Wearable Activity Recognition using Invariant Signal Segments

Thesis approved by the Department of Computer Science
Technical University of Kaiserslautern
for the degree of
Doctor of Engineering (Dr.-Ing.)

presented by
Matthias Johannes Michael Kreil

Date of Defense: 08.11.2018
Dean: Prof. Dr. Stefan Deßloch
Reviewer: Prof. Dr. Paul Lukowicz
Reviewer: Prof. Dr. Bernhard Sick

D 386

Abstract

Wearable activity recognition aims to identify and assess human activities with the help of computer systems by evaluating signals of sensors which can be attached to the human body. This provides us with valuable information in several areas: in health care, e.g. fluid and food intake monitoring; in sports, e.g. training support and monitoring; in entertainment, e.g. human-computer interface using body movements; in industrial scenarios, e.g. computer support for detected work tasks. Several challenges exist for wearable activity recognition: a large number of nonrelevant activities (null class), the evaluation of large numbers of sensor signals (curse of dimensionality), ambiguity of sensor signals compared to the activities and finally the high variability of human activity in general.

This thesis develops a new activity recognition strategy, called *invariants classification*, which addresses these challenges, especially the variability in human activities. The core idea is that often even highly variable actions include short, more or less invariant sub-actions which are due to hard physical constraints. If someone opens a door, the movement of the hand to the door handle is not fixed. However the door handle has to be pushed to open the door. The *invariants classification* algorithm is structured in four phases: segmentation, invariant identification, classification, and spotting. The segmentation divides the continuous sensor data stream into meaningful parts, which are related to sub-activities. Our segmentation strategy uses the zero crossings of the central difference quotient of the sensor signals, as segment borders. The invariant identification finds the invariant sub-activities by means of clustering and a selection strategy dependent on certain features. The classification identifies the segments of a specific activity class, using models generated from the invariant sub-activities. The models include the invariant sub-activity signal and features calculated on sensor signals related to the sub-activity. In the spotting, the classified segments are used to find the entire activity class instances in the continuous sensor data stream. For this purpose, we use the position of the invariant sub-activity in the related activity class instance for the estimation of the borders of the activity instances.

In this thesis, we show that our new activity recognition strategy, built on invariant sub-activities, is beneficial. We tested it on three human activity datasets with wearable inertial measurement units (IMU). Compared to previous publications on the same datasets we got improvement in the activity recognition in several classes, some with a large margin. Our segmentation achieves a sensible method to separate the sensor data in relation to the underlying activities. Relying on sub-activities makes us independent from imprecise labels on the training data. After the identification of invariant sub-activities, we calculate a value called *cluster precision* for each sensor signal and each class activity. This tells us which classes can be easily classified and which sensor channels support the classification best. Finally, in the training for each activity class, our algorithm selects suitable signal channels with invariant sub-activities on different points in time and with different length. This makes our strategy a multi-dimensional asynchronous motif detection with variable motif length.

Zusammenfassung

Wearable Activity Recognition zielt darauf ab, menschliche Aktivitäten mithilfe von Computersystemen zu identifizieren und zu bewerten, indem Signale von am menschlichen Körper angebrachten Sensoren ausgewertet werden. Dies liefert wertvolle Informationen in verschiedenen Bereichen: in der Gesundheitsfürsorge, z.B. Überwachung der Flüssigkeits- und Nahrungsaufnahme; beim Sport, z.B. Trainingsunterstützung und -überwachung; in der Unterhaltung, z.B. Mensch-Computer-Schnittstelle mit Körperbewegungen; in industriellen Szenarien, z.B. Computerunterstützung für erkannte Arbeitsaufgaben. Es gibt mehrere Herausforderungen für Wearable Activity Recognition: eine große Anzahl nicht relevanter Aktivitäten (Nullklasse), die Auswertung einer großen Anzahl von Sensorsignalen (Fluch der Dimensionalität), die Mehrdeutigkeit der Sensorsignale im Vergleich zu den Aktivitäten und schließlich die hohe Variabilität der menschlichen Aktivität im Allgemeinen.

In dieser Arbeit wird eine neue Aktivitätserkennungsstrategie entwickelt, genannt *Invariants Classification*, die diese Herausforderungen angeht, insbesondere die Variabilität menschlicher Aktivitäten. Die Kernidee ist, dass oft selbst sehr variable Aktivitäten, kurze, mehr oder weniger invariante Unteraktionen enthalten, die auf harte physikalische Bedingungen zurückzuführen sind. Wenn jemand eine Tür öffnet, ist die Bewegung der Hand zum Türgriff nicht festgelegt. Am Ende muss jedoch der Türgriff gedrückt werden, um die Tür zu öffnen. Der *Invariants Classification* Algorithmus ist in vier Phasen unterteilt: Segmentierung, Invarianten-Identifikation, Klassifizierung und Spotting. Die Segmentierung unterteilt den kontinuierlichen Sensordatenstrom in sinnvolle Teile, die sich auf Subaktivitäten beziehen. Unsere Segmentierungsstrategie verwendet die Nulldurchgänge des zentralen Differenzquotienten der Sensorsignale als Segmentgrenzen. Die Invarianten-Identifikation ermittelt die invarianten Subaktivitäten durch Clustering und eine Auswahlstrategie, die von bestimmten Features abhängt. Die Klassifizierung identifiziert die Segmente einer bestimmten Aktivitätsklasse anhand von Modellen, die aus den invarianten Subaktivitäten generiert werden. Die Modelle umfassen das invariante Subaktivitätssignal und Features, die aus Sensorsignalen berechnet werden, die mit der Subaktivität in Zusammenhang stehen. Beim Spotting werden die klassifizierten Segmente verwendet, um die gesamten Aktivitätsklasseninstanzen im kontinuierlichen Sensordatenstrom zu finden. Zu diesem Zweck verwenden wir die Position der invarianten Subaktivität in der zugehörigen Aktivitätsklasseninstanz, um die Grenzen der Aktivitätsinstanzen abzuschätzen.

In dieser Arbeit zeigen wir, dass unsere neue Aktivitätserkennungsstrategie, die auf invarianten Subaktivitäten basiert, von Vorteil ist. Wir haben sie an drei menschlichen Aktivitätsdatensätzen mit Wearable Inertial Measurement Units (IMU) getestet. Im Vergleich zu früheren Veröffentlichungen zu den gleichen Datensätzen konnten wir die Aktivitätserkennung in mehreren Klassen verbessern, einige davon mit großem Abstand. Unsere Segmentierung bietet eine sinnvolle Methode zur Trennung der Sensordaten in Bezug auf die zugrunde liegenden Aktivitäten. Wenn wir uns auf Subaktivitäten verlassen, sind wir unabhängig von ungenauen Kennzeichnungen der Trainingsdaten. Nach der Identifizierung von invarianten Subaktivitäten berechnen wir für jedes Sensorsignal und jede Klassenaktivität einen als *Cluster Precision* bezeichneten Wert. Dieser zeigt, welche Klassen einfach klassifiziert werden können und welche Sensorkanäle die Klassifizierung am besten unterstützen. Schließlich wählt unser Algorithmus im Training für jede Aktivitätsklasse geeignete Signalkanäle mit invarianten Subaktivitäten zu unterschiedlichen Zeitpunkten und mit unterschiedlicher Länge aus. Dies macht unsere Strategie zu einer mehrdimensionalen, asynchronen Motiverkennung mit variabler Motivlänge.

Contents

1 Introduction **1**

1.1 Wearable Activity Recognition 2
1.2 Challenges 3
1.3 Thesis Objective 4
1.4 Related Work 4
    1.4.1 Taxonomy of Human Activity 4
    1.4.2 Activity Recognition with Accelerometers 5
    1.4.3 Other Sensor Modalities 6
    1.4.4 Types of Scenarios 7
    1.4.5 Motif Detection 9
1.5 Contributions 10
1.6 Thesis Overview 12

2 Basic idea **15**

2.1 Introduction 16
2.2 Segmentation 17
2.3 Invariant Identification - Clustering 18
2.4 Invariant Models 18
2.5 Class Instance Model 19
2.6 Activity Spotting 19
2.7 Evaluation 20
2.8 Sensor Signals 20
    2.8.1 Trajectories 21
    2.8.2 Joint Angles 25
2.9 Case Studies 29
    2.9.1 Drink and Work 29
    2.9.2 Bicycle Repair 31
    2.9.3 Car Workshop 32
2.10 Summary 34

3 Segmentation **37**

3.1 Introduction 38
3.2 Related Work 38
3.3 Segmentation 40
    3.3.1 Basic Segmentation 40
    3.3.2 Low Velocity Segment Fusion 44
    3.3.3 Segment Optimization 46
3.4 Summary 47

4 Invariant Identification - Clustering 49
4.1 Introduction 50
4.2 Related Work 50
4.3 Cluster Algorithm 51
4.4 Best Invariant Selection 52
4.5 Cluster Precision 55
4.6 Summary 60

5 Invariant Models 63
5.1 Introduction 64
5.2 Related Work 64
5.3 Invariant Models Creation 66
5.3.1 Template Creation – Average 66
5.3.2 SVM – Sensor Signal Features 66
5.3.3 Activity Position Constraints model 67
5.4 Invariant Detection 68
5.4.1 Template Matching 68
5.4.2 SVM 68
5.4.3 Position Constraints Matching 69
5.5 Invariant Detection with Segmentation Optimization 71
5.6 Results 71
5.7 Summary 75

6 Class Instance Model 77
6.1 Introduction 78
6.2 Evaluation 78
6.3 Class Instance Model Creation 81
6.3.1 Possible Results 82
6.3.2 Parameter Training 90
6.4 Invariant vs. Normal Classifier Selection 91
6.5 Results 94
6.6 Invariant Detection Strategy Set Variation 98
6.7 Summary 102

7 Activity Spotting 103
7.1 Introduction 104
7.2 Evaluation 104
7.3 Spotting 107
7.4 Results 109
7.5 Summary 115

8 Conclusion and Outlook 117
8.1 Summery of achievements 118
8.2 Outlook 120
*

1

Introduction

The interaction between humans and computers evolved from switches performing a simple task like booting, to a keyboard communicating with a computer using a language, to the use of a mouse which provides a symbol based human-machine interface. This human-machine communication is bidirectional and consciously perceived from the human side. However, the human is always the active and the computer the passive part. A computer with the possibility to interact proactively, because it understands what the human is doing, is the next step in the technological evolution. Examples are a smartphone which automatically mutes the calls, when it detects the participation of an important meeting, the tracking of food intake and motion activity to aid in a dietary, the support for the mounting of furniture by video and audio instructions adjusted to the detected actual work step. Marc Weiser's vision in his article *The Computer for the 21st Century* [131] was that computer systems submerge in everyday life by supporting people nearly invisibly which he calls Ubiquitous Computing. He says that the most profound technologies are those that disappear by merging into the background. Furthermore, he compares this disappearing to the first information technology, the written word. In day to day life, our surrounding is full of writings and we get all this information at once without conscious thinking. This state of consciousness should also be reached for computer systems. However, people still stare into their screen while they interact with their computer or smartphone, instead the systems should be integrated into the daily routine and automatically aid human beings. Marc Weiser writes that for such systems two points are very important: scale and location. Scale means the size of the computer system which in the year 2017 has already reached several goals of ubiquitous computing, like smartphones, tablets, smartwatches and fitness monitoring devices. The location offers many possibilities: from fix mounted camera systems in a train station, to smartwatches at the wrist, smartphones in trouser pockets, or sensors woven into a cap. After all, size and location of the technical device are important for the disappearance but also the proactivity of the the computer system. Therefore a central challenge is the development of activity recognition methods to sense and evaluate human actions, which will be the main focus of this document.

1.1 Wearable Activity Recognition

Activity recognition consists of two kernel tasks: the sensing of human activity and the evaluation of sensor signals to recognize the human activity. Starting with the first, there are different possible sensor configurations. One option is to use extrinsic sensors, which observe the human being from the outside e.g. video [96] or infrared cameras [47] mounted on the ceiling recording the activities in a room. The advantage is the broad information provided by video surveillance. In contrast, the variation of the light conditions leads to problems. Also the location of the monitoring is not variable due to the fixation of the sensors in the environment. A further negative factor concerning especially video sensors are privacy issues. Another option of external sensing is to put sensors on every object of interest. Here we want to sense the interaction between the human being and different objects. An example is a person picking up a hammer, while the sensor or the identification marker in or on the hammer provides proper knowledge of this activity. The Internet of Things is the further logical step, where many devices of everyday life are interconnected and provide each other with information. Examples for sensors vary from acceleration sensors [18] to radio-frequency identification (RFID) tags [53, 106], microphones [40], break-beam sensors or contact switches [133]. Another sensing alternative is to monitor the subject of the activity directly using intrinsic sensors, which are inside the subject. The expression "inside" can be seen on different levels, from mobile phone sensors in a trouser pocket [14], to sensors woven into tight-fitting sports wear [98], to body implanted sensors. Examples for sensor modalities are acceleration [85], magnetic field [107], capacitative sensors [25] and pressure sensors [113]. The advantage here is that we get data directly from the human being with the ability to mimic its own biological sensing. This is only possible to a certain degree due to the difference of electronic sensors and the human senses. A question also is how to fit the sensors on the body parts properly. First of all, the sensors have to be fixed to locations which are important in the activity motion. Being left or right handed can make a big difference and negate a prior working system because it is mounted on the wrong side. The position of the sensors on the same body part can also introduce sensor signal variations [75]. We focus in this thesis on wearable sensing. This means only sensors, which can be worn on the human body and which can be removed easily are of interest, excluding all external sensing devices and also implanted sensors.

The second central task in activity recognition is to evaluate the sensor signal, mapping the signal data to the executed human activity. As explained before, we focus on wearable sensors. The experiments to evaluate our algorithm used inertial measurement units (IMU). This means that we can track the movements of human body parts with these sensors and then try to deduce the conducted activity from these movements. While low-level activities like walking, sitting or running can already be detected well, the more complex activities add more challenges to the recognition process [34]. Taking the baking of a pizza as complex activity example, the difficulties are that the order of the ingredients can vary, or some ingredients can be added or removed. The form of the executing of the single steps can also vary, e.g. someone cutting the cheese for the topping with a knife or rasping it with a grater.

Next, we describe a typical processing chain of an activity recognition algorithm [20]. First, all data coming from the different sensors is preprocessed. Filters can be adopted to get rid of sensor artifacts, the sensor data can already be fused or the labeling can be adopted. The next step is to segment the continuous data stream, which is to split the signal data into meaningful sub parts. This lowers the dimensionality of the data thus making it easier to handle the data. Segments are ideally picked with a strategy so that the segment borders are chosen at characteristic points related to activity instances. Next different features are extracted from the sensor signal segments, which separate all different activities from each other well. Additionally, a feature selection can improve the class separation. In the following activity recognition, the segments are investigated to find class memberships. Often a training step is needed beforehand where a classification algorithm is trained, using labeled signal data of the activity class. Typical classification algorithms include e.g. hidden Markov model [83], conditional random field [124] , support vector machine [21], decision tree [9], k-nearest neighborhood [74] and naive Bayes [101].

1.2 Challenges

Activity recognition is a complex research topic. Below we list several kernel challenges.

1. Activity variety: depending on the complexity of the activity which we want to recognize, the execution can vary extremely. The alternation can be different between human beings due to personal behavior or conditions, like being left or right handed, tired, excited, bored. The variation can also be possible between the activities of the same user. Some people may stick for example to adding ingredients while cooking in the order the recipe says while others already knowing the recipe by heart add ingredients randomly or as they remember or what is nearest in their actual view. Different body parts can also be used for the same activity. The usual way to close a door is to use a hand. However, having both hands occupied, the legs or the back can be used as well which results in a totally different body movement. The speed of the activity execution can also differ a lot depending on factors like urgency or practice of the activity. A hungry person will put the food faster into the mouth than a saturated one. Someone who knows to knit can produce a scarf faster than someone who has never done it before. When an activity class is defined, the different forms of execution have to be taken into consideration.

2. Null class: the null class is the class describing all activities which are not of interest. Therefore the activity recognition algorithm should not detect any of this found activity. The first problem with the null class is that the set of activities in this class contains all possible human actions except the ones which are of interest. This is a very large quantity. Secondly, the duration of the null class activities is usually much longer than the duration of the activities of interest. So the variation in the activities of the null class and the duration in time of the null class, makes it difficult to differentiate it from the activity classes of interest. It is also difficult to build a general model for the null class, e.g. for training reasons.

3. Sensor combination: when several sensors are used in the activity recognition scenario, then we have to synchronize them and fuse the data. The more sensors we integrate, the higher the dimensionality of the data gets. In this way, training of classifiers gets more complex which is known as the curse of dimensionality. Another problem occurs concerning the activity variety. If the door is closed with the hand or with the leg, then totally different sensors become important for the detection. Thus the identification, which sensor is important for which activity, plays an important role to make the classification process simpler and more accurate.

4. Ambiguity of sensor signals: in wearable activity recognition we put sensors on the human body. In this way, we try to get information similar to the own senses of a human being. Unfortunately, the information from the sensors does not have the

same content as the human senses provide us. This means that different activities which can be easily differentiated by human beings by relying on their senses, are not equally distinguishable by a wearable sensor setup. When a person moves his hand with a spoon to his mouth, or he moves his hand with a glass to the mouth, this looks quite similar when the motion is recorded with an acceleration sensor on the subject's hand.

1.3 Thesis Objective

The objective of this thesis is to develop a new process chain for wearable activity recognition, addressing several challenges, especially the problem of the variation in the activity execution. The main idea is to identify typical sub-activities or motifs for each activity class in the sensor data. These motifs are segments which are nearly identical for all executions of one activity class due to physical constraints. Therefore we have to separate the continuous sensor data stream into segments which form already sensible parts of the activities and then identify the best motifs from them. We call this new algorithm *invariants classification.*

1.4 Related Work

In this section, we introduce the state of the art in the research field of activity recognition. First, we discuss the term activity and its definition, and its development in social sciences. This helps us in identifying what we really want to recognize. We also see from a theoretical point of view, what represents an activity. Next, we introduce one of the most used sensor types in activity recognition, the accelerometer, and its application in this field. The experiments which we used in the evaluation of our new algorithm, have also utilized this sensor. Furthermore, we present other sensor modalities which allow the recording of several aspects of human activities. To motivate the research in activity recognition, we show different case studies where the use of a computer system, implementing these techniques, can be highly beneficial. The last subsection addresses motif detection, which is related to the central idea of our proposed new activity recognition algorithm. The aim is to find certain recurrent patterns in a dataset.

1.4.1 Taxonomy of Human Activity

In wearable activity recognition, we want to detect and classify human activities. But what is exactly human activity and how can it be defined? There exist several categorizations of human activities, which can help in the development of an activity recognition system. *The index of independence in activities of daily living* (Index of ADL) [60] describes a set of activities for the evaluation of autonomy of people with chronic illness and elderly people. In [103], time-use data is analyzed for ubiquitous computing applications. These datasets are collected by state or commercial institution and show the activities of people's daily life. *The compendium of physical activities* [1] collects a list of human physical activities and related energy cost in the context of sports. A more theoretical view on activities is presented in *Activity theory in a nutshell* [59]. It introduces the basic concepts and principles of activity theory, also taking historical developments into account. Next, we summarize the central statements of this publication:

The two central ideas which originate from Russian psychology are that consciousness and activity form a unity and that the human mind has a social nature. The basic principles of activity theory follow next: object-orientedness, hierarchical structure of activity, internalization-externalization, mental processes versus external behavior, interpsychological versus intrapsychological, mediation and development. The object-orientedness describes the strong relation between the activity and the object, which motivates and directs the activity of the subject. This can be seen as the objectives which give sense to what human beings do. The hierarchical structure of activity is expressed in three levels: activity, action, and operation. Actions are conscious and goal oriented and serve the fulfillment of the object. These actions can be further split into sub-actions with their own sub-goals. On the lowest layer are the subconscious and automated processes called operations. They do not have own goals but provide an adjustment of the action to the actual situation. The internalization-externalization is a process which puts the human mind into its social and cultural context. The next two points, mental processes versus external behavior and interpsychological versus intrapsychological, are strongly connected. The mental processes versus external behaviors highlight the difference between internal and external activity. Internalization puts the physical interaction with the environment into the mental domain. This can be partial or total. Externalization is the opposite of internalization and happens when the mental abilities are not enough or in a collaboration of several people. The interpsychological versus intrapsychological functions describe the form of mental development. First interpsychological functions are learned from another person. Later this turns into an intrapsychological function, when the social interaction is not anymore necessary, after mastering the ability. The mediation between subject and object is done by using tools. The development in the social and cultural context is an important part of activity theory. The article ends with the development of human activity in its interaction with the environment which is an aspect of research in activity theory. In our algorithm, we follow the idea that human activities can be split into a hierarchy of different layers. At the bottom layer, we identify the basic movements separated only by direction changes in the motion. This can be identified as operation, and examples here are one step forward in the walking process or pushing down a door handle. In our algorithm, these basic movements are separated by using our segmentation. On the middle layer of the activity hierarchy, we already take more complex activities which we get by aggregating several base movements, e.g. opening a door or drinking from a cup. This can be seen as action like proposed in the article before. Here the interesting thing is that the base movements can be divided into operations which vary in the execution or are not relevant for the action and operations which are performed in a uniform way due to physical constraints. On the top level, we find activities which are the most complex and can vary a lot not only between different users but also within the same user. Examples for this activity level are cooking a dish or going to work. Our classification algorithm will not address this type of complex activity. We want to identify actions with the help of typical and invariant operations. In the further text, we will use the term activity for action and the term sub-activity for operation.

