEE. 2008, pp. 154–161.
- [28] Meinard Müller. "Dynamic time warping". In: *Information retrieval for music and motion* (2007), pp. 69–84.
- [29] Sebastian Nowozin and Jamie Shotton. "Action points: A representation for low-latency online human action recognition". In: *Microsoft Research Cambridge, Tech. Rep. MSR-TR-2012-68* (2012).
- [30] Jonathan Owens and Andrew Hunter. "Application of the self-organising map to trajectory classification". In: *Proceedings Third IEEE International Workshop on Visual Surveillance*. IEEE. 2000, pp. 77–83.
- [31] Xinlong Pan et al. "Online detection of anomaly behaviors based on multidimensional trajectories". In: *Information Fusion* 58 (2020), pp. 40–51.

-
- [32] Harris Papadopoulos, Volodya Vovk, and Alex Gammerman. "Conformal prediction with neural networks". In: *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Vol. 2. IEEE. 2007, pp. 388–395.
- [33] Claudio Picciarelli, Christian Micheloni, and Gian Luca Foresti. "Trajectory-based anomalous event detection". In: *IEEE Transactions on Circuits and Systems for video Technology* 18.11 (2008), pp. 1544–1554.
- [34] Carl Rasmussen. "The infinite Gaussian mixture model". In: *Advances in neural information processing systems* 12 (1999).
- [35] Pavel Senin. "Dynamic time warping algorithm review". In: *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855.1-23 (2008), p. 40.
- [36] Prateek Srivastava and WC Wong. "Hierarchical human activity recognition using GMM". In: *AMBI-ENT 2012: The Second International Conference on Ambient Computing, Applications, Services and Technologies*. 2012, pp. 32–37.
- [37] Hsi Guang Sung. *Gaussian mixture regression and classification*. Rice University, 2004.
- [38] Giorgio Tomasi, Frans Van Den Berg, and Claus Andersson. "Correlation optimized warping and dynamic time warping as preprocessing methods for chromatographic data". In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 18.5 (2004), pp. 231–241.
- [39] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.
- [40] Kun Xia, Jianguang Huang, and Hanyu Wang. "LSTM-CNN architecture for human activity recognition". In: *IEEE Access* 8 (2020), pp. 56855–56866.
- [41] Salisu Wada Yahaya, Ahmad Lotfi, and Mufti Mahmud. "Towards a data-driven adaptive anomaly detection system for human activity". In: *Pattern Recognition Letters* 145 (2021), pp. 200–207.
- [42] Salisu Wada Yahaya et al. "Gesture recognition intermediary robot for abnormality detection in human activities". In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2019, pp. 1415–1421.
- [43] Guangming Zhu et al. "An online continuous human action recognition algorithm based on the Kinect sensor". In: *Sensors* 16.2 (2016), p. 161.



A. Some Appendix

Use letters instead of numbers for the chapters.