1.4.2 Activity Recognition with Accelerometers

To get knowledge about human activity for further processing, we first of all need detectors to measure different physical properties of the environment in which the human is acting. The most common sensor used in wearable computing is the acceleration sensor. This sensor was also used in combination with a gyroscope and magnetic sensor in the evaluation experiments of this thesis. A typical scenario for context recognition, using this sensor, is to detect the basic locomotion of the human being like in [109], where accelerometers were used to identify walking, running, sitting, walking upstairs, downstairs, and standing. Mäntyjärvi et al. [84] also describe an experiment using multiple accelerometers to detect walking on a level, upstairs and downstairs. Gao et al. [44] examine single and multi accelerometer systems for the detection of eight activities: sitting down and standing up from an armchair, sitting down and standing up from a kitchen chair, sitting down and standing up from a toilet seat, walking up and down stairs, sitting down and standing up from a bed, lying down and getting up from a bed, getting in and out of a car seat, and walking ten meters. The aim is to implement not computationally intensive algorithms which can run on each single sensor node. The best result was achieved with a multi-sensor system using variance and mean features with a decision tree classifier providing a classification accuracy of 96.4%.

The integration of accelerometers in smartphones opened a new possibility to monitor users without additional hardware setups. In [77] they tried to identify six different activities of twenty-nine subjects: walking, jogging, ascending stairs, descending stairs, sitting and standing. The acceleration data is divided into ten-second segments. On each segment, 43 features are calculated and then three different classifiers are trained and tested: decision trees (J48), logistic regression and multilayer neural networks. The ten-fold cross-validation returned a classification result for most of the classes over 90%. In [11] again the triaxial accelerometer of a smartphone is used to identify six different activities which results in an overall classification accuracy of 91%.

Maurer et al. [86] check the influence of the sampling rate and the body placement of accelerometers. A dataset with six subjects performing six activities was recorded: sitting, standing, walking, ascending stairs, descending stairs and running. The recognition results were optimal with a sampling rate of the acceleration sensors between 15 Hz and 20 Hz. A higher frequency only showed minimal improvements. Six different positions on the body were tested (wrist, pocket, bag, necklace, shirt, and belt), whereas the wrist position showed the best results. In [66] they optimize the sampling rate of accelerometers and show the outcome on five benchmark datasets. If the sampling rate is too high, then resources are wasted, while a too low sampling rate loses information about the activities. Here statistical tests on unlabeled data decide which sampling frequency is best for each dataset.

Kunze et al. [75] deal with the misplacement of acceleration sensors on a certain body part. They could improve the recognition rate of displaced sensors from 24% to 82%, where the original result was 96%. In the context of sensor placement, they show in [76] that it is possible to detect, on which body part a sensor is located, while in [90] it is explained how to determine the orientation of a 3D accelerometer using gravity.

1.4.3 Other Sensor Modalities

In activity recognition, a lot of different sensor modalities are tested. Related to the conventional accelerometer sensors are ball or tilt switches, used in [126]. Here a prototype was constructed using ball switches, which close switches if sensors are tilted over a certain threshold and therefore only have two states, open or closed. Nine ball switches were positioned in 45-degree increments of each other in three perpendicular planes. In an experiment, the setup was compared to accelerometers for the recognition of the following activities: lying, kneeling, sitting, standing, walking, running, climbing stairs, descending stairs, bicycling and jumping. Despite the worse overall classification rate of 65.24% of the ball switches in contrast to 93.88% of the acceleration sensors, it was indicated that using rather simple sensors heavily distributed in clothing could return a benefit in the context of unobtrusiveness and also when the perfect sensor position cannot be easily determined. Another option is the pressure sensor. In [98] four force sensitive resistors are fixed around the lower forearm (right behind the wrist) and also around the upper forearm (right under the elbow) using a close fitting sleeve. Additionally, one 3D acceleration and gyroscope sensor was fixed on the back of the hand. Then 16 different gestures were conducted, all related to giving a talk in a seminar. The result shows that while the accelerometer performs better than the pressure sensor, it outperforms the gyroscope. Adding the pressure sensor to another sensor can bring additional information and improve the accuracy by 1% to 29%. The same force sensitive resistors were used in [70]. Here sensors were fixed on the front and the back of both thighs, sewed into a tight bicycle trouser. Then a subject rides a bicycle mounted on a roll. Now it is possible to count online, not only pedaling but also the force used by the muscles. This can be shown by detecting the gear used by the subject. Especially when the seven gears are grouped in three gear classes, the article shows the possibility to detect online the different strength efforts with this sensor modality. In this way additional information which is not accessible to acceleration sensors is provided. In [28] a matrix of 32 x 32 pressure sensors are integrated into a floor mat. When people stand on the floor, then it is possible to detect not only different gestures but also the executor. 78.7% accuracy is reached in the classification of seven activities conducted by 11 subjects. The identification of the performing subject reaches an accuracy of 88.6%. With the same pressure sensor matrix, the variability of this sensor modality is shown [27]. The analysis of five different scenarios are presented: the matrix is used on a car seat detecting driver's motions, as sports mat monitoring the exercises, as table surface checking nutrition intake, as sofa surface implementing a user interface for e.g. a TV and as smart carpet monitoring user activity.

In [24] sound is used to analyze human activity in a bathroom in the context of elderly care. For the classification of showering, urination, defecation, flushing, washing hands and sighing, a hidden Markov model was used with mel-frequency cepstral coefficient features. Most classes could be identified with an accuracy rate of 84%. In [52], an 8 x 8 thermal sensor array was tested for a surveillance scenario at home, where persons were tracked. The detection of the use of different kitchen devices, like toaster, egg cooker, and water cooker, and different actions like opening the refrigerator, were also investigated. This sensor system is not only cheap and simple but also has the advantage of fitting very well to the environments with privacy issues. It does not provide data which could violate the privacy of a person, like a camera system. Pirkl et al. [107] introduce a wearable magnetic sensor system for the position tracking of human beings and also for the tracking of different body parts. In an experiment, nutrition-related gestures and Tai Chi movements were tested. Cheng et al. [25] introduce capacitive sensor where the change in the capacity of the human body is measured. This capacity change can be related to various human motions. The sensor consists of a material which can be integrated into garments. This enables an unobtrusive placement of the sensors in the human environment.

1.4.4 Types of Scenarios

The motivation behind the development of an activity recognition algorithm is that computers should be enabled to interact and help humans by automatically detecting the needs and the context of the human being. In the publications in this research field we can identify three main areas for activity recognition: medical scenarios, sport and entertainment scenarios, and industrial scenarios. The medical scenarios help in the health care to ease the handling and supervision of the patient. This is also related to assisted living. In the sports scenarios the surveillance of the body in the training process is the priority. In work scenarios guidance for work processes should support the worker. Next, we show different use cases in these three areas.

Medical Scenarios

One aspect of activity recognition in the area of health care is the observation of fluid and food intake. This is helpful when patients have to follow a diet or in elderly care, where it is crucial that people do not dehydrate. Related to sports, computer support for the nutrition in a training plan is also beneficial. In this thesis, we have used one experiment connected to this topic. Here the gesture of drinking from four different vessels is embedded in several daily life activities. The identification of this dataset of drinking activities was already investigated in [2]. In [29] a capacitive sensor neckband is developed to monitor swallowing. This allows not only the detection of events related to eating, like having lunch or dinner, but also the detection of the activity level. Inactive periods can be distinguished here from sleeping. The scope of application lays in the area of cognitive disease monitoring or elderly care. In [26] a contact-free interface for the use in hospitals was implemented. A capacitive sensor was integrated into the garment of the doctor's overall. When the physician visits a patient, he can access the data of the patient on a PDA by moving with simple hand gestures over the cloth integrated sensor. Thus the physician - patient conversation is not disturbed by this unobtrusive interface and also the hygienic component of not having physical contact with the interface is a big advantage. In [46] 10 people with a bipolar disorder were observed using data from their smartphones. The aim was to detect transition changes between manic and depressive mood states. The analysis of the location, motion, and length of telephone calls provided the necessary information to see changes in the mood of the patients. Related to the former paper, [79] tried to identify the mood of users analyzing the data gathered by smartphones. The user statistics of the phones of 25 subject were recorded. This already allowed to differentiate four mood classes with an average accuracy of 91%. Mazilu et al. [87] introduce a wearable system to help Parkinson's disease patients with the problem of freezing of gait. Two IMUs are fixed at the ankles for the detection of the gait freeze. A smartphone analyzes the incoming sensor data, and when a freeze of gait is recognized, an auditory cueing is provided manifested as a metronome clicking. The option to train the reaction on freeze of gait incidents, using the system, is also given.

Sport and Entertainment Scenarios

In [91] video data is analyzed to recognize team activities in a football game. Three classes are identified: ball possession, quick attack, and set piece. Four classification strategies are tested: neural network-based classifier, k-nearest neighbor classifier, support vector machine classifier, and random forest classifier, while the last shows the best results. In [104] the team activity in a basketball game is analyzed with the help of video data. For this purpose, the data is first separated in three game phases: offense, defense and timeout. Then the strategy of the teams is checked in detail, identifying if the team is in one of three states: starting formation, screen or move. Seventy-one examples of three types of basketball offense are analyzed with this approach to show its consistency with the video data.

Wearable sensors are taken in video games as a human-computer interface. Mortazavi et al. [94] show the results of a developed fitness game including soccer related motions and running motions. Four wearable IMUs, attached on both wrist and the top of both feet, were used for the data acquisition of the players. An F-score of 0.9 was reached in a leave-one-out cross-validation with support vector machines classifiers. In [8] an IMU was used to gear a virtual car in a 3D video game into a parking space. The IMU was fixed on top of a glove on one hand. The recognition was able to identify 16 different simple arm gestures online.

Another application in the sports and entertainment field is related to emergency scenarios. For this purpose, it is interesting to know the activity not only of one single person but also of a whole crowd, e.g. in a big sports event. This is called crowd sensing, and a typical use case is to control and guide the evacuation of a crowd from a spot. In [132] the crowd density at one location was detected by scanning for Bluetooth devices. During a watch party of the European soccer championship, several smartphones were used to scan the environment for Bluetooth devices. On the basis of this data, it was possible to classify the density of the crowd from seven density classes with an accuracy of 75%. In [134] the crowd density was also analyzed. Here during the Lord Mayor's Show 2011 in London, people could install an app on their smartphone which gave information about the festival and also shared the position for the crowd analysis. As position sharing was voluntary, the distribution of people providing data was not known. Therefore the group density was estimated by calculating the movement speed of the people who shared their position.

Industrial Scenarios

The use of computer systems in a working environment with a running activity recognition system can have various advantages. Workers can be guided automatically, if there are questions, without the need of looking something up because the computer system knows the activity the worker is conducting at the moment. Furthermore, the work processes can be supervised and the worker can be warned of errors or dangers. In this thesis, we incorporate two experiments which are allocated to the industrial topic. The first one is a maintenance scenario in a car production factory which was also investigated before in [17, 97, 114]. The second is a bicycle repair scenario which was studied in [99, 100, 115]. Both scenarios are explained in detail in Chapter 2.9. In [112] a real-life manufacturing process was recorded with the aim of generating a benchmark dataset. One worker paints interior and exterior parts, adds serial numbers and boxes them. Two Kinect sensors were installed in the factory to record the activity of the worker. In total two full work days in normal operation were recorded resulting in a dataset of 17 hours. In [130] a wood workshop scenario was introduced. The activities to detect were sawing, hammering, filing, drilling, grinding, sanding, opening a drawer, tightening a vise and turning a screwdriver. The sensor setup consisted of two 3D accelerometers and two microphones, both fixed on wrist and arm.

1.4.5 Motif Detection

The idea of our algorithm is to find data segments which serve as descriptive representatives of an activity class. This can be seen as motif detection. The term motif was introduced in [81] for repeatedly occurring patterns in time series. This relates to bioinformatics where finding patterns in DNA sequences is a well-known problem, e.g. in [19]. The difference between human activity motifs and DNA motifs is that unlike DNA data, human activity signals consist usually of continuous data. Therefore a discretization step is often used like in [80]. Here a symbolic representation of time series called SAX is introduced which can be applied for motif detection. This allows using similar strategies as in bioinformatics [6, 32, 45]. Further advantages are the reduction of the dimensionality/numerosity, the definition of distance measures which are lower bounded to the distance measures on the raw data and the ability to be used online. [12] also implements a symbolic representation of the sensor signals. Here the motif discovery identifies patterns for long-term leisure activities like Zumba, cycling or badminton. The 3D acceleration data from a sensor attached to the wrist is first segmented using a linear approximation algorithm. Then the signal is discretized by mapping the slope of connected segments to a set of symbols. Typical motifs are detected with the help of a suffix tree.

Different segmentation strategies are applied to time series to identify motifs. In [41] a polynomial approximation of the time series is used for the segmentation and the clustering to detect the motifs. Other segmentation options implemented are minimal description length [121], hidden Markov models (HMM) [88, 89] or piecewise linear approximation [65].

Another factor in the finding of motifs is whether there is a one dimensional or a multidimensional time series. In activity recognition, it is nowadays common to record several sensors which requires the detection of motifs on several data channels. In [123] a detection algorithm for nonsynchronous motifs on multi-dimensional time series is proposed. First, it applies the motif detection of [30] to each data dimension, which is described next: the dimension of the time series is first reduced by applying piecewise aggregate approximation (PAA) [62] and then the time series values are mapped to equal sized areas on a Gaussian curve which represent the discrete symbols. Then the random project algorithm from [105] is adjusted to detect the motifs. For this purpose, repeatedly sub-strings are randomly chosen and then hashed. The sub-strings with most frequent collisions in the repeatedly hashing are then checked for a motif. The final step in the multidimensional case is to use a graph clustering for the motifs of the single dimension results to detect motif appearance over several time series dimensions.

1.5 Contributions

In this thesis we introduce a new algorithm for activity recognition: the *invariants classification*. In the following section, we want to discuss the research questions which arose during the development of our new algorithm. We also show the contributions of this thesis to the research field. Starting with a continuous data stream of sensor data, the first step is to organize this data in a way to make the activity recognition possible. A standard procedure is to separate the data into meaningful chunks. This brings us to the first research question:

- What is the best strategy to segment continuous sensor data, to get meaningful signal units related to the activity motions?

In chapter 3 we introduce our segmentation strategy. Here we calculate the central difference quotient on the sensor signal data and identify all its zero crossings. These zero crossings serve as segment borders of the original sensor signal. We see a logical relation between activities and this segmentation strategy because the zero crossing in the difference quotient signal is equal to a direction change in the motion of the activity. We want to detect the smallest sub-activity unit. Therefore a direction change in the motion is the adequate start and end of any sub-activity. We also calculated the mean and the standard deviation of the differences of the nearest segment border to the activity label borders. This helped us to identify if an add-on to our basic segmentation made sense or not. Furthermore, we introduced an improvement in the segmentation where we combined adjacent segments of the basic segmentation. In the later classification, we see that this segment aggregation generates some improvements in the classification results.

Next, we continue with the segmented data of several sensors and the labeling, which denotes where the activities of interest happen. Having in mind the challenges of activity recognition struggling with activity variation and sensor combination, we came up with the following three questions:

- Is it possible to select specific invariant activity parts which describe the complete activity?
- Which sensor signals are central for the recognition of specific activities?
- Can we compensate imprecise labeling of activity instances?

One basic idea of this thesis is that human activities consist of two parts: one being variable due to inter- or intrapersonal differences and other being relatively invariant. This invariance of certain sub-activities is caused by fixed physical preconditions. An example is the door handle which has to be pushed in a certain way and at a certain position to open it. In chapter 4 we introduce a clustering method which finds typical segments in the training activity instances. We call these clusters of such sub-activities *invariants*. The agglomerative hierarchical clustering is used on segments of all training activity instances. We identify clusters which have segment members covering a majority of the training activity instances. These segments can be seen as invariant sub-activities or motifs. A difference between finding motifs in DNA sequences and time series of human activities is that human activity can be conducted at different speeds, which changes the length of the signal in the time domain. To cope with this challenge we use dynamic time warping (DTW) as similarity measure in our algorithm, also in the clustering. The found clusters are the basis to construct three invariant models which enable us to detect class segments and in a further step to classify the entire class activity instance. We show on evaluation data that there exist certain activities which can be classified well by applying our *invariants classification*.

The whole clustering is done on the data segments of all sensors, identifying good sub-activity representatives. In the *class instance model* we train which *invariant models* and thus which clusters we use in the final classification. As each cluster is connected to a particular sensor signal, we automatically choose the best sensor channels for each activity class in the training of the *class instance model*. This makes our strategy a multi-dimensional asynchronous motif detection algorithm with variable motif length. This means that we find motifs on several data channels, motifs do not have to start and end at the same time on different data channels, and the motif length is not fixed.

The clustering is done, as mentioned earlier, on segments of all training class instances of a specific class. The instance borders of the activities are usually known from manual labeling. In contrast, the segment borders are automatically generated during the segmentation process. If the manual labels are not put precisely at the real start or end of the activity, then this can have a negative impact on the following classification. For our further classification, only the segments are essential. The segment and the instance borders do not exactly match, therefore we always also use segments which overlap the class instance borders. As a consequence our approach, relying on the segments and not the instance borders, is safe against inaccurate activity labeling.

The possibility to know at an early stage which classes are easy or difficult to identify and which sensors provide much or little information for the classification, can already help a lot in the further processing. Therefore we asked ourselves:

- Is it possible to make an early statement in the activity recognition process chain which classes can separate well and which classes can make problems?

- Is it possible to make an early statement in the activity recognition process chain which sensor signals are meaningful for the separation of different classes?

In the clustering process in chapter 4 we found a list of representative segments for the activity classes. By evaluating the pairwise similarities of the representative segments of two activity classes we get a new value which we call *cluster precision*. We calculate this value for all classes, always comparing two selected activity class sets. With this value, we can show which activity classes tend to be difficult to distinguish because the *cluster precision* reflects the similarities of representative segments. If we only take representatives from a certain sensor for the calculation of the *cluster precision*, we can also see how well this sensor differentiates among the different classes. This provides us with an early insight even before the classification step.

Finally, the most basic question in the area of activity classification is:

- What is the best way to classify human activity?

Our proposed *invariants classification* starts with the search for representative sub-activities of the activity classes. From these sub-activities we construct *invariant models* for the detection of segments of the activity class. After the construction of the *class instance model* and the training of different parameters for the fusion of several *invariant models*, we can classify class segments and then estimate the entire activity class instance. We test the *invariants classifier* on different scenarios and compare the performance with conventional classifiers from other publications. As a conclusion we see, that while our new classifier is not overall perfect in recognizing activities, there are certain types of activity classes, where the *invariants classification* brings a great benefit, as compared to conventional classifiers.

1.6 Thesis Overview

Here we present a short overview of each chapter showing the structure of this thesis.

Chapter 2 – Basic idea:
In this chapter we introduce our new activity recognition algorithm, called *invariants classification*, and give an overview of the single processing steps. First, the segmentation of the signal data is introduced. Next, the clustering step for the identification of groups of invariant sub-activities is described. These invariant sub-activities can serve as descriptive representatives of an activity class. Then we show the construction and application of three invariant models for the detection of invariant segments. To fuse several invariant models into a final classification result, we explain the construction of the class instance model. The last step of the *invariants classification* is the activity spotting where we classify activity class instances in a continuous data stream. After that, we discuss the evaluation approach of our activity recognition algorithm. Next, we show the preparation of the signal data which are used in the evaluation. We recorded IMU sensor data which provide orientation information of the sensor. As the sensors are fixed to a human body, we can construct a stick body model which provides us with an image of the human body postures. The data which we use in the further processing are the 3D coordinates of different body joints of the upper body part and the angles between them. Finally, we present three case studies where our activity recognition algorithm was evaluated: the *drink and work* experiment which records drinking gestures in an office/home scenario, the *bicycle repair* experiment which records the maintenance of a bicycle, and the *car workshop* experiment which records quality inspection activities of a worker on the line of a car manufacturing.

Chapter 3 – Segmentation
This chapter shows the segmentation of the sensor data in detail. First, we show different segmentation options from several publications. Then we introduce our basic segmentation strategy: we calculate the central different quotient from the continuous sensor signals and insert segment borders where the central different quotient signal has zero crossings. To improve this, we tried to merge segments with a low signal variance but did not stick to it due to the outcome. An improvement is the fusion of neighboring segments using a criterion from the clustering step, which we call *segment optimization*. Here the cluster member segments are used to assess if it is favorable to merge two segments.

Chapter 4 – Invariant Identification - Clustering
The aim here is the identification of invariant segments which can serve as representatives of an activity class. First, we introduce related work. Then we describe our cluster strategy, an agglomerative hierarchical approach with average linkage. A specialty in our algorithm is that we keep all possible clusters during the construction of the cluster tree. Then we select the best invariants by first sorting the clusters and then removing clusters with overlapping members along the order. To get an early statement about the classification prospects, we introduce the *cluster precision* measure. This value is determined by calculating the pairwise similarity of all cluster centers from two activity classes. This shows which classes tend to be mixed in a further classification due to the similarity of their segment representatives. The quality of data channels can also be derived in this context.

Chapter 5 – Invariant Models
This chapter introduces different invariant detection strategies of our algorithm. We show the construction of three invariant models: *template model*, *SVM model* and *activity position constraint model*. The *template model* builds a template of the invariant which is the cluster center of the invariant. The *SVM model* constructs an SVM with a feature set from other signal channels. The key point here is that the segmentation of the invariant's signal channel is used on these signal channels. The *activity position constraint model* determines the constraints which are related to the invariant member segment positions in their activity instances. These three models are applied to new sensor data signals in the invariant detection step. The *template model* implements a template matching with DTW similarity measure. The trained SVM of the *SVM model* is then applied only on the results of the template matching. Finally, the *position constraint model* checks the results of the template matching and the SVM, and also on the raw data if the position constraints are fulfilled. Furthermore, we show the invariant detection with the segment optimization which we introduced in the segmentation chapter. Here we fuse adjacent segments and apply either the template matching or the SVM model on these new segments.

Chapter 6 – Class Instance Model
To get the final classification result of the *invariants classification* we need to create the *class instance model*. This model fuses the results of several invariant detections and returns classified activity instances. In the *class instance model* creation we have to train several parameters. For a first insight into the classification capability of the *invariants classifier*, we show several results as precision-recall plots after a sweep over all these parameters. We also depict here the comparison classification results of conventional classifiers. Then we introduce the parameter training, using a leave-one-out cross-validation on training data which provides us with the *class instance*

model ready for classification. In a further step, we fuse the conventional classifiers with the *invariants classifier*. We decide which classifier to utilize by comparing the F1-score of the conventional classifier with the F1-score of the *invariants classifier* on training data and choose the classifier with the higher value. Next, we present the classification results of the evaluation experiments using the *invariants classifier*, the conventional classifier and the fusion of both. We show that the *invariants classifier* provides a notable improvement in the classifier fusion case for the *bicycle repair* and the *car workshop experiment*. In the final step, we try to vary the set of used invariant detection strategies which provides us with a small improvement in the outcome of the classifier fusion.

Chapter 7 – Activity Spotting
In this chapter, we use the detected class segments which we obtained from the former chapters and estimate the entire activity class instance. For this purpose, we use the information of the invariant position in its class instance which was already calculated in the *activity position constraint model*. Like in the *simple segment position model*, we estimate the class instance start and end position for each found class segment. Then we check, how many estimated class instance start and end positions match each other. All estimated class instances, meeting a certain threshold of matching instance borders, are kept in a list. Finally, we fuse all overlapping class instances and then have a final result of found activity class instances. Precision, recall, F1-score and SET measure show the quality of the end results applied to the evaluation experiments.

Chapter 8 – Conclusion and Outlook
The final chapter presents a summary of our new algorithm, the accomplished achievements and an outlook of the thesis.

2

Basic idea

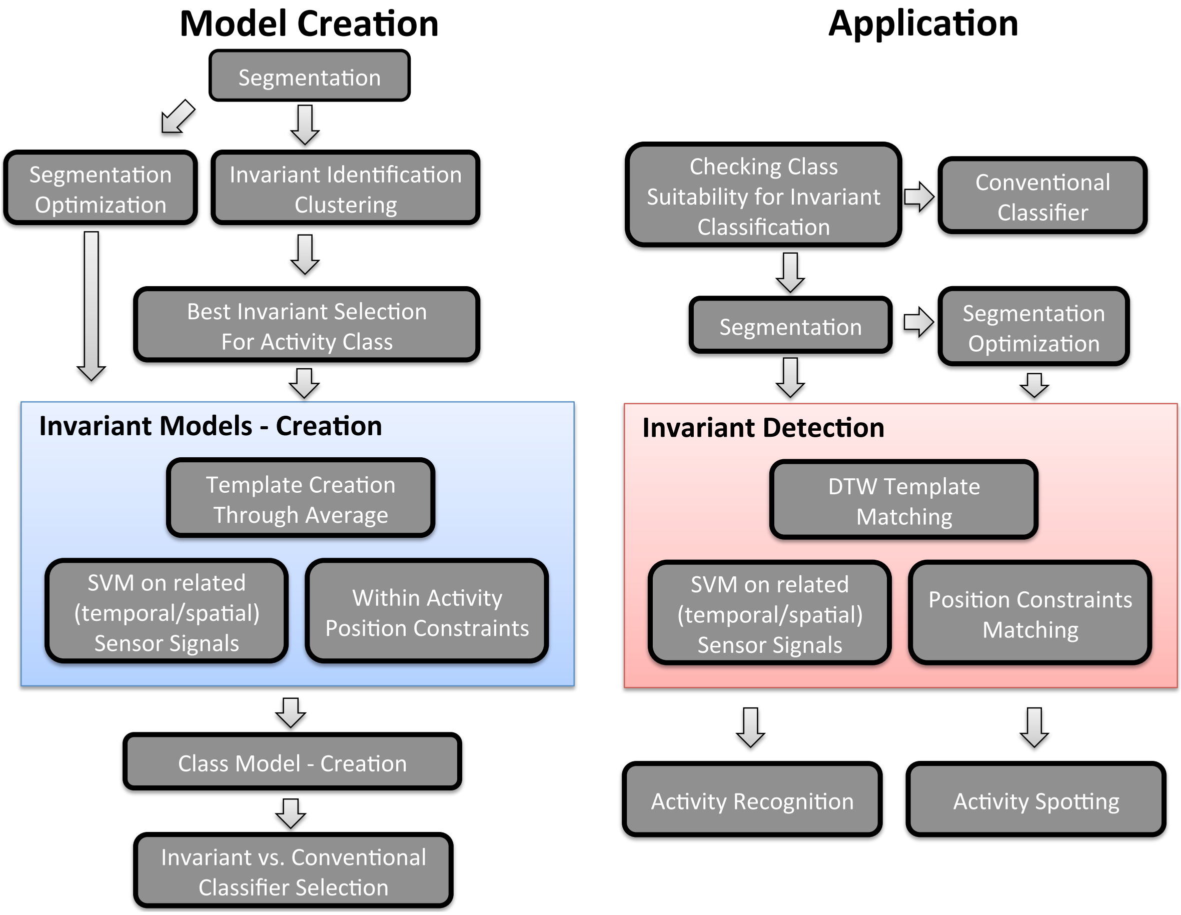

Model Creation
Application
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 2.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model.

2.1 Introduction

In this chapter, we show the basic concepts of the proposed *invariants classification* algorithm for human activities. The central idea of the algorithm is that beside of all variations in human activities, certain motions are constrained due to physical precondition and hence posses a similar, more or less invariant form. The *invariants classification* algorithm aims to detect such invariant signal segments and use them to improve the classification of the respective activities. The method is meant to be combined with existing classification approaches in an ensemble like fashion, being applied only to the classes for which appropriate invariants can be found and leaving the other classes to be handled by conventional methods.

In Figure 2.1 we see the overview of the processing chain of the *invariants classification* algorithm, which is divided into two parts: the creation of the classifier model and the application of the classifier model. The invariants classifier model building starts with the segmentation. Here we divide the continuous signals of the sensors into basic "closed" motions. This is done by cutting the data stream at all zero crossings of the central difference quotient of each sensor signal. We perform optionally a segmentation optimization step, where we fuse adjacent segments. Next, we try to identify the invariant sensor signals for each activity class. For this purpose, we use an agglomerative cluster algorithm on the segments of several activity instances of a specific class. Then we select a set of clusters which are our set of invariants. Selection criteria include temporal constraints of the invariant occurrence within the class instance as well as requirements with respect to the number of invariant occurrences inter and intra class instances. For every selected invariant we build three models which are needed later for the detection of invariant signal segments and also for the final activity classification and spotting. These three models are:

1. a template as an average of the cluster members,
2. a support vector machine (SVM) based classification model, using features extracted from other sensor channels on a time window that corresponds to the invariant,
3. constraints on the temporal position of the invariant within the respective class signal.

Next, we create the *class instance model* which is responsible for the final classification. For this purpose, we train optimal values for various parameters such as basic or optimized segmentation strategy, thresholds for the position constraints, and the set of invariants used in the recognition phase. For each class, we asses the performance of the system and compare it to the performance of a relevant conventional classifier. We then decide which classifier to use for the respective class.

The application of our classification model is structured in the following way: We distinguish between classes that can be handled by the invariants methods because they have considerable invariants and other available methods perform significantly worse as determined in the last step of the classifier model building. Then we perform the segmentation and optionally a segment optimization. In the invariant detection, we compare each segment to the templates of the class that we are trying to recognize using dynamic time warping (DTW). If the similarity is above a trained threshold (determined in the model creation), we then check if the application of the *SVM model* or the *activity position constraints model* makes sense to increase the detection result further. Next, the *class instance model* fuses the single results of the invariant models to a classification result. Here we take the activity instance borders as known. In the spotting we estimate these

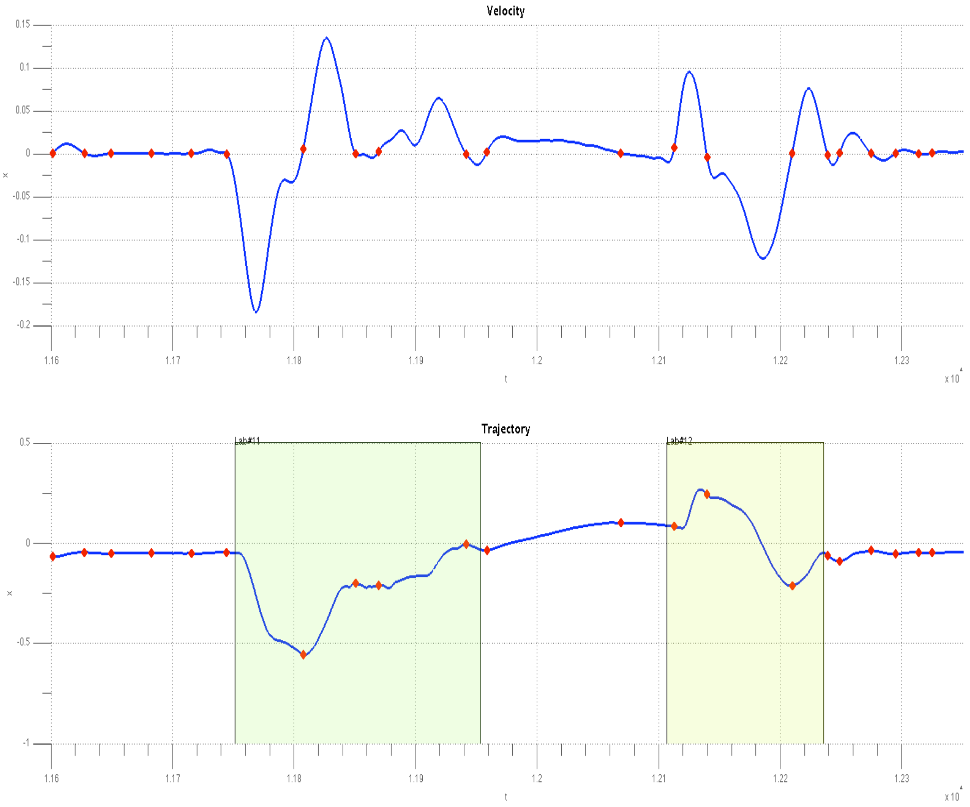


Figure 2.2: Segmentation example of the car workshop experiment. The lower plot is the raw signal (x-axis of a trajectory signal), the upper plot is its central difference quotient. The two colored regions of the lower plot represent activity class instances. The red dots are the segment borders (lower plot) or the zero crossing (upper plot).

activity instance borders using information from the *activity position constraints model*, providing a final classification result.

Furthermore, we introduce the evaluation of the *invariants classification* algorithm using three different experiments. We show the wearable sensor systems which were applied and explain how the sensor data is analyzed by calculating trajectories and angles of a human stick model. The chapter ends with the detailed description of the three experiments: the *drink and work* scenario, the *bicycle repair* scenario and the *car workshop* scenario. This chapter is based on [71, 72, 73].

2.2 Segmentation

The common approach to reduce the complexity of a continuous data signal is to split it into segments. In our case, we cut the sensor signal at all zero crossings of the central difference quotient.

$$\frac{\Delta_y}{\Delta_x} = \frac{f(x + \frac{1}{2}\Delta_x) - f(x - \frac{1}{2}\Delta_x)}{\Delta_x} \tag{2.1}$$

Figure 2.2 shows the segmentation of one axis of the trajectory signal of one body sensor in the car workshop experiment. The lower plot depicts one dimension of the trajectory signal. The red dots here represent the segment borders and the two colored areas represent two different activity class instances. The upper plot shows the central difference quotient signal. The zero crossings are marked here with red dots.

It is possible that the segments found in this way are too short to describe the activity well. So in an optional step, we optimize the segments by using knowledge from the *invariant identification*. This knowledge is gained by clustering class segments which is described in the following section. To optimize an invariant segment, we take all members of a found cluster, fuse them with their adjacent segments and test if they have a pairwise similarity as before. This is measured by instance coverage, which is the number of class instances the cluster members belong to, where class instances are the training samples of the entire activity we are searching. This additional strategy allows us to use more complex and longer signal snippets.

2.3 Invariant Identification - Clustering

After segmenting the signal data, the goal in the next step is to find significant segments for each activity class. In this context significant means that the segment is a typical representative of the activity. In the ideal case, this segment can be found unaltered in every activity instance. In many wearable activity recognition scenarios, several sensors are recorded, and these sensors return multi-dimensional signals, so we end up with several signal channels. Thus we want to select the best channels containing the best descriptive segments for each activity. To achieve this aim, we use hierarchical clustering with average linkage on all segments of one activity class of training data. Each signal channel is clustered separately as is each activity class. Ideally, each activity class should find one cluster with exactly one segment per training class instance which implies that the cluster has the same number of cluster members as training class instances. Class instance refers to the entire class activity which is separated into several segments. If one cluster contains several segments of one class instance, or if a cluster does not contain segments of several class instances then this cluster gathers segments which seem to be not very distinctive for this activity. Contrary to a standard cluster algorithm we collect all possible computed clusters which are generated from the bottom up clustering approach. In the subsequent *best invariant selection* step we sort the clusters and remove clusters with overlapping segment members. We finish with a set of disjunctive clusters for each activity class and each channel. We call these clusters *invariants*.

2.4 Invariant Models

For every identified invariant we build a model consisting of three parts which are used later for the invariant detection, the classification, and the spotting. These are:

1. template model,
2. SVM model on related sensor signals,
3. activity position constraints model.

In the *template model* we identify a signal segment which serves as representative for the invariant motion part of the activity. This representative segment is the average of the cluster members, which is the cluster center. The detection of invariant segments with this *template model* is realized using a template matching approach. We compare new signal segments with the invariant template. The similarity measure is DTW. The needed threshold value is evaluated by calculating the similarities between the cluster center and all cluster members. We choose the threshold value as the highest calculated similarity so that a predefined percentage has a smaller similarity to the cluster center. If the segment which we want to test is at least as similar as the threshold, then the segment is accepted as class segment.

The *SVM model* extends the former *template model*: in the model building and in the model application we use only segments which are detected by the *template model.* The start and end points of these found segments are used as segmentation on other signal channels. Next, we calculate features on these newly created segments. We try to get the most important features by using the feature selection measure called information gain. With the set of selected features, we build the *SVM model.* As we are working on labeled training data, we know which segments are true positives and false positives for the *SVM model* creation. In the application of the model we detect segments by only applying the trained *SVM model* to positively classified segments from the *template model.* In the *activity position constraints model* we build a model to check the invariant's position in the whole activity instance. To determine the instance position of the invariant, we calculate the mean and the standard deviation of the start and the end position in percentage of all member segments. The mean value is the start/end position in the instance. The standard deviation is needed as threshold for the tolerance of variations in the position. This threshold is used in the invariant detection when the *activity position constraints model* is applied. In the spotting phase, we do not use the *activity position constraints model* any more for the invariant detection. Instead, we use the information of this model to estimate the class instance border of all found class segments.

2.5 Class Instance Model

The invariant models detect class segments. To classify the entire class instance with these found class segments, we have to build the *class instance model.* This *class instance model* combines a certain number of invariant models and fuses their invariant detection results to a final classification result. Several parameters have to be trained among others the number of used invariants and the number of needed invariants. The number of used invariants defines how many invariants are used in the *class instance model* for the detection of class segments. The number of needed invariants denotes the number of invariants which have to detect a class segment inside the borders of a class instance so that the class instance is counted as found. Furthermore, we combine our classification algorithm with conventional classifiers in a fusion step. There we decide which classifier to use depending on classification results on training data.

2.6 Activity Spotting

The *class instance model* classified the activities with already prior known class instance borders. In the activity spotting, we identify the class instance borders for the found class segments of the *class instance model.* For this purpose, we estimate for each found segment the instance start and end by using information from the *activity position constraints model.* In this way, we can compare different instance estimations and stick to the ones which are found by several invariants. In the last step, we fuse found instances which overlap and get then the final outcome of the *invariants classification.*

2.7 Evaluation

To evaluate the proposed algorithm on wearable sensor data we use three different experimental setups:

1. The *drink and work* scenario where we want to detect the drinking motion from four different vessels in the context of office work, eating, gaming and leisure.
2. The *bicycle repair* scenario where the task is to maintain a bicycle.
3. The *car workshop* scenario. This is a car quality inspection scenario in a production line.

The reason why we chose these scenarios is that they can be seen as basic principles for the implementation of real-world applications. The *drink and work* setting is connected to healthcare and sports. If it is possible to detect the fluid intake of a person, we could, e.g. secure that in a home for elderly people all residents drink enough. This is crucial because the elderly have a decreased thirst sensation [61]. The surveillance of the water balance of the own body during the whole day can also be favorable for people being on a diet or following a training plan. The *bicycle repair* and the *car workshop* settings are located in the industrial area, where workers are supported in their job by “intelligent” devices. The *car workshop* scenario design evolved in cooperation with Skoda where the idea was to help workers in the quality inspection after the car assembling. While they check the car, they should carry a device like a tablet which recognizes over body-worn sensors if any tasks were missed. The possibility to look up information about any task is another assistance. The application for the *bicycle repair* scenario could be in the end user field, where a tablet or smartphone supports a non-professional in mounting a new bought part on a bicycle.

The three experiments were recorded in our research group, and we took the datasets to test our new *invariants classification* algorithm. Scientific results based on all three studies have been published in several papers as mentioned in the following descriptions of the experiments.

In the evaluation of our algorithm, we apply two connected leave one out cross-validation [68], where we train on n-1 or n-2 subjects and test on one subject (n stands for the number of all experiment subjects). We decided for the leave one out cross-validation and against, e.g. a ten-fold stratified cross-validation because the subjects form a natural unity in each experiment and have several repetitions of each activity. The first cross-validation is to train the parameters, and the second cross-validation is to test the classification quality of the algorithm.

2.8 Sensor Signals

In the experiments all subjects had body mounted inertial measurement units (IMUs). These sensors were fixed on different body positions, which can be seen in Figure 2.5. Here follows the set of all used sensor locations with their abbreviation:

- back (bac)
- chest (cht)
- right/left upper arm (rua/lua)

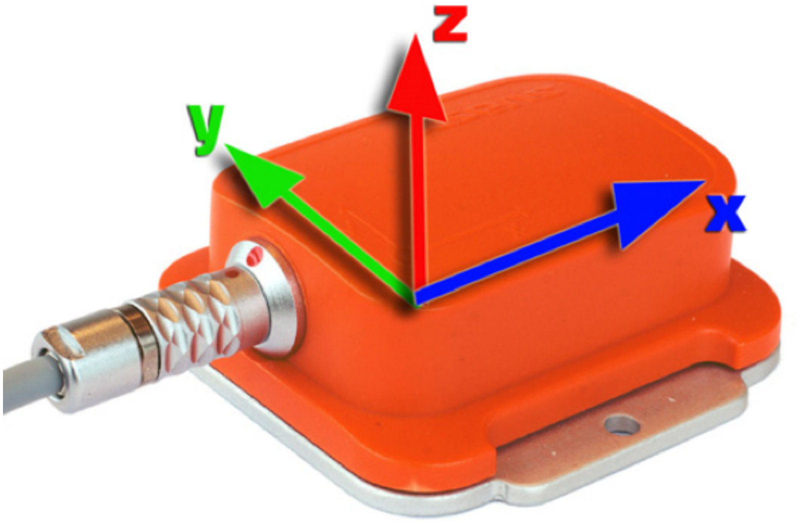


Figure 2.3: MTx sensor from Xsense

- right/left lower arm (rla/lla)
- right/left hand (rha/lha)

The IMU we utilized is depicted in Figure 2.3. The sensor called MTx is from Xsense[138]. It has a size of 38 x 53 x 21 mm (width x length x height) and weighs 30g. It collects 3D acceleration, 3D rate of turn (gyroscope) and 3D earth-magnetic field data. Additionally using the previous raw measurements and a Kalman filter, a drift-free orientation of the sensor is internally computed, like described in [82, 111]. The measured gravity and magnetic north pole of the earth are used to compensate the drift which comes from the integration error of rate of turn from the gyroscope which is often called Attitude and Heading Reference System [137]. The orientation can be obtained with 100 Hz while calibrated sensor reading is possible with maximum 512 Hz. In Table 2.1 is an overview of the sensor's specification of the data performance.

Table 2.1: MTx sensor data performance specification.

	Rate of Turn	Acceleration	Magnetic Field
Dimensions	3 axes	3 axes	3 axes
Full Scale (FS)	± 1200 deg/s	± 50 m/s	± 750 mGauss
Linearity	0.1 % of FS	0.5 % of FS	0.2 % of FS
Alignment Error	0.1 deg	0.1 deg	0.1 deg
Noise Density	0.05 deg/s Hz	0.001 m/s^2 Hz	0.4 mGauss/Hz

2.8.1 Trajectories

The IMUs used in our experiments provide us not only with the raw acceleration, rate of turn and magnetic field signals but also with the orientation information of each sensor either as unit quaternions, Euler angles or rotation matrix (directional cosine matrix). Each rotation matrix can be seen as the three unit vectors of the sensor coordinate system

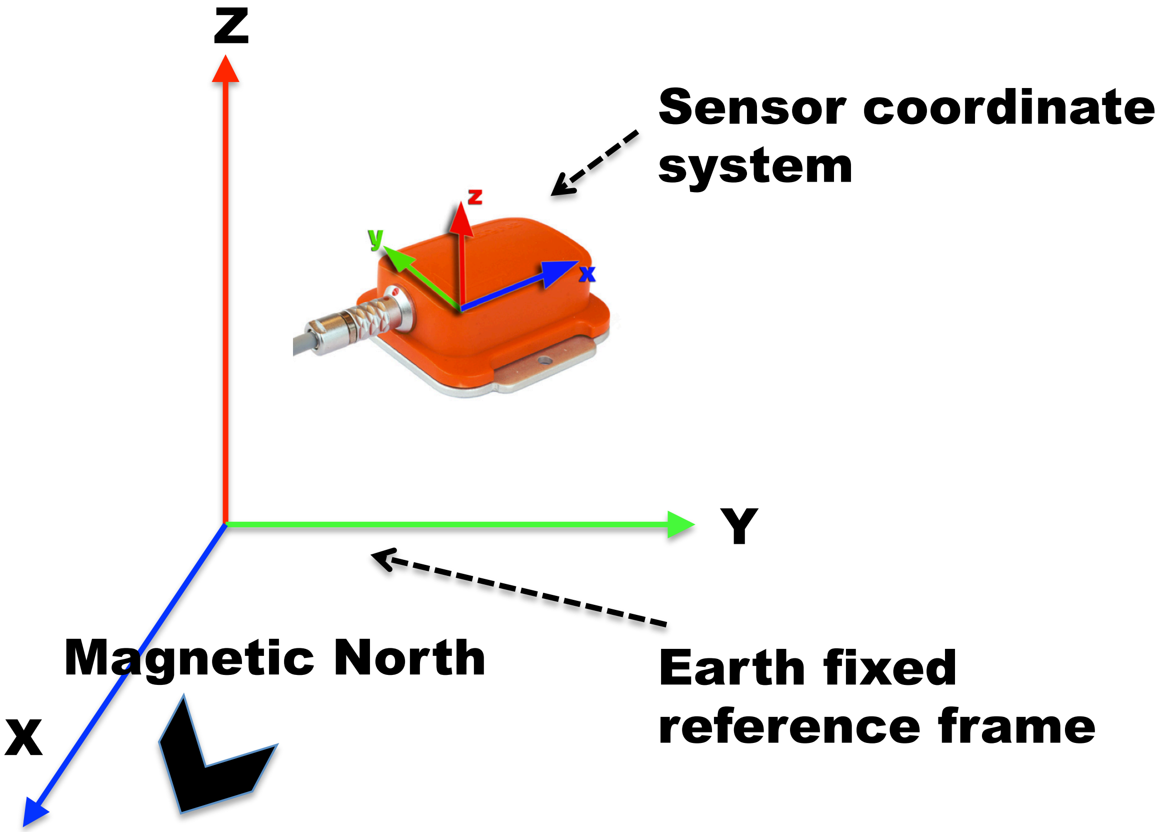


Figure 2.4: MTx-sensor: earth fixed versus sensor fixed coordinate system.

expressed in the earth fixed reference coordinate system, see Figure 2.4. We construct a stick model of the whole upper human body by connecting the single orientations of the sensors fixed on the body. Additionally, we need the estimation of the length of every single body part. The results are x, y and z coordinates in a Cartesian coordinate system for the right/left hand tip (rht/lht), right/left hand wrist (rwr/lwr), right/left elbow (rel/lel), and right/left shoulder (rsh/lsh), see Figure 2.5. These 3D vectors over the time, which we call trajectories, give us a time series which represents the body movement for further analyzes. The central bottom of the human torso is the origin $(0, 0, 0)$ of the chosen earth-fixed reference frame [7]. The trajectory of the right and left shoulder is calculated by applying the rotation matrix of the back or chest sensor on a vector with the length of the estimated upper body. As the left and right shoulder is dependent on the data of one sensor, we also have included the estimated broadness of the human upper body to get both trajectories. The rest of the stick figure model is constructed by rotating a vector with the assumed body limb length according to the corresponding sensor orientation and then adding all body trajectories which the new body part depends on. The broadness of the body limbs are not considered. T stands here for the 3×3 rotation matrix of each sensor and l for the estimated length value. Next all formulas for the right body side, the

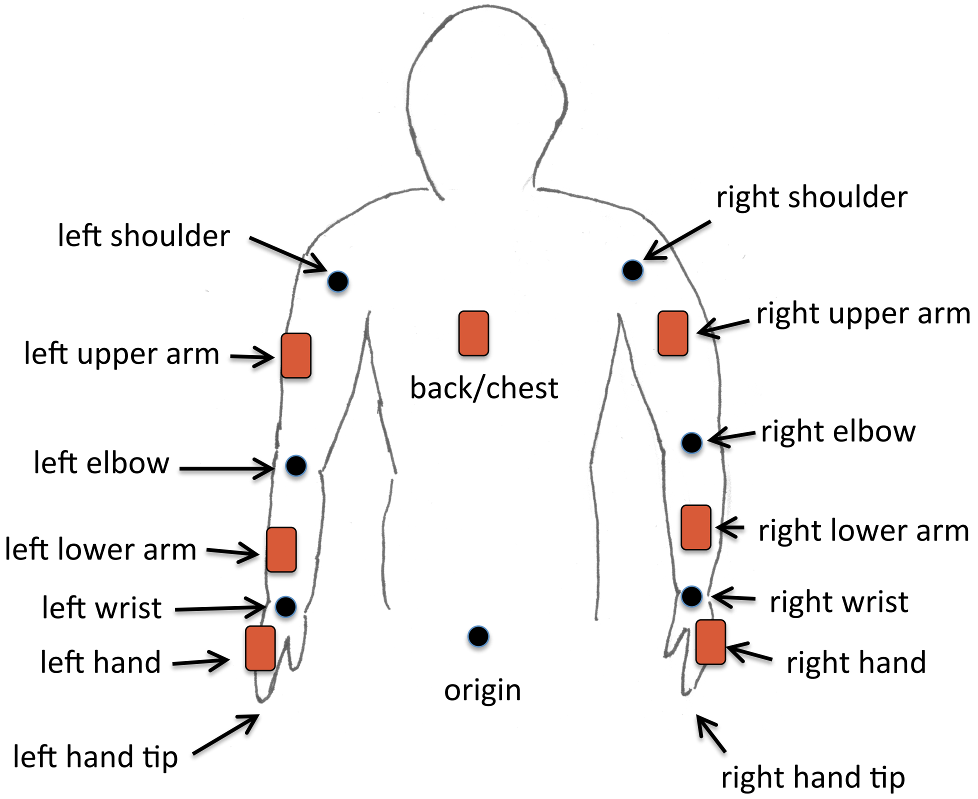


Figure 2.5: The position of the MTx sensors and the trajectory locations.

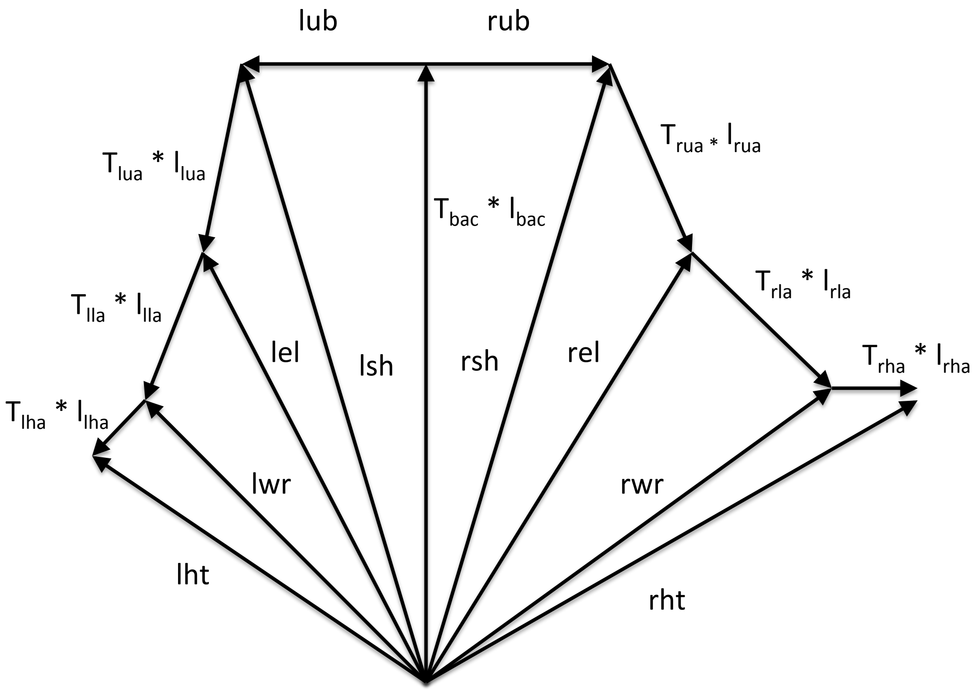


Figure 2.6: Stick figure constructed of the sums of the orientation matrices T multiplied with the body part length vectors l.

left side is calculated equally. All different vectors are also depicted in Figure 2.6.

$$\begin{aligned}
r\vec{sh} &= T_{bac} * \begin{bmatrix} l_{bac} \\ 0 \\ 0 \end{bmatrix} + r\vec{ub} \\
r\vec{el} &= T_{rua} * \begin{bmatrix} l_{rua} \\ 0 \\ 0 \end{bmatrix} + r\vec{sh} \\
r\vec{wr} &= T_{rla} * \begin{bmatrix} l_{rla} \\ 0 \\ 0 \end{bmatrix} + r\vec{el} \\
r\vec{ht} &= T_{rha} * \begin{bmatrix} l_{rha} \\ 0 \\ 0 \end{bmatrix} + r\vec{wr}
\end{aligned} \tag{2.2}$$

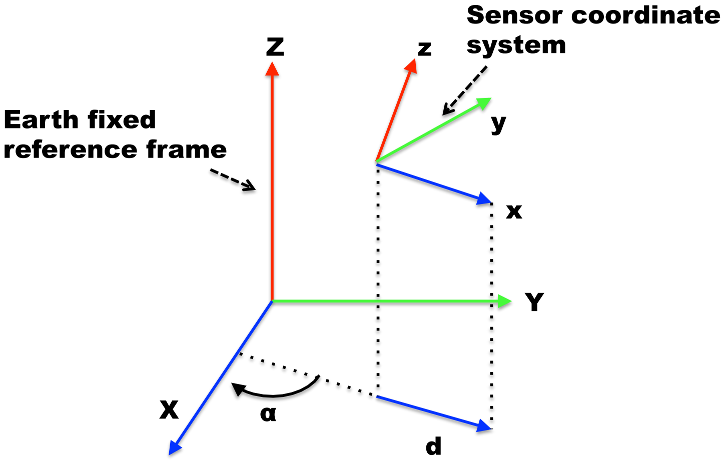


Figure 2.7: Heading compensation.

After the calculation of all trajectories, it is possible to plot the body posture over time. When we look at our stick model of the upper human body during different executions of the same activity, we can identify a divergence factor: the yaw angle of the sensors. This is the heading of the recorded persons, and it can induce a big variation into the human movement. We decided to compensate this by turning all subjects in the same direction. For this purpose, we have to calculate the angle α between the earth-fixed reference coordinate system and the sensor coordinate system of the back/chest sensor, see figure 2.7. We project the x-axis of the sensor coordinate system on the plane spanned by the x- and y-axis of the reference coordinate system which results in a vector $\vec{d}$.

$$\begin{aligned}
\vec{x}_{bac} &= T_{bac} * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
\vec{d} &= \begin{bmatrix} \vec{x}_{bac}(1) \\ \vec{x}_{bac}(2) \\ 0 \end{bmatrix}
\end{aligned} \tag{2.3}$$

Between the vector $\vec{d}$ and the x-axis of the reference system we get the angle with the following formula:

$$\alpha = atan(\frac{\vec{d}(1)}{\vec{d}(2)}) \tag{2.4}$$

We now know the compensating angle with which we can rotate all the body nodes so that the entire body faces the same direction. This compensation turn uses the rotation matrix R_{comp} which is calculated in the following way.

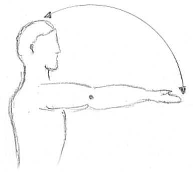
$$R_{comp} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

By simply multiplying all trajectories with the rotation $matrixR_{comp}$ we get the heading corrected. The trajectory calculation and the orientation adaption of the upper body was taken from [114].

2.8.2 Joint Angles

(a) upper arm moving sideways - body (shoulder joint).

(b) upper arm moving forward - backward (shoulder joint).



(c) extension - deflection of upper/lower arm (elbow joint).

(d) hand movement in the direction of radius - ulna (wrist).

(e) hand movement in the direction of the back of the hand - palm (wrist).

Figure 2.8: Joint angles.

Furthermore, we calculate the angles between body limbs which correspond to possible joint movements. In particular, we are interested in 5 angles:

1. upper arm moving sideways - body (shoulder joint); Figure 2.8a.

2. upper arm moving forward - backward (shoulder joint); Figure 2.8b.

3. extension - deflection of upper/lower arm (elbow joint); Figure 2.8c.
4. hand movement in the direction of radius - ulna (wrist); Figure 2.8d.
5. hand movement in the direction of the back of the hand - palm (wrist); Figure 2.8e.

Angle between two Vectors

The IMUs provide us with a 3 × 3 rotation matrix for each sensor. This rotation matrix can be seen as the result of a rotation of the three axes of the Euclidian coordinate system of the sensors (passive transformation). Figure 2.3 shows the MTX sensor with the three axes marked on it. To calculate the angle between two IMUs we take the two appropriate vectors derived from the orientation matrices and calculate the angle between them.

The first strategy used was to take the formula of the dot product returning an angle value in radiant.

$$\begin{aligned}\vec{a} \cdot \vec{b} &= |\vec{a}||\vec{b}| cos \angle(\vec{a}, \vec{b}) \\ \angle(\vec{a}, \vec{b}) &= acos\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}\right) \end{aligned} \tag{2.6}$$

The problem by calculating the angle using the arc cosine is that we get angles only between 0°and 180°. Also the arc cosine has inaccuracies near 0°and 180°. If we test the following formula 2.7 in Matlab we see the result in Figure 2.9. As input we take all values between 0 and π in steps of $\frac{\pi}{10^8}$.

$$y = x - acos(cos(x)) \tag{2.7}$$

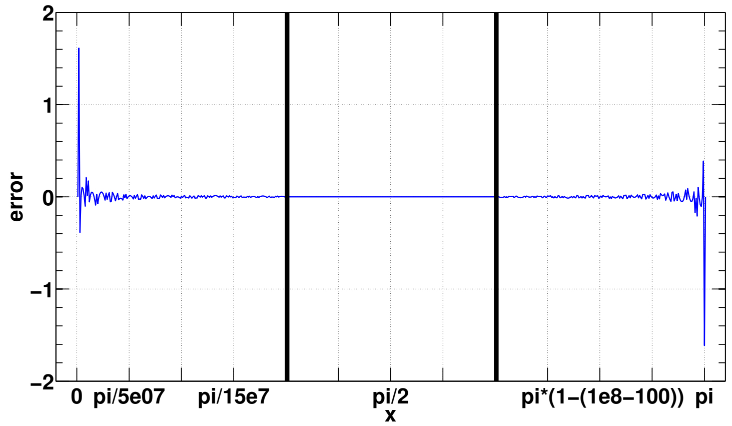


Figure 2.9: Error of arc cosine: results of $x - acos(cos(x))$, $x = 0 : \pi/1e08 : \pi$.

Angle within a Plane

To overcome the limited angle range and the inaccuracy we tried to compute the angle with another strategy which we explain next. In the end, we decided against this strategy because of problems in determining if the angle lays between 0°-180°or 180°-360°. Also, the inaccuracy in the arccosine function is not that relevant for our analysis. We use as before the two vectors $\vec{a}$ and $\vec{b}$ from the rotation matrices of the sensor. Additionally, we add a reference vector $\vec{r}$ which helps us to extend the angles to 360°. Instead of arccosine we now use arctangent for the final angle calculation. First we have to calculate the dot and the cross product of both vectors $\vec{a}$ and $\vec{b}$.

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{3} a_i * b_i \tag{2.8}$$

$$\vec{a} \times \vec{b} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix} \tag{2.9}$$

To know if the angle is between 0°-180°or 180°-360°, we need the reference vector $\vec{r}$ which is orthogonal to the other two vectors and is located on the appropriate body joint axis, i.e. we can imagine this reference vector as joint around which the two vectors $\vec{a}$ and $\vec{b}$ turn. We take for this reference vector one of the appropriate axis vectors, obtained from the rotation matrix of one sensor. This vector is compared to the cross product of the two original vectors $\vec{a}$ and $\vec{b}$. If the dot product of the cross product of $\vec{a}$ and $\vec{b}$ with the reference vector $\vec{r}$ is positive, then we know that both lay on the same side of the plane spanned by $\vec{a}$ and $\vec{b}$ otherwise not. Depending on the sign of the outcome of the dot product we can decide in which range the angle has to be.

$$signvalue = (\vec{a} \times \vec{b}) \cdot \vec{r} \tag{2.10}$$

The final step of the angle calculation is to calculate the atan2 of the length of the cross product of $\vec{a}$ and $\vec{b}$ and the dot product of $\vec{a}$ and $\vec{b}$. This is multiplied with the sign of *signvalue*. Finally, to get the angle in radiant, we only have to calculate modulo $2 * pi$.

$$|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| sin\angle(\vec{a}, \vec{b}) \tag{2.11}$$

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| cos\angle(\vec{a}, \vec{b}) \tag{2.12}$$

$$\angle(\vec{a}, \vec{b}) = atan(\frac{|\vec{a} \times \vec{b}|}{\vec{a} \cdot \vec{b}}) \tag{2.13}$$

$$angle = mod(sign(signvalue) * atan2(|\vec{a} \times \vec{b}|, \vec{a} \cdot \vec{b}), 2 * pi) \tag{2.14}$$

The problem we had with this strategy was that we could only be sure that the reference vector was orthogonal to one of the two vectors $\vec{a}$ and $\vec{b}$, the one which was from the same MTx sensor and its orientation matrix. The other vector was not necessarily perpendicular to the reference vector. Therefore we could not always correctly determine if the angle was between 0°- 180°or 180°- 360°.

Final Angle Calculation Strategy

Going back to the biological preconditions we see that angles greater than 180°are only relevant for the shoulder joint when the arm moves forward and backward. The movements of interest of elbow and wrist joint are between 0°and 180°. So we decided to use two different angle calculation methods:

First the shoulder joint movement and its need for angles between 0°- 360°. We start with the 3×3 rotation matrices A of the upper arm and B of the back, both provided by the MTx sensors. Each row in above matrices stands for a basis vector of the sensor coordinate system: $\vec{a_x}, \vec{a_y}, \vec{a_z}$ is the basis for the upper arm sensor and $\vec{b_x}, \vec{b_y}, \vec{b_z}$ is the basis of the back sensor, while $\vec{1_x}, \vec{1_y}, \vec{1_z}$ stands for the basis of the earth fixed reference frame. The relation between the earth fixed reference frame basis, the sensor basis and the rotation matrices can be described like this:

$$\begin{bmatrix}\vec{a_x}\\ \vec{a_y}\\ \vec{a_z}\end{bmatrix} = A^\top \begin{bmatrix}\vec{1_x}\\ \vec{1_y}\\ \vec{1_z}\end{bmatrix} \tag{2.15}$$

$$\begin{bmatrix}\vec{b_x}\\ \vec{b_y}\\ \vec{b_z}\end{bmatrix} = B^\top \begin{bmatrix}\vec{1_x}\\ \vec{1_y}\\ \vec{1_z}\end{bmatrix} \tag{2.16}$$

The next step is to transfer the basis vectors of the upper arm into the coordinate system of the back sensor. For that, we need a new rotation matrix R which we get using formula (2.15) and (2.16). In the end, we divide the transposed rotation matrix A of the upper arm by the transposed rotation matrix B of the back resulting in a 3×3 matrix R. In this way, we get the coordinates of each basis vector of the arm sensor in the coordinate system of the back sensor.

$$\begin{bmatrix}\vec{a_x}\\ \vec{a_y}\\ \vec{a_z}\end{bmatrix} = R \begin{bmatrix}\vec{b_x}\\ \vec{b_y}\\ \vec{b_z}\end{bmatrix}$$

$$A^\top \begin{bmatrix}\vec{1_x}\\ \vec{1_y}\\ \vec{1_z}\end{bmatrix} = RB^\top \begin{bmatrix}\vec{1_x}\\ \vec{1_y}\\ \vec{1_z}\end{bmatrix}$$

$$R = \frac{A^\top}{B^\top} \tag{2.17}$$

The basis vector of the x-axis of the upper arm sensor $\vec{a_x}$ points alongside the arm. We select this vector and transfer it into the coordinate system of the back sensor obtaining the new vector $\vec{r_x}$. Actually $\vec{r_x}$ is the first row of the rotation matrix R, calculated above. The axes of the back sensor point in the following direction: x-axis alongside the body, y-axis broadside the body and z-axis outside the body. To get the angle of the movement of the upper arm from the body sidewise up and down, we need the plane of the x- and y-axis of the back sensor. We project $\vec{r_x}$ into this plane getting the new vector $\vec{p_x}$ and then calculate the angle between $\vec{p_x}$ and the x-axis of the back sensor $\vec{b_x}$.

$$\vec{p_x} = \begin{bmatrix} n_x(1) & n_x(2) & 0\end{bmatrix} \begin{bmatrix}\vec{b_x}\\ \vec{b_y}\\ \vec{b_z}\end{bmatrix} \tag{2.18}$$

$$angle = atan2(n_x(1), n_x(2)) \tag{2.19}$$

For the angle of the movement of the upper arm in the forward and backward direction the only difference is that we project the transferred x-axis vector of the upper arm sensor $\vec{r_x}$ on the plane spanned by the x- and z-axis vectors of the back sensor $\vec{b_x}$ and $\vec{b_z}$. A problem can occur when the vector $\vec{r_x}$ is nearly perpendicular to the plane. Sensor noise can cause bias in the calculated value of an angle. In our case, we plotted the angles of our experiments and checked for anomalies but did not find any obvious errors.

The remaining angles for the elbow and joint wrist do not need an angle greater than 180°. Also projecting one limb into a plane is not so easy for these joints. For example, considering the elbow joint: the possibility of the lower arm to turn around makes it difficult to define a stable plane for the elbow joint angle calculation. So we decided to

Table 2.2: Basis vectors for elbow and joint wrist angle calculation.

movement	vector 1	vector 2
elbow wrist up down	upper arm x	lower arm x
hand moving sideways (radius - ulna)	lower arm y	hand x
hand moving up down (back of the hand - palm)	lower arm z	hand x

calculate the angle between two selected basis vectors, obtained from the rotation matrices of the sensors, using the arc cosine formula. The accuracy problem we mentioned before is not severe. We also tried the arctangent formula but it resulted in data jumps when the angle changes from positive to negative angles, so we decided for the arc cosine formula. We selected basis vectors shown in Table 2.2. The two vectors of the angle calculation start both from the origin. To get the correct angle of the body limbs, we have to multiply one of the vectors with -1 so that it points in the opposite direction as it can be seen in the last formula.

$$\angle(-\vec{a},\vec{b}) = acos(\frac{-\vec{a}\cdot\vec{b}}{|-\vec{a}||\vec{b}|}) \tag{2.20}$$

2.9 Case Studies

2.9.1 Drink and Work

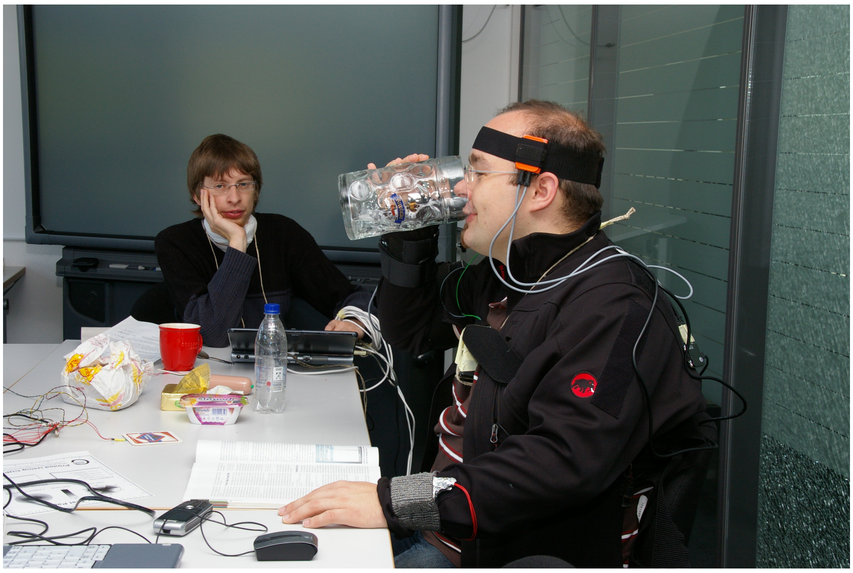


Figure 2.10: Drink activity from the drink and work experiment.

This dataset consists of several drinking events embedded in a daily scenario. The subject drinks from four different drinking vessels (glass, bottle, mug and cup) while completing four scenarios in a typical daily routine: office work, eating, gaming and leisure. Figure 2.10 shows a picture from the data recording of one drink activity. The detailed activity protocol, taken from [2], follows:

Office: *Sit down, turn on computer and select OS to boot. Start browser, go to the University homepage, download expenses form, start word processor (by clicking the*

Table 2.3: The four different classes contained in the drink and work dataset.

class	activity	class	activity
1	drink from glass	3	drink from mug
2	drink from cup	4	drink from bottle

Table 2.4: Signal channels of the drink and work experiment.

nr	channel	nr	channel	nr	channel
1	right shoulder x	5	right elbow y	9	right wrist z
2	right shoulder y	6	right elbow z	10	rsh angle front-back
3	right shoulder z	7	right wrist x	11	rsh angle side-body
4	right elbow x	8	right wrist y	12	rel angle extend-deflect

form) and fill in details with the keyboard. Print the form, fetch pages from the printer (in a printer room at 10 m distance), go back to the desk, sit and sign the form. Go to copier (in another office room), copy it, put one in post tray, go back to desk and sit. Punch second copy and put it in file. During the above activities, take at least ten sips from a glass at the time of your choice.

Eating: *Go to the coffee corner, take the water tank from the coffee machine, go to tap, empty and refill water, go back to machine and insert back the tank. Fetch a cup from the cupboard and fill it with coffee/tea/water from the machine. From this stage on, start taking sips from the cup at arbitrary moments (at least 10 for the scenario). Fetch a plate and knife, go to the table and sit down. Cut a slice of bread/cake, put topping (butter, cheese, etc. onto the bread) and eat it. Use a tissue to clean the mouth and then bring back the plate and knife.*

Gaming: *Fetch a bottle of water from a tray at room corner, open it for the first time and put it on the table. Switch on the Wii console, start a game and play Wii sports for 5 min (standing, using the gesture based Wii controllers). Switch console off again. During pauses in the games take at least 10 sips from the bottle.*

Leisure: *Reading journal, scratching head, answering phone call (simulated), glancing at watch, talking and make further arbitrary movements. Take at least 10 sips from a mug at arbitrary moments.*

The presence of ambiguities (e.g. drinking vs. eating or scratching the head) and a great percentage of the NULL class makes this dataset particularly interesting for analysis. The final aim is to classify the drinking activities as shown in table 2.3. We used three different class definitions:

1. Drink and work:
   Distinguishes four drinking vessels (glass, bottle, mug, and cup); the gesture starts from lifting the vessel from the surface and putting it back.
2. Drink and work, one vessel:
   All four drinking vessels are counted as one class; the gesture starts with lifting the vessel from the surface and putting it back.

3. Drink and work, one vessel, sip:
   All four drinking vessels are counted as one class; only the act of actual drinking is recognized as a gesture.

Altogether six subjects were recorded, each for 50 - 60 minutes, including about 12 minutes of drinking. In total we collected 5.84 hours of data and 560 drinking activities from different vessels were conducted. Each subject had three IMUs attached: right upper arm, right lower arm and back torso. Table 2.4 shows all data channels which we used in our gesture recognition algorithm. This dataset was published in [2].

2.9.2 Bicycle Repair

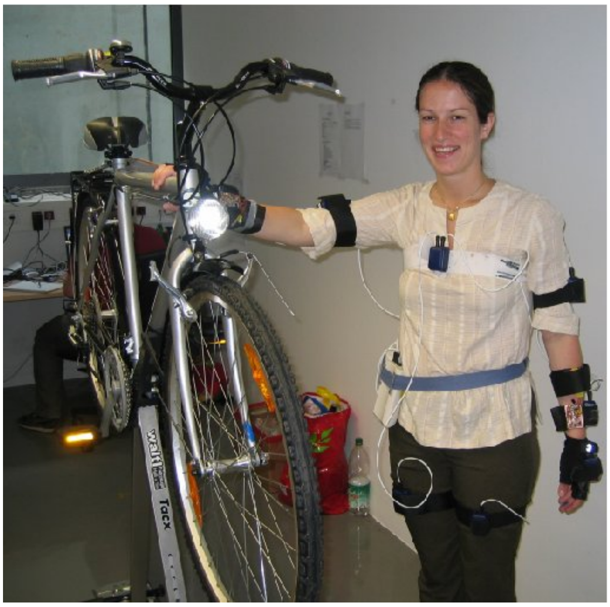


Figure 2.11: Activity from the bicycle repair experiment.

The experiment scenario is about repairing a bicycle fixed on a repairing stand. Figure 2.11 shows a picture from the data recording. It consists of 23 activities, which are shown in table 2.6. The set of gestures can be divided into activities with periodical motions and those without. A detailed description of the activities taken from [97] follows:

- *Pumping (gestures 1 and 2): Pumping begins with unscrewing the valve. Thus it consists of more than just the characteristic periodic motion. Pumping the front and the back wheel differs in terms of location, however, depending on where the valve is during pumping the location is rather vaguely defined. People tend to use different valve positions for the front and the back wheel, which means that statistically there is a difference in the acceleration signal as well.*

- *Screws (gestures 3 to 8): The sequence contains the screwing and unscrewing of three screws at different, clearly separable locations. Screw A requires a hexagon wrench key, screws B and C require an ordinary screwdriver. Combined with different arm positions required to handle each screw, this provides some acceleration information to distinguish between the screws (in addition to location information).*

- *Pedals (gestures 9 to 12): The set contains four pedal related gestures: just turning the pedals, turning the pedals and oiling the chain, testing the gear switch while turning the pedals and turning the pedals while marking buckles of the back-wheel with chalk. Pedal turning is a reasonably well-defined gesture.*

Table 2.5: Signal channels of the bicycle repair and the car workshop experiment.

nr	channel	nr	channel	nr	channel
1	right shoulder x	13	right wrist x	24	left hand tip z
2	right shoulder y	14	right wrist y	25	rsh angle front-back
3	right shoulder z	15	right wrist z	26	lsh angle front-back
4	left shoulder x	16	left wrist x	27	rsh angle side-body
5	left shoulder y	17	left wrist y	28	lsh angle side-body
6	left shoulder z	18	left wrist z	29	rel angle extend-deflect
7	right elbow x	19	right hand tip x	30	lel angle extend-deflect
8	right elbow y	20	right hand tip y	31	rwr angle radius-ulna
9	right elbow z	21	right hand tip z	32	lwr angle radius-ulna
10	left elbow x	22	left hand tip x	33	rwr angle back-palm
11	left elbow y	23	left hand tip y	34	lwr angle back-palm
12	left elbow z				

- *(Dis)assembly (gestures 13, 14 and 20 to 23): Among the gestures most difficult to recognize are assembly and disassembly of the pedals, the front wheel, and the back light. All of them can be performed in many different ways, while the hand seldom remains at the same location for a significant time.*

- *Wheel spinning (gestures 15 and 16): The wheel spinning gestures involve hand-turning of the front or back wheel. The gestures contain a reasonably well defined motion (the action of the spinning). However, there is also a considerable amount of freedom in terms of overall gesture. Front and back can be easily distinguished by locations. In most cases different hand positions were used for turning the front and the back wheel.*

- *Bell (gesture 17): Another challenging gesture is the testing of the bell. The time for ringing the bell up to five times is so short that only few location samples are available.*

- *Seat (gestures 18 and 19): These gestures alter the position of the seat. The first increases the seating position by twisting the seat within its mounting using both hands. In addition to the twisting, the degrading gesture requires the pounding with a fist to drive the seat into its mounting.*

The sensor setup consists of seven MTx sensors mounted on the chest, and both upper and lower arms and hands of the contestant. Table 2.5 shows the used signal channels in the later evaluation of our algorithm. The dataset was recorded with six different subjects over a total time of 404 minutes consisting of 1240 gesture instances which had a duration of 128 minutes. This dataset was published in [100, 115, 97, 99].

2.9.3 Car Workshop

The last dataset originates from a collaboration with Skoda derived from a real-life industrial scenario. It contains a set of activities where a quality service worker checks

Table 2.6: The 23 different classes contained in the bicycle repair dataset.

class	activity	class	activity
1	pumping (front wheel)	13	disassembling front wheel
2	pumping (back wheel)	14	assembling front wheel
3	blue screw (open)	15	turning front wheel
4	blue screw (close)	16	turning back wheel
5	red screw (open)	17	testing bell
6	red screw (close)	18	seat (up)
7	green screw (open)	19	seat (down)
8	green screw (close)	20	disassembling pedal
9	turning pedal	21	assembling pedal
10	turning pedal (chain oiling)	22	changing bulb (remove)
11	turning pedal (gear switch test)	23	changing bulb (insert)
12	turning pedal (wheel buckles)		

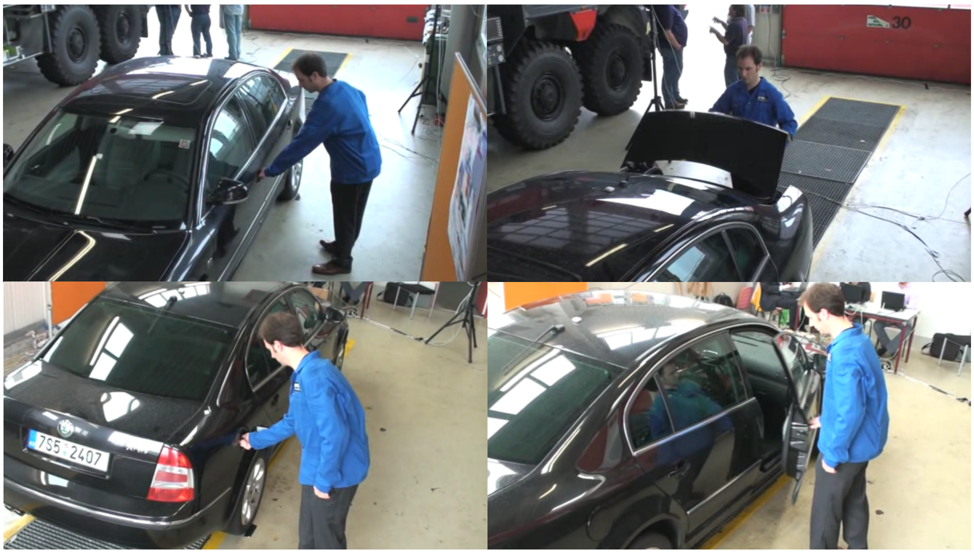


Figure 2.12: Activities from the car workshop experiment.

a newly built car after assembling for obvious mechanical problems. Figure 2.12 shows some exemplary pictures of the activities. This experiment has already been investigated thoroughly in activity recognition [101, 97, 116, 114] and is publicly available. The test subjects wear a body sensor network integrated into a jacket with 7 IMUs, with one each fixed on the worker's back torso, the upper and lower arms, and the hands. Table 2.5 shows the used data channels in the later evaluation of our algorithm. Nineteen different activities are present in the dataset as annotations, which are shown in Table 2.7. A detailed description of the activities was taken from [97]:

- *Opening and closing: These classes are short movements with a single hand without a significant body movement for the doors (class: 7-12) and bi-manual gestures with a significant body movement for the trunk, hood and the spare wheel box (classes: 1,2,3,4,18,19).*

- *Lock tests: Testing a door lock (classes: 15,16), the trunk fixation (class: 4), the fuel lid (class: 6) and the mirror fixation (class: 13) are repeated usually between three and five times, thus these gestures result in a repetitive pattern in both the motion and muscle signals.*
- *Gap tests: Testing chassis gaps (classes: 14,17) is a sliding movement of the fingers over the tested gap.*

Eight subjects were recorded getting about 12 hours of data in total. Each subject repeated the experiment procedure about ten times. Observers annotated the data during the recording.

Table 2.7: The 19 different classes contained in the car workshop dataset.

class	activity	class	activity
1	open hood	11	open two doors
2	close hood	12	close two doors
3	open trunk	13	mirror
4	check trunk	14	check trunk gaps
5	close trunk	15	lock check left
6	fuel lid	16	lock check right
7	open left door	17	check hood gaps
8	close left door	18	open spare wheel box
9	open right door	19	close spare wheel box
10	close right door		

2.10 Summary

This chapter gives an overview of the *invariants classification* algorithm for human gestures using wearable sensors. The algorithm is divided into two parts, the construction of the classifier model and the application of the classifier model. The model creation starts with the segmentation of the sensor data at the zero crossings of its central differential quotient. Then a bottom-up clustering on segments of several instances of a specific class identifies invariant segments. After the selection of the best clusters, we build three different invariant models: a template model as an average of the cluster members, an SVM based classification model, using features extracted from other sensor channels on a time window that corresponds to the invariant, and an activity position constraints model related to the temporal position of the invariant within the respective class signal. After this, we create the class instance model which fuses a certain number of invariant models and provides the final classification. Several parameters have to be trained for that. The last step in the model creation is to decide for each class if it is better to use the trained invariants classifier or a conventional classifier for the activity recognition.

In the application of the model, we distinguish between classes that can be handled by the invariants methods because they have significant invariants and other available methods perform significantly worse as determined in the last step of the classifier model building. Next, we segment the sensor data as before. Then we apply the class instance model which fuses the invariant models in the invariant detection. In the activity spotting step we estimate the class instance border from the found class segments using the activity position constraints model receiving a final classification result.

Next, we describe the method for evaluating the *invariants classification* algorithm. We introduce the three evaluation datasets: the *drink and work* scenario which has a healthcare context, the *bicycle repair* scenario, and the *car workshop*, both from an industrial background. All three experiments have IMUs in common. We explain how we calculate trajectories and angles of a human stick model from the sensor data. Trajectories and angle signals are then used in our classification algorithm. The final part is the detailed description of the three experiments.

3

Segmentation

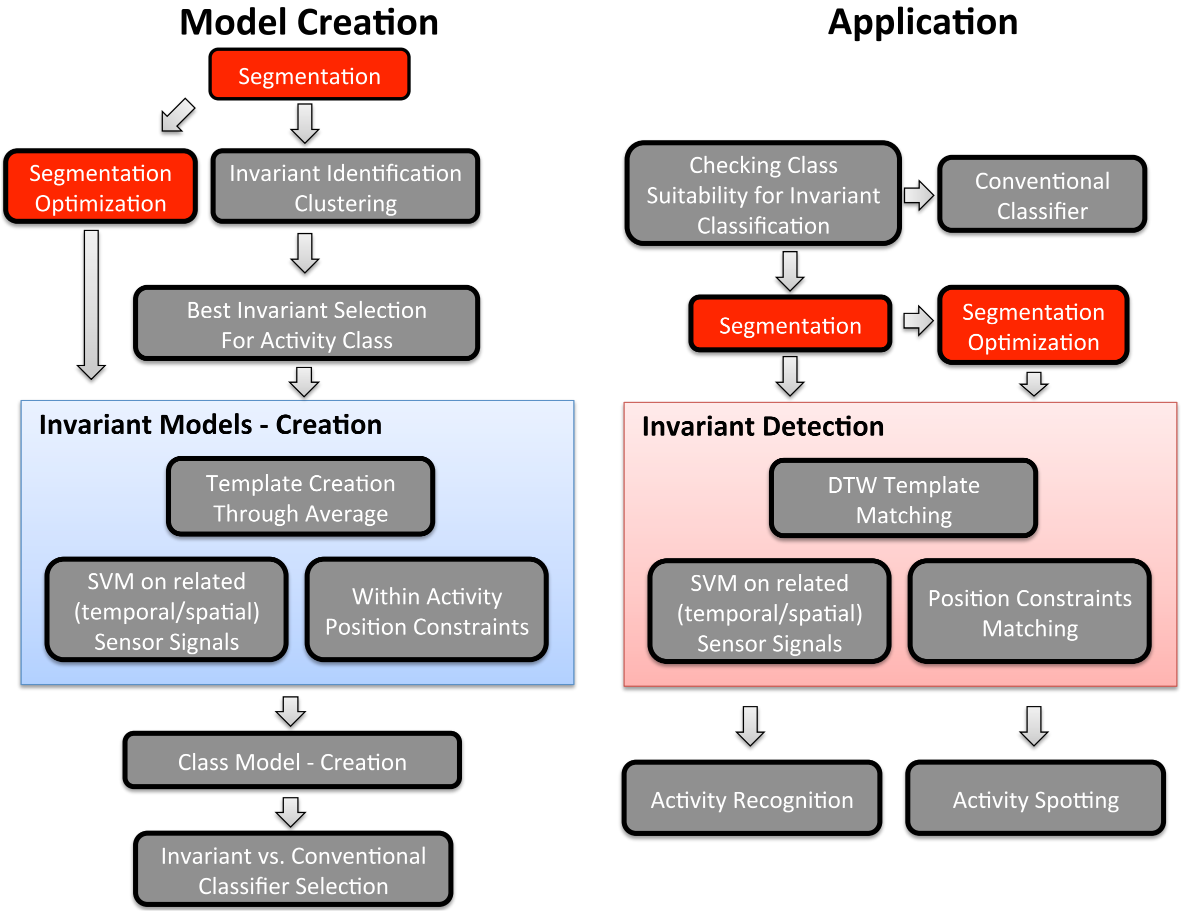

Model Creation
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Application
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 3.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter.

3.1 Introduction

The idea of segmentation is to form associated data subsets from the complete dataset, be it in the spatial or in the time dimension. This helps us later in the process to separate relevant data from irrelevant one because of the lower dimensionality compared to working with every single data point. In image processing, the segmentation tries to identify areas of interest in the picture. In activity recognition, using nonvisual sensors, the segmentation tries to separate important sensor signals from not important ones in the time domain. In computer vision, the segmentation of video data aims to locate relevant regions in the images over time. As our new algorithm is located in activity recognition, the goal is to split the continuous data stream which we receive from our sensors, in meaningful data packets in the particular time domain. Sensors record data with a much higher frequency compared to the duration of interesting human motions which means that several sensor data points can be merged. While understanding our dataset as time series, we can also see the separation as splitting the data in shorter time series which keeps the information we want to analyze and also lowers the dimensionality for the later processing by classification algorithms. At first sight, the perfect segmentation entity, in our case, covers a complete activity. The problem here is the variability of human activities which would result in a big variance in the segments of one activity class. The solution presented in our algorithm is to use a segmentation strategy where the data is segregated into sub-activities. These sub-activities are defined by the change of the movement direction on one of the three axes in 3D space. Thus we split complex activities into basic motions. These basic motions can be analyzed so that we find good representatives for the activity classes which are useful in the further classification.

In this chapter, we first show different options for segmenting data in the related work. Then we introduce the choice we made: cutting the continuous data where its central difference quotient has a zero crossing. The results of our basic segmentation strategy, applied to the data of our three evaluation experiments, are discussed afterward. Next we describe two extensions of our basic segmentation algorithm: first, we tried to combine adjacent segments with a low sum of central difference quotient values. This strategy was tested, but we did not stick to it as the segmentation quality was worse. The second way to improve our basic segmentation was to fuse adjacent segments to a new segment by using the knowledge we get from the clustering step of Chapter 4. Here adjacent segments can be merged if the coverage of train instances from a certain activity class stays nearly equal for a selected cluster. This segment fusion, called *segment optimization*, was later used in the invariant detection step. This chapter is based on [71, 72, 73].

3.2 Related Work

A straightforward strategy to segment a data stream is to use a sliding window of a fixed size. This is done by pushing the window forward by a fixed number of data points and using the data of each window frame for the next processing step in the classification chain, e.g. feature calculation. Several publications in the field of activity recognition implement such a method: In [125] two accelerometers fixed on the thigh were used to learn different human activities (e.g. sitting, standing, walking, running, climbing stairs, descending stairs and riding a bicycle) with low teaching effort for the user. In [35] sitting, walking, biking and riding the subway were classified with data of two 2D accelerometers in real time. In [9] the size of the sliding window is 512 samples with an overlap of 256 samples which is 6.7 seconds for each window, as the five biaxial accelerometers are sampled with 76.25 Hz. Twenty various everyday tasks were collected from 20 different subjects with five sensors attached to different body parts. In [54] daily routines are modeled and recognized without annotation. Two sensor boards were used, one was put in the right hip pocket, and the other was attached to the dominant right wrist. They recorded the real-time clock and a 3D accelerometer, with a sampling rate of 100Hz. The sliding window size was between 0.4 and 4 seconds. In [15] the spotting of low-level activities is used to detect daily routine activities like working or commuting. Here also two 3D accelerometers are recorded with 100Hz and features calculated with a sliding window of 0.4 seconds. In [10] the detection of difficult distinguishable hand motions like pressing different types of switches with wearable sensors is addressed. One problem in the sliding window approach is the fixation of the window size, which can negatively influence the classification result. In [55] beside of features, the window size is also investigated for activities like walking, standing and sitting, in combination with wearable acceleration sensors.

Another approach to segment time series is to linearly approximate the signal piecewise which satisfy a predefined error condition. In [63, 64] different existing approaches are shown. One central difference in these linear approximation algorithms is that either it can be calculated online while the data is coming in, or offline with the data completely known. Three methods are common for linear approximation: sliding window, bottom up and top down. The sliding window approach is capable of online calculation. It starts at a data point and continues with the linear approximation until an approximation error condition is met. Bottom-up and top-down are both offline algorithms. Bottom-up starts with the whole data stream separated in segments consisting of two points, then continuing to aggregate the segments with the lowest cost due to the linear approximation error until the error condition is met. Top-down starts with the whole data stream as one segment and divides at optimal points so that the linear approximation is best and continues with approximation until the error condition is met. The offline algorithms return better approximations than the sliding window algorithm. To get the advantages of both algorithm families, having a good approximation and an online algorithm, a fusion of these algorithms is proposed in [63, 64], calling it Sliding Window and Bottom-up (SWAB). It introduces a buffer for data points and applies the bottom-up approach in this buffer. When the bottom-up approximation stops, it removes the leftmost segment and gets new data into the data buffer, like the sliding window method which makes it an online algorithm with the approximation quality of the bottom-up method. In [3] the SWAB algorithm is applied to segment the motion of human nutrition intake recorded with four inertial measurement units (IMU) which were placed on the upper and lower part of the left and right arms. Bannach et al. [8] also implement the SWAB algorithm for segmenting the raw acceleration and gyroscope sensor data from an IMU mounted on the hand. The aim here is to detect different hand gestures in real time, to guide a car in a game into a parking space. In [58] the SWAB algorithm segments two different scenarios: one with everyday life activities and with IMU data from the trunk and the right arm, and the second with food intake activities and with IMU data from the trunk and both arms. All three previously mentioned papers incorporate a subsequent computational step where the found segments are compared to training segments in a similarity search. Features are calculated on the retrieved segments, and then a similarity score to the same features on the training segments is determined, e.g. by computing its Euclidean distance. On the basis of a threshold, it is decided if the segment is accepted or not.

Related to the linear approximation is the SwiftSeg algorithm, introduced in [42]. Here the segmentation is realized by piecewise approximating the time series with polynomials. Segment borders can be defined by using the approximation error or using average, slope, curvature, and change of curvature of the signal, which can be derived directly from the algorithm. The advantage of this algorithm is that it can be used online and as it uses an update mechanism for the polynomial approximation step, the computational cost is dependent on the polynomial degree but not on the length of the time window.

Another segmentation option relies on hidden Markov models (HMM). This is a stochastic model containing two layers: one with hidden states connected by transition probabilities and the second one with observable states which are dependent on the invisible states, described by a probability distribution. A detailed introduction to HMMs can be found in [108]. Two algorithms are important concerning HMMs: the forward algorithm and the Viterbi algorithm. The forward algorithm returns the probability of a sequence of observable states by calculating the result with recursions. The Viterbi algorithm provides the sequence of invisible states with the highest probability, given a sequence of observable states. Here also the calculation is done with recursion to avoid exponential complexity. The advantage of HMMs is that the variation of the signal in time is not a problem. In [92, 93] different hand gestures are spotted in a continuous video stream with HMMs. For every hand motion an HMM is trained, and then a normalized Viterbi algorithm returns the HMM output score for every time step. Now locating the peaks in these score data results in the end point of the activity class. In [143] left-right HMMs were combined with sliding windows to spot short activities with an IMU on the right wrist. This segmentation did not provide satisfying results.

We implemented a similar segmentation strategy as described in [143]. In this article, ten very short activities in a car context were recorded with an IMU on the right wrist. The idea for the segmentation here is that the movement of the hand slows down when gestures start and end. As a consequence, the variance signal of the hand movement is calculated with a sliding window of 10 samples. Whenever a minimum in the variance signal is reached, a new segmentation point is set. In [144] this strategy is tested on the data from the car workshop scenario (Section 2.9.3) with 5 IMUs connected in a body model providing 3D vectors. Blanke et al. [16] apply this segmentation strategy on three experiments again to evaluate different types of features for activity recognition.

3.3 Segmentation

3.3.1 Basic Segmentation

In our segmentation strategy, we want to split the continuous sensor data into sensible parts. This will separate the different human activities in a way which enables the following classification process. The sensors of our evaluation experiments provide us with orientation information of the upper body and its limbs. From this information, we construct a body model which gives us 3D trajectories for the torso, and the left and right body shoulder, elbow, wrist and hand tip (or a subset of the body position trajectories depending on the experiment). We also calculate several angles between two body limbs using the trajectory information which represent body joints: the shoulder joint with two angles (upper arm moving sideways from/to the body and the upper arm moving forward and backward), the elbow joint with one angle (extension - deflection of upper/lower arm), and the wrist joint with two angles (the hand movement in the direction of radius - ulna and the hand movement in the direction of the back of the hand - palm). To lower the dimensionality of the incoming data stream we have to split the data into packets which are better manageable and segregate the data related to their information. We are looking for human activities, so the most reasonable segmentation instances would be such human activities. One big problem in human activity recognition is the variability

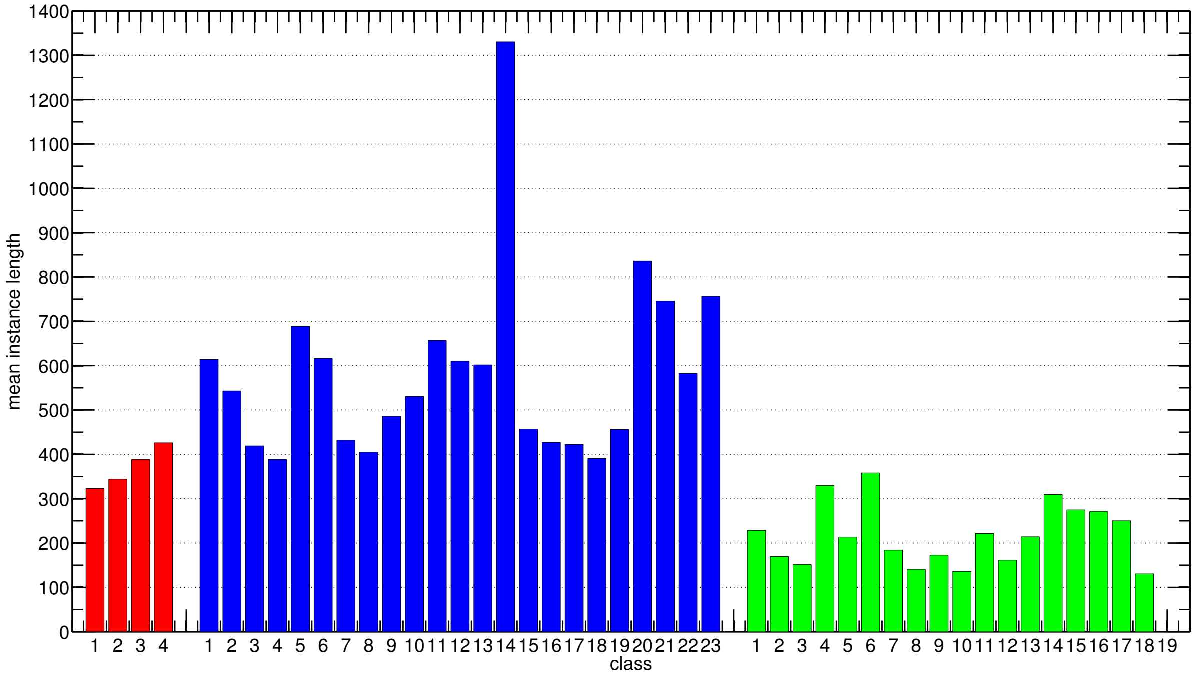


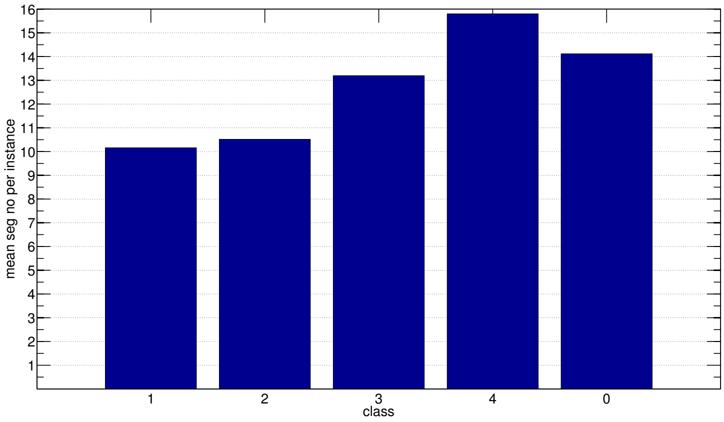
Figure 3.2: Mean length of instances in data points of the activity classes of all evaluation experiments. red: drink and work; blue: bicycle repair; green: car workshop.

in complex activities where the single steps can be interchanged, e.g. baking a cake and adding the ingredients in different orders. Short activities can also vary tremendously. There can be a difference not only in between the execution of the same activity by different subjects but also between the execution of the same activity by the same subject at different moments. If a person is performing a task while bored or tired or eager to finish it, the outcome can look quite different. To cope with this problem we decided to divide the data into sub-activities or “closed” motions. In our opinion a good border for such motions are zero crossings of the central difference quotient of the original raw data.

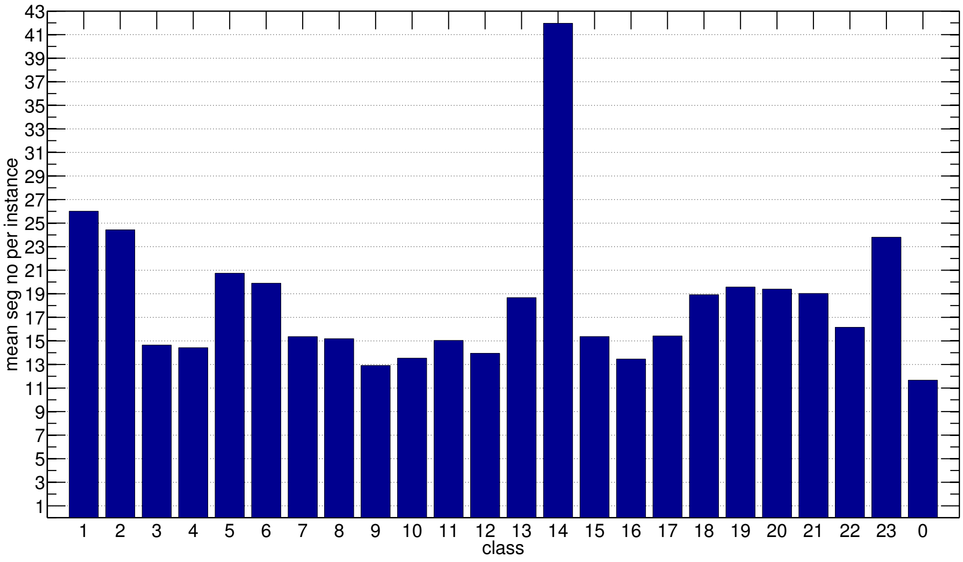
$$\frac{\Delta_y}{\Delta_x} = \frac{f(x+\frac{1}{2}\Delta_x) - f(x-\frac{1}{2}\Delta_x)}{\Delta_x} \tag{3.1}$$

In our case we use the trajectories of the body limbs and the angles of the body joints as raw data. The central difference quotient of the trajectories is equivalent to the velocity of the movement of the body limbs. The segmentation at the zero crossings of the velocity makes sense as these segment borders are located where the movement makes a direction change. The important factor here is the break which is a natural incision in the motion. Therefore we define the border of a sub-activity as pause or break in the movement.

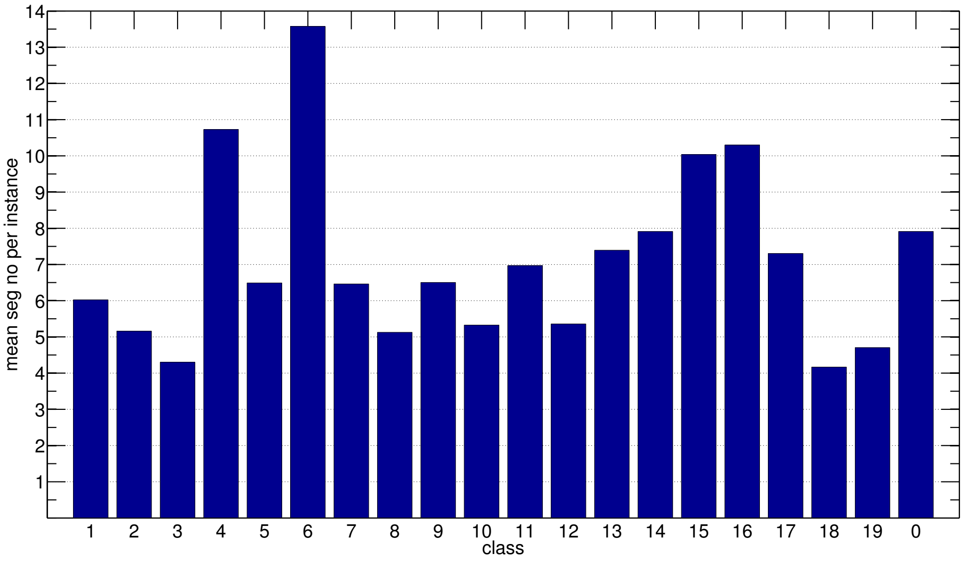
The more segments found in an activity, the more sub-movements exist, and the activity can be interpreted as more complex. Figures 3.3a, 3.3b and 3.3c show the mean number of segments per class instance for our evaluation experiments. The x-axis in the figures refers to the class numbers. These class numbers correspond to the activity classes in Tables 2.3, 2.6 and 2.7 from Section 2.7, where the evaluation experiments were introduced. In the *drink and work* experiment, *drinking from a glass* (class 1) or *drinking from a cup* (class 2) seem to be similar as the mean number of segments are nearly the same. *Drinking from a mug* (class 3) varies more from the perspective of found segments. The activity class where the subject drank from a bottle (class 4) returns the highest number of segments and seems to be the most different movement in this set of near related activities. In the *bicycle repair* scenario nearly all activities come in a pair



(a) Drink and work experiment.

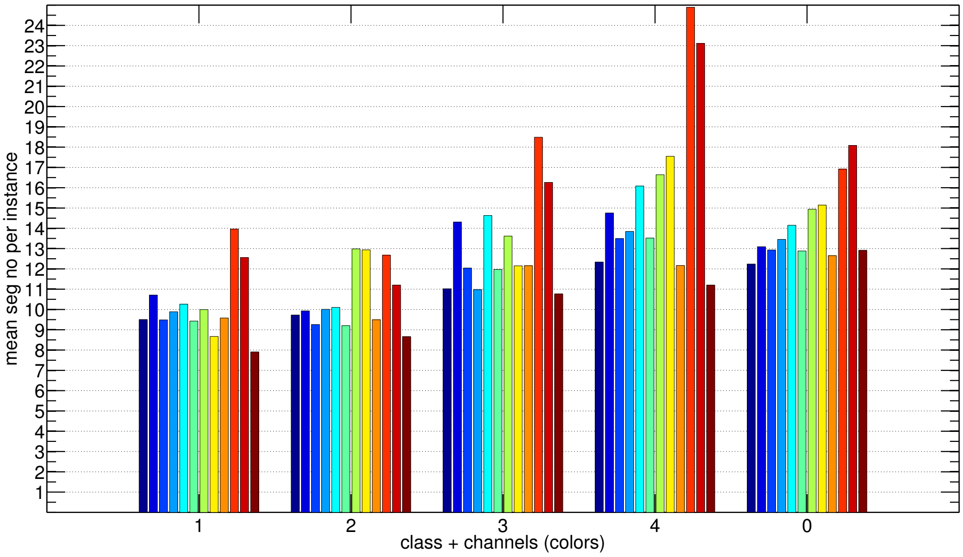


(b) Bicycle experiment.

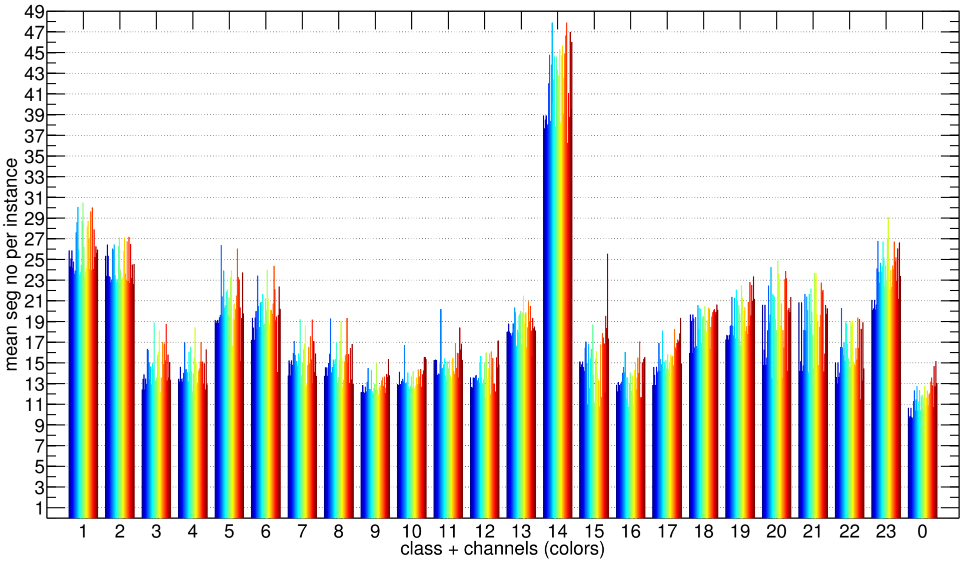


(c) Car workshop experiment.

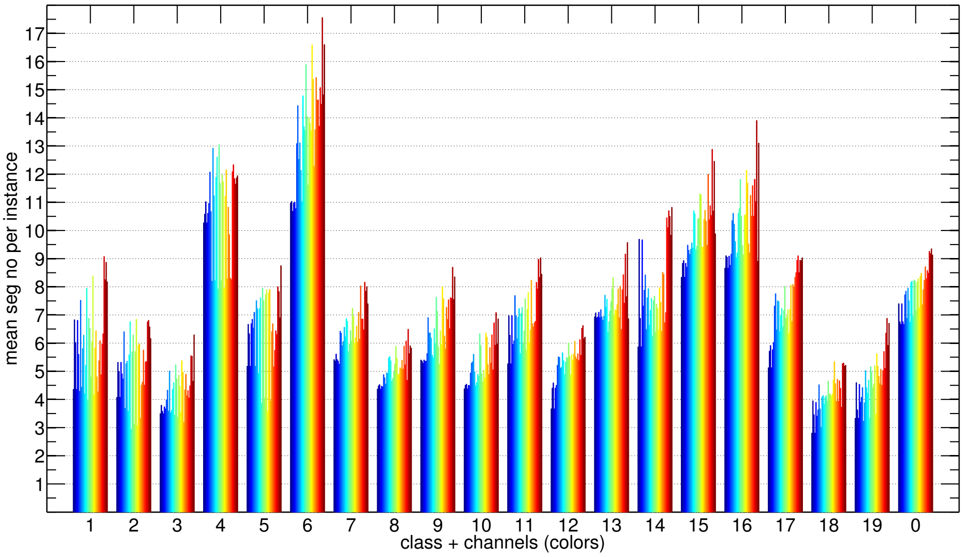
Figure 3.3: Mean number of segments per class instance.



(a) Drink and work experiment.



(b) Bicycle experiment.



(c) Car workshop experiment.

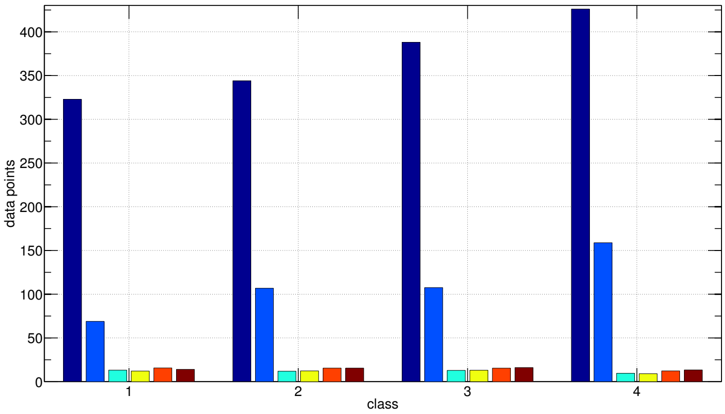
Figure 3.4: Mean number of segments per class instance for each signal channel.

consisting of an activity and its complement e.g. *loose screw* (class 3, 5, 7) and *tighten a screw* (class 4, 6, 8). Most of these activity pairs have a nearly identical mean number of found segments. Only class 13 *assembling front wheel* versus class 14 *disassembling front wheel* and class 22 *remove bulb* versus class 23 *insert bulb* show a notable difference in the mean number of segments. In [97] it is already mentioned, that these 2 activity pairs are "among the most difficult to recognize" as "all of them can be performed in many different ways, while the hand seldom remains at the same location for a significant time". This can explain the difference between the found segments due to the complexity of the activity. In the *car workshop* experiment again activity pairs, like *open hood* (class 1) versus *close hood* (class 2) or *open door* (class 7, 9, 11) versus *close door* (class 8, 10, 12), have nearly the same number of mean found segments. Similar activities like *open left door* (class 7), *open right door* (class 9) and *open two doors* (class 11) are even more alike. The two activities with the highest number of mean segments, *fuel lid* (class 6) and *open trunk* (class 3), contain both periodic movements. The fuel lid is turned several times into one direction, and the trunk door is seesawed several times to check if it opens smoothly. This explains the higher number of found segments. Out of the three scenarios, the *car workshop scenario* has the least number of found segments. Figure 3.2 shows the mean length of the activity class instances of all experiments. Red color indicates the activities of the *drink and work* experiment, blue of the *bicycle repair* experiment and green of the *car workshop* experiment. In general, the length of the instance correlates most of the times with the mean number of segments. The shorter the activity, the lesser sub-activities exist. That is why the *car workshop* activities have less found segments due to the general shorter activity instances. There is also a difference of found segment numbers between the different signal channels, depending on the body part (e.g. shoulder, elbow, wrist), the axis (e.g. x, y, z) and whether the signal is a trajectory or an angle. Figures 3.4a, 3.4b and 3.4c show the mean number of segments per class instance for all different channels of our experiments. Table 2.4 and table 2.5 show the signal channels with their description. The order of the channels is same as the order depicted in the plots.

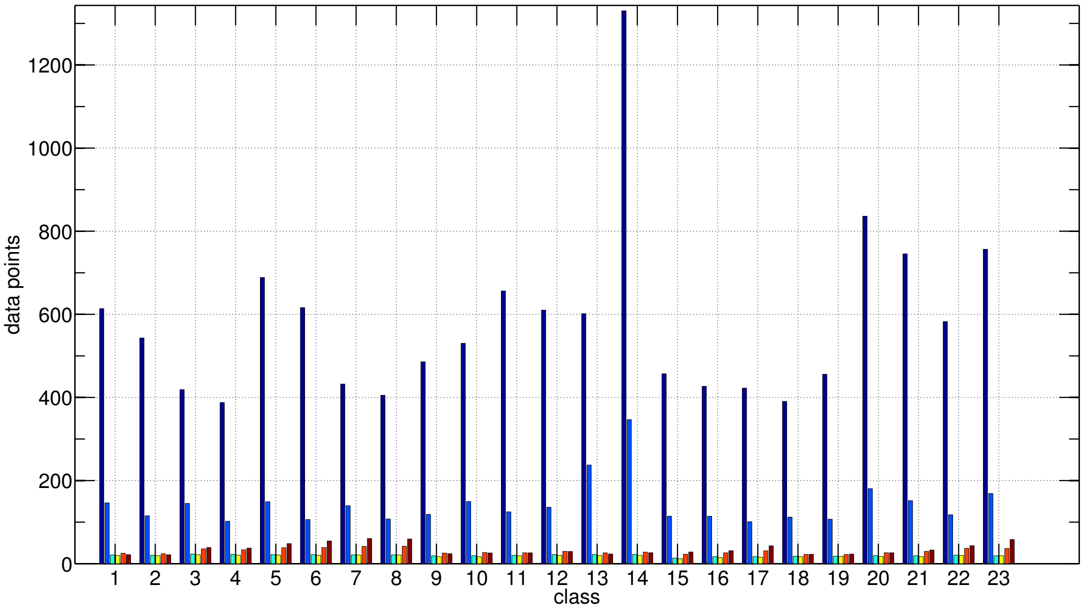
3.3.2 Low Velocity Segment Fusion

To refine the existing basic segmentation strategy which separates the data at the zero crossings of the central difference quotient, we merged the segments which have overall low difference quotient values. For our experiments, this means we want to aggregate segments where the subject does not move much. For example, if a person is standing still, there are yet some little motions or the sensor signal jitters a bit which can result in detected zero crossing of the velocity signal and therefore in segments. We evaluated this segmentation strategy against the basic segmentation strategy by checking the mean difference between found segment borders and the labeled class instance borders. We decided to stick with the basic segmentation as the results were better which we will describe in detail in the following paragraph after the introduction of the *low velocity segment fusion* strategy.

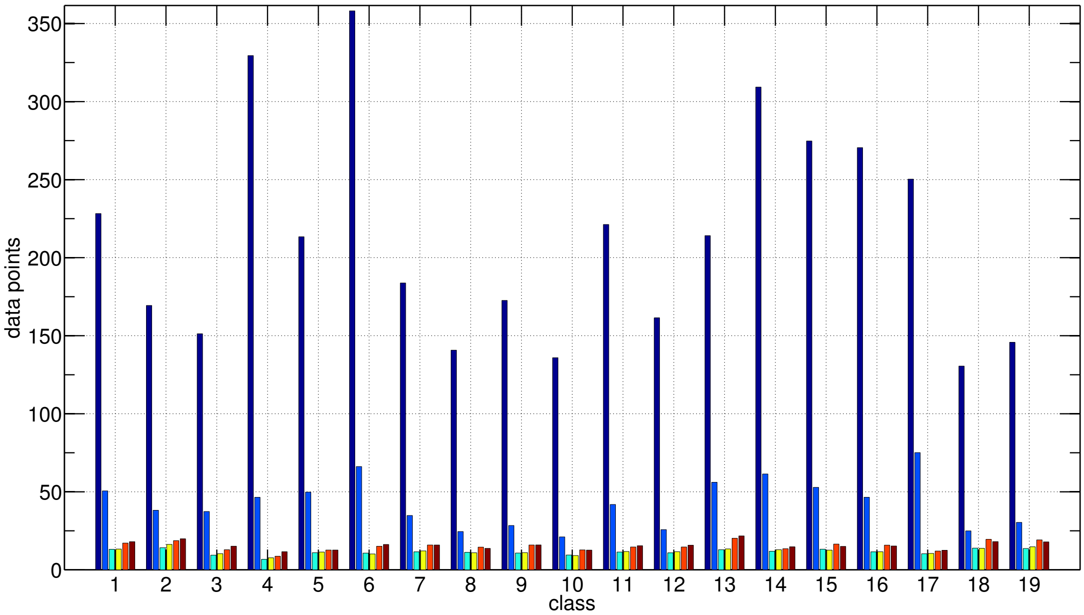
Using the result of the basic segmentation strategy, we start with the first found segment and sum up all central difference quotient values which are inside its borders. If this sum is smaller than a threshold then we start the fusion process with a neighbor segment otherwise we jump to the next segment and start from the beginning of the algorithm. We call this value *motion variance score*. The fusion of the segments works as follows: we calculate the *motion variance score* of the segment before and after and unite it with the one which has the lower *motion variance score*. In this way, we add the



(a) Drink and work.



(b) Bicycle repair.



(c) Car workshop.

Figure 3.5: dark/light blue: mean/SD of class instance length; turquoise/yellow: mean/SD of instance - segment difference with *basic-segmentation*; orange/red: mean/SD of instance - segment difference with *low velocity segment fusion*.

segment with the low central difference quotient values with the one which is the most consistent neighbor segment. We tested several threshold values and chose 0.018 as our best threshold.

To decide for the better strategy, we took the labeled class instances (the class instances were labeled during the recording of the experiments) and calculated the difference between the start and the end of each instance, and its nearest segment. Then we computed the mean and the standard deviation (SD) of all these differences for each activity class. To relate these differences to the length of the class instance, we also determined the mean and SD of the length of the labeled instances for each class. In the plots 3.5a, 3.5b and 3.5c, we see the results of the *drink and work*, the *bicycle repair* and the *car workshop* experiment. The dark and light blue bars show the mean and SD of the class instance lengths, the turquoise and yellow bars indicate the mean and SD of the difference between instance and segment border using the basic segmentation strategy, while the orange and red bars show the mean and SD of the difference between the instance and segment borders using the low-velocity segment fusion strategy. In all experiments, the mean and SD values of the low-velocity segment fusion strategy are higher than the basic strategy. This means that the segment borders of the low-velocity segment fusion strategy are in general further away from the labeled activity class instance borders than the segment borders of the basic strategy. So we decided to stick with the basic strategy, as it is also less complicated and does not need any parameter. In the case of an online algorithm, the low-velocity segment fusion strategy means an additional complication as we need to know one segment in the future forcing us to work with a segment buffer. For the identification of typical segments which can describe a class instance, this fusion strategy does not bring advantages because we use a clustering algorithm where only good and descriptive segments are kept. Short segments without significance are sorted out, irrespective of whether they are separated or fused. In the next section, we introduce another method to combine segments.

3.3.3 Segment Optimization

The segments we find, using the basic segmentation strategy, are relatively short. In the next step, we fuse adjacent segments to form a more descriptive activity snippet, when a certain condition is met. This condition originates from the clustering process where we try to find descriptive, invariant segments for each activity class instance (explained in chapter 4). The idea now is to try to extend all member segments of a good cluster with adjacent segments. The set of good clusters is first preselected after the clustering (Section 4.4) and then trained in the *class instance model* creation (Section 6.3). The clustering is executed on all segments of the training instances for each activity class separately. When a good cluster is chosen, we can see the number of training instances included by checking the instance affiliation of all cluster member segments. This is called *instance coverage*. We decide about the extension of cluster member segments by comparing the *instance coverage* of the original cluster member segments with the *instance coverage* of cluster member segments after the segment fusion. The best chosen segment extension is then applied in the later creation of the *class instance model* and in the application of the *class instance model* which is the final classification.

In detail, we take four rounds, where we add a segment before, a segment after, and both segments before and after the starting cluster member segments. In the first round the starting segments are the original, unaltered member segments of the cluster and in the other three rounds, the starting segment is always the starting segment from the round before with the fused segments before and after. For each segment fusion step, we calculate pairwise the dynamic time warping (DTW) similarities of the new segments and find the new cluster center, which is the segment with the highest average DTW similarity to all other segments. The quality criteria for the segment extensions is then the class *instance coverage* in comparison to the original cluster. For this purpose, we calculate the *instance coverage* of all fused segments which have a DTW similarity to the new cluster center less than or equal to the maximum DTW similarity between member segments and the cluster center of the original cluster. We choose the segment extension parameters with the most fused segments and having at least 80% of class *instance coverage* of the original cluster. If there is no 80% class *instance coverage*, we stay with the original cluster segments which were not fused. For the following classification, we know now how many segments should be fused to get a descriptive segment of the activity. We use this information in the invariant detection in Chapter 5.5.

3.4 Summary

In this chapter, we showed different strategies to segment the continuous data stream from the sensor readings. First already known segmentation strategies were introduced to show the status quo. Then the segmentation algorithm, used in our algorithm, was explained. As we try to classify human activities, the best way is to segregate our signals into units which correspond to these human activities. A big problem in human activities is the variability of their conducting. To cope with this problem we decided to take sub-activities, consisting only of short, "closed" motions, as segmentation unit. The motions are best separated by zero crossings of the central difference quotient of the raw data. In our evaluation experiments, we work with trajectories and angles of the upper body including the upper body limbs. In this case, the central difference quotient can be seen as the velocity of the body part and a zero crossing as the change in the direction of the movement. In our basic segmentation strategy, we cut the segments every time when the direction of the movement changes. In this way, we segment every data channel separately, e.g. the x-axis of the trajectory of the hand movement. The next idea to advance our segmentation is to combine segments with a low sum of all central difference quotient values which we call *low velocity segment fusion.* We decided against this because the basic segmentation delivered more accurate segment points in comparison to the labeled class instances than the *low velocity segment fusion.* The last section describes another way to optimize our basic segments. We use knowledge from the following step in our algorithm: the search for the best descriptive, invariant segments for an activity instance called *invariant identification.* For this purpose, we cluster the data of segmented class instances and get a set of segment clusters. Next, we try to extend the cluster members of a cluster by fusing them with adjacent segments. If all segment members of this new cluster cover at least 80% of train activity instances as the original cluster member segments, the segment combination is accepted and used in the further classification chain.

4

Invariant Identification - Clustering

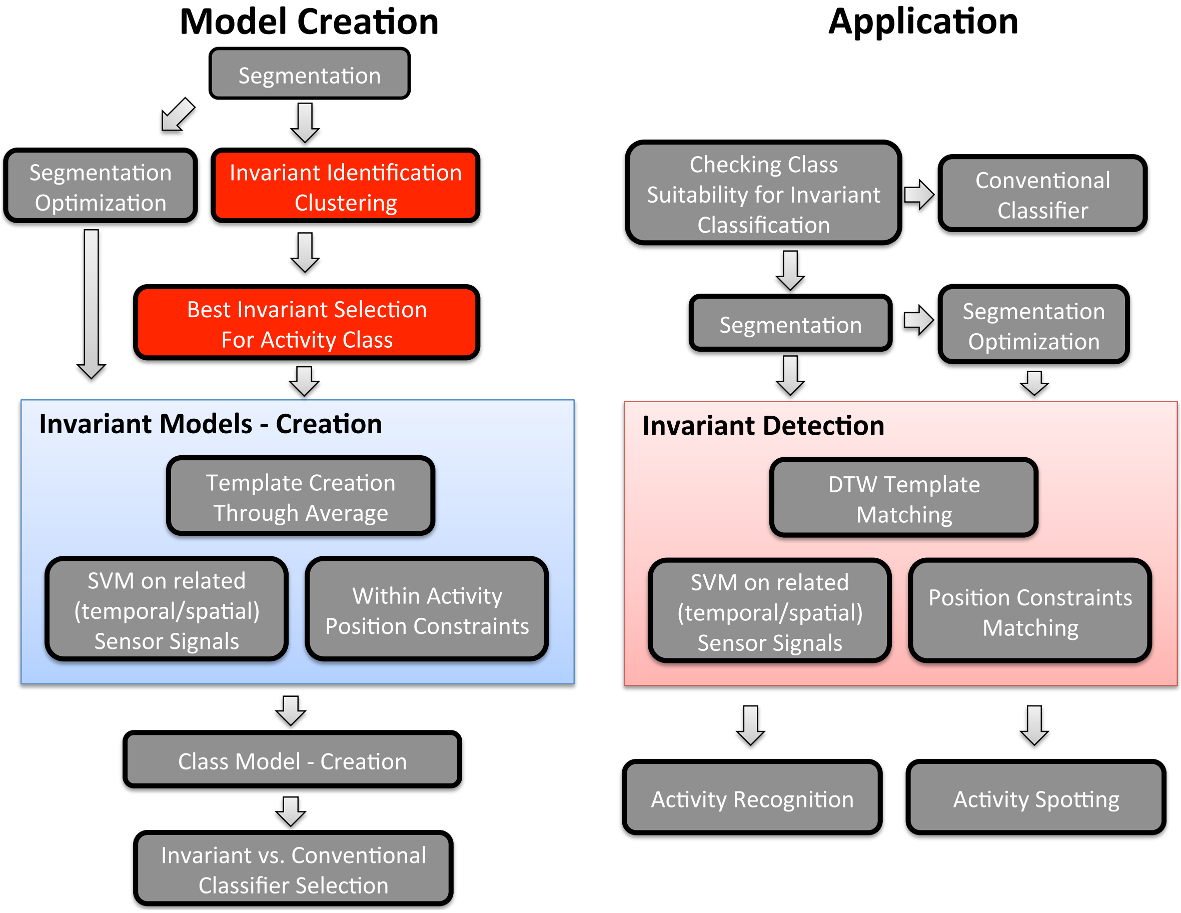

Model Creation
Application
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 4.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. The red marked step is explained in this chapter.

4.1 Introduction

One central problem of classifying human activity is its variability in the execution. Our solution to this problem is to identify sub-activities which are similar due to physical precondition. This identification process is realized with a clustering algorithm, described in detail in this chapter. The result of this *invariant identification* are clusters consisting of signal segments. We call these clusters *invariants* based on their member segments which occur in the optimal case in an invariant form in all class activities. The clustering returns further advantages for our activity recognition task considering the selection of sensor signals. The sensor network which we use to obtain signals of human activity provides high dimensionality of resulting data. While a high number of sensors gives the possibility to detect many facets of human activities, it also makes the activity recognition more complex. When we identify the most relevant sub-activity, we check all different sensor channels. This means that each identified sub-activity is connected to one sensor signal. So when we use a certain sub-activity for the further classification, we only have to analyze the connected sensor channel which lowers the large amount of sensor data considerably. Furthermore, we introduce a heuristic derived from the cluster results. This enables us to know at an early stage which classes tend to be easy classified and which are more prone to be mixed up with other classes due to their similar invariants. This heuristic can also be applied on the signal channels, showing us which channels provide in average a high number of discriminable invariants for all activity classes.

In this chapter, we first introduce the related research in the context of cluster algorithms. Then we introduce the clustering strategy which we use to find the important signal snippets to represent our activity classes. Next, we show an additional heuristic called *cluster precision* which enables inspection of the high-dimensional data at an early stage in the activity recognition process. Through this, we obtain an early insight into which sensors and features can be expected to perform well in distinguishing the different target classes. This chapter is based on [71, 72, 73].

4.2 Related Work

Clustering is an unsupervised machine learning approach which tries to form groups of similar objects. In our algorithm, we use a clustering algorithm to group similar sensor signal segments of a certain activity class for the identification of activity representatives. In the beginning, we tried a k-mean cluster algorithm [49] but then changed to a hierarchical clustering [57]. The reason was that we look only for a small number of clusters in each clustering. The hierarchical clustering supports this because we can follow the construction of the clusters and test each partial result if it is a good representative cluster. In the clustering process, we compare different signal data with each other. For this purpose, we need a measure to compare two different time series [36]. The most common measure is the Euclidean distance [38], besides others like the related L_p norms [140]. In our proposed algorithm we choose the result of the dynamic time warping algorithm [13] as similarity measure. This algorithm is typically used in speech recognition and is able to compare two signals with different length. This is especially important not only when two equal words are spoken in a different speed but also in the case of human activity, where the time of the execution of equal activities varies most of the time.

In activity recognition, there are different fields of applications of cluster algorithms. In [125] Kohonen self-organizing maps are used to cluster the sensor data. This helps to realize a system for context awareness with a low effort in the training phase. Krause et al. [69] also use a Kohonen self-organizing map to build up a context-aware system without supervision. Here the codebook vectors of the self-organizing map are clustered with k-means clustering, where the number of clusters is derived from the Davies Bouldin index [33]. In [136] k-mean clustering is used for the vector quantization of body posture features. The 3D positions of the joints of a human body are identified using Microsoft Kinect. After reducing the dimensionality with linear discriminant analysis, the resulting feature vector is then transferred into a symbol with the aforementioned k-mean cluster algorithm. Each frame of the activity sequence is then represented by a symbol of the body posture vocabulary. The words generated for the different activities are then processed for the final classification with HMMs. In [110] trajectories are clustered to identify important regions of activity in a video stream. The trajectories are extracted from the video and then clustered with an agglomerative hierarchical graph cluster algorithm [120]. The adequate number of clusters was determined using Cattell's scree test [23]. Kwon et al. [78] utilize unsupervised learning algorithm to overcome the use of a training set. The activity set consists of walking, running, sitting, standing, and lying down. The sensors of a smartphone record the human motions. First, the clustering is performed with the known number of clusters using k-means clustering, mixture of Gaussian, and average-linkage hierarchical agglomerative clustering. After that, the same algorithms are tested again without knowing the correct number of clusters. The detection of the number of clusters is then based on the Calinski Harabasz index [22].

In our used cluster algorithm we did not explicitly chose the best number of clusters. However, some clustering algorithm require the decision when the clustering has reached its optimum. Typical measures are the Dunn index [37] and the Davies-Bouldin index [33].

4.3 Cluster Algorithm

The basic concept of our new *invariants classification* algorithm is that there exist sub-activities which are typical and similar in each instance of a specific activity class. The first idea was that maybe the begin and the start of an activity instance could be such an essential part of the activity. Therefore we tried to use only a small signal frame at the start and at the end of the activity in a first test. However, this did not seem flexible enough. So we came up with the plan of first separating the sensor signals into meaningful segments and then searching for a segment inside the complete activity instance which describes the activity well. For this purpose the clustering algorithm is the method of choice: it groups all segments of the training data of the activity class instances. The next step is to choose the cluster with the best descriptive segments, which means that the segments have to be similar and member of preferably all train class instances in a small number. We tried first to work with the k-mean algorithm. The problem here was that we were looking only for a subset of clusters. All clusters with activity segments which are not descriptive are nonrelevant. So choosing the quantity of clusters beforehand is difficult. Thus we changed to hierarchical clustering with average linkage because here we can trace the cluster construction and check every new cluster if it is the one we are looking for. Next, we explain our cluster approach in detail.

The clustering is done on all segments (training data) belonging to the instances of one specific activity class and one sensor channel. To measure the similarity between segments, we use dynamic time warping (DTW). First, we get the DTW similarities between all segment pairs, and then we calculate the cluster tree with average linkage, which means that for determining the distance between two clusters we use the mean pairwise similarity of all members of the two clusters. While constructing the clusters bottom-up, we keep all evolving clusters, shown in Figure 4.2. Distinctive to a standard cluster algorithm, we do not decide how many clusters we expect or when the clustering has found enough clusters, but just keep all evolving clusters while building up the cluster tree. The reason is that in this way we do not miss any important cluster and we do not have to decide for a fixed number of clusters or where to cut the cluster tree. This would be difficult because only a few clusters contain the descriptive segments. All other clusters are not important due to their variety of segment members. By keeping all possible clusters, we end up with a big set of clusters.

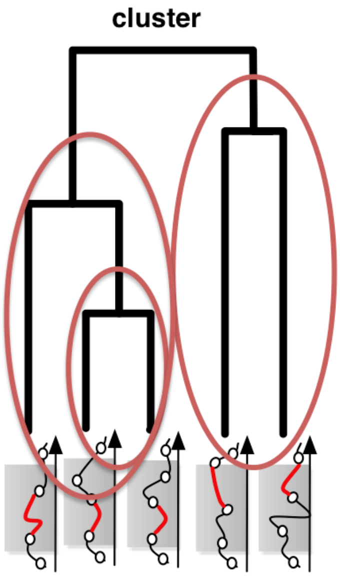


Figure 4.2: The cluster tree (dendrogram) and all possible clusters, represented by the red ellipses.

4.4 Best Invariant Selection

To reduce the big amount of possible clusters, we first remove all clusters which cover less than 20 percent of all activity instances of the class. Then we remove all sub- and supersets in the cluster list so that only clusters with diverse segment members are used for further steps. For this purpose we sort all clusters by the sum of the order of three features:

1. percentage of covered instances
2. number of covered instances divided by number of covered segments
3. standard deviation of the start position in the instance of all cluster segments

For the first two features the order is: the higher, the better; the last one has the contrary order. The percentage of covered instances ensures that the important segment can be found in most class instances, the perfect value would be 100 percent. The number of covered instances, divided by the number of covered segments, secures that the important segments are not found too often in a class instance with a perfect value of one, meaning that for each class instance only one segment is in the cluster. The standard deviation of the start position in the instance of all cluster segments guarantees that the start position of the important segment is more or less always the same in each class instance. Here the perfect value is zero, showing that the percentaged start position of the segment in the class instance is everywhere the same.

Next, the ordered clusters are sequentially treated by removing all sub- and supersets until no more clusters exist with overlap in member segments. Table 4.1 shows the number of clusters per channel and class for the *drink and work* experiment before and after sub and superset removal. In Table 4.2 we see the number of clusters for each class summed up over the channels of the three evaluation experiments. By using this reduction step, we can lower the number of clusters between 15 and 25 percent of the original number.

Table 4.1: Number of clusters for every class and channel before (white) and after (gray) sub-/superset removal of the *drink and work* experiment.

channel/class	1		2		3		4	
1. x shoulder right	59	17	52	12	75	20	82	21
2. y shoulder right	80	17	62	14	89	16	109	24
3. z shoulder right	63	17	51	12	63	15	94	22
4. x ellbow right	62	12	53	12	68	17	100	18
5. y ellbow right	73	14	63	12	77	15	104	21
6. z ellbow right	67	15	56	15	62	16	92	19
7. x wrist right	56	13	85	13	77	16	105	22
8. y wrist right	61	10	73	14	75	17	106	25
9. z wrist right	60	15	52	14	72	17	83	22
10. rsh angle front-back	88	20	73	18	100	18	205	33
11. rsh angle side-body	82	16	62	13	94	19	179	32
12. rel angle extend-deflect	59	13	58	12	63	15	72	15
$\sum$	810	179	740	161	915	201	1331	274

Table 4.2: Number of clusters for every class before (white) and after (gray) sub-/superset removal of all three evaluation experiments.

experiment/class	1		2		3		4		5		6		7		8		9		10		11		12	
drink and work	810	179	740	161	915	201	1331	274																
drink and work, one vessel	903	168																						
drink and work, one vessel sip	431	83																						
bicycle repair	2705	635	2363	524	1514	309	1371	260	1544	297	1413	264	1537	308	1486	279	1819	444	1725	388	2560	666	1745	387
car workshop	1490	333	1301	285	1068	225	2203	495	1649	358	2688	680	1494	336	1143	261	1465	354	1178	277	1642	371	1310	284

experiment/class	13		14		15		16		17		18		19		20		21		22		23		$\sum$	
drink and work																							3796	815
drink and work, one vessel																							903	168
drink and work, one vessel sip																							431	83
bicycle repair	2656	567	5736	1325	1964	430	1713	365	1383	261	3531	837	3606	854	2807	739	2471	608	2113	414	2686	562	52448	11723
car workshop	1224	249	2054	455	1767	403	1866	464	1607	356	1037	222	1131	256									29317	6664

4.5 Cluster Precision

Additional to the task of finding invariant segments for the further classification, the clustering also gives an instant insight into sensor signal quality and the danger of activity class' confusion. This information can be easily represented graphically for visual inspection, prior to any classification evaluation. Figure 4.3 shows the central plot of the values which lead us to this insight. We call this measure *cluster precision*. Each quadratic subplot corresponds to one sensor channel (trajectory/angle of shoulder, elbow, wrist or hand tip). The axes' label indices represent the different activity classes of the three evaluation datasets. The subplots are symmetrically divided by the diagonal white line of the *cluster precision* value of pairs of the same class. The *cluster precision* value between two class pairs shows the similarity between the invariant template segments found for these two classes which reflects the difficulty to separate the two classes from each other: Bright colors show high inter-class invariant similarity, which explains the white diagonals of pairs of the same class, and dark colors a low similarity. Knowing this, it is possible to identify channels which tend to mix up different invariant template segments from those which have a good class distinctiveness. It is also possible to see for single channels, which classes' data tend to be close to each other and might get easily mixed up.

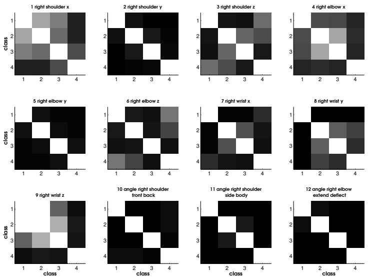
The *cluster precision* is calculated in the following way: The invariant template segment is identical to the cluster center, which is the cluster member segment with the highest average DTW similarity to all other cluster member segments. For each invariant template segment of one channel and one class, we look for the most similar invariant template segment of another class on that channel by using the DTW similarity. If we have found such a pair, we remove it from both of the lists of invariant template segments of the two specific classes. This is done until one list of invariant template segments of the two classes is empty. The sum of all pairwise invariant template segments' similarities divided by the quantity of these similarities provides the *cluster precision* value. The quantity of the invariant template segments' similarities is equal to the smaller invariant quantity of the two compared classes.

$$clusterPrecision_{class1-class2} = \frac{\sum_{n=1}^{N} similarities_n}{N} \tag{4.1}$$

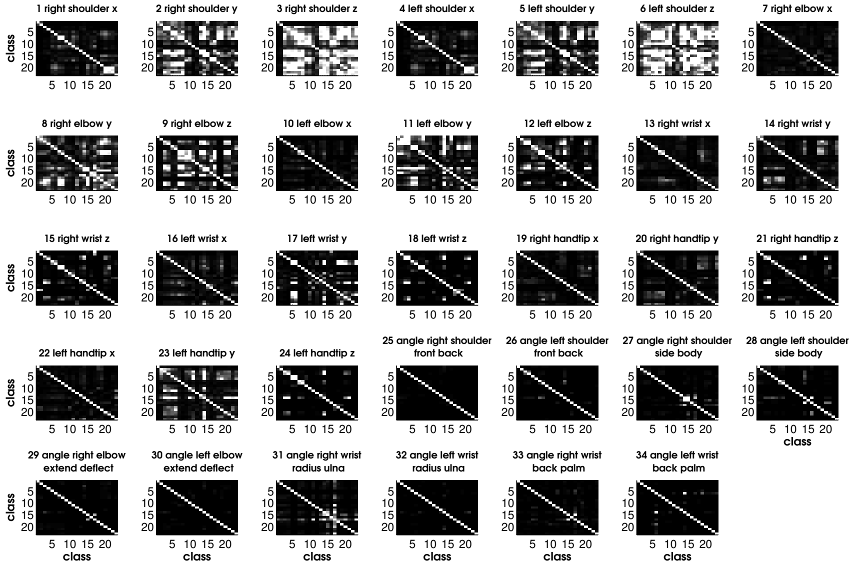
where $N = min(no.of.invariants_{class1}, no.of.invariants_{class2})$ and $similarities$ are the DTW similarities between the invariant template segment pairs. The higher the value, the more dissimilar are both classes on this channel. Additionally, we can sum up all *cluster precisions* of one class over all channels which shows the average similarity of invariant templates of this specific class compared to all other classes. Furthermore, the sum of all *cluster precisions* of one channel shows the average similarity of the invariant templates of all classes in this specific channel. In this way we can already tell the quality of signal channels and how difficult it will be, to classify classes before the classification has started.

Later in the evaluation of our classification algorithm, we use a leave-one-out cross-validation. Therefore, for the final evaluation of the *cluster precision*, we also use a leave-one-out cross-validation. One subject is always spared for testing while the other subjects are first segmented and then clustered, as described before. For the *cluster precision* we need only the training dataset because it is directly calculated after the clustering. Having n subjects in an experiment we get n *cluster precision* results. For the final result, we calculate the mean of these n *cluster precisions*.

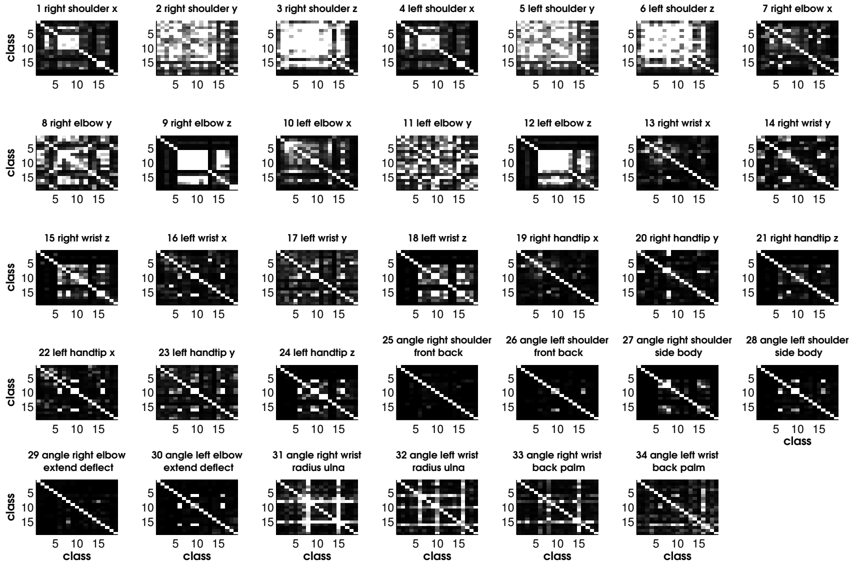
Figure 4.3a depicts the *cluster precision* value for each channel of the *drink and work* experiment. Bright areas indicate high inter-class similarities, whereas darker areas in-



(a) drink and work experiment.

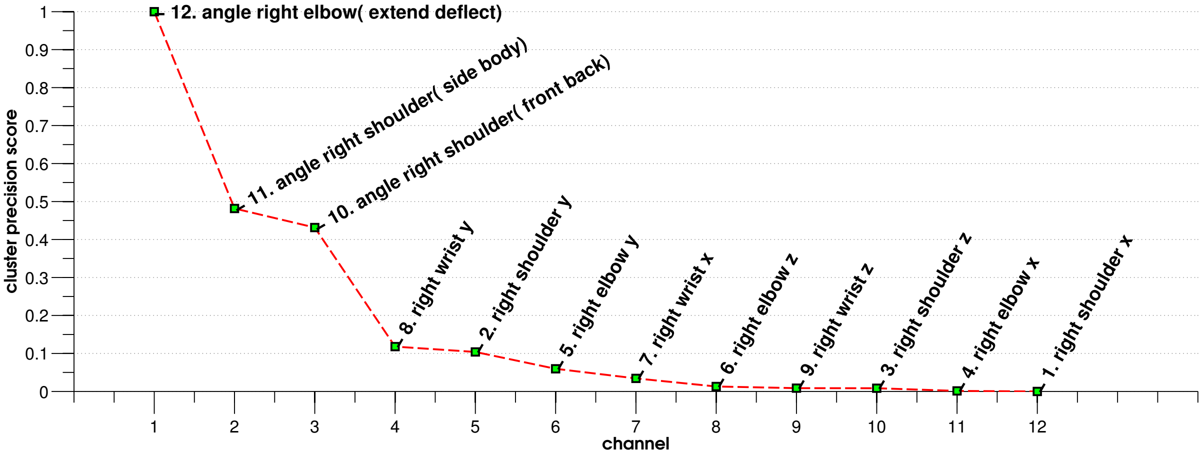


(b) bicycle experiment.

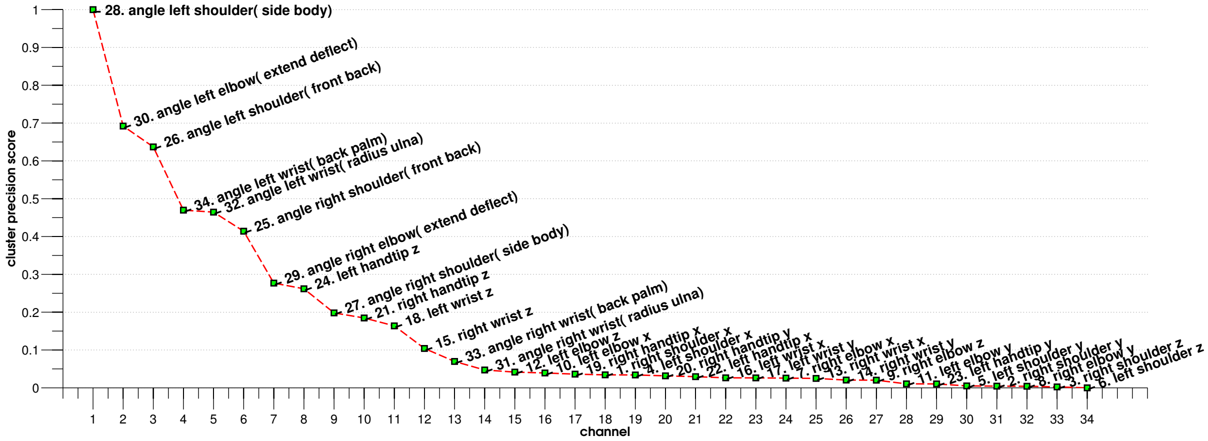


(c) car-workshop experiment.

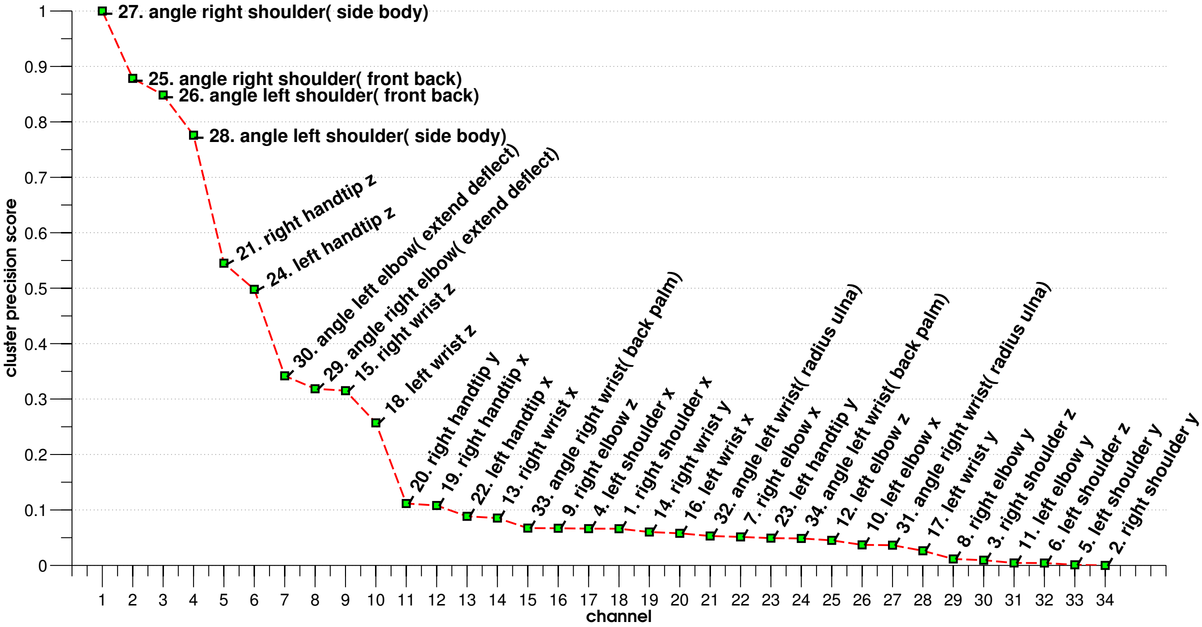
Figure 4.3: Visual representation of cluster precision results for all sensor signals (trajectory/angle of shoulder, elbow, wrist or hand tip) over all classes of the three experiments; Bright areas indicate high inter-class similarities, whereas darker areas indicate low inter-class similarities.



(a) drink and work experiment.

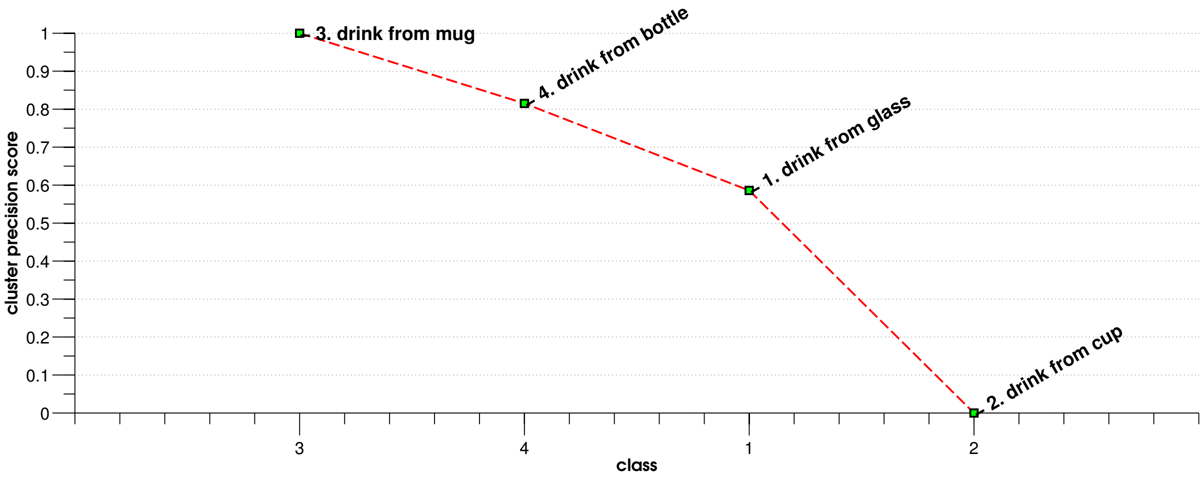


(b) bicycle experiment.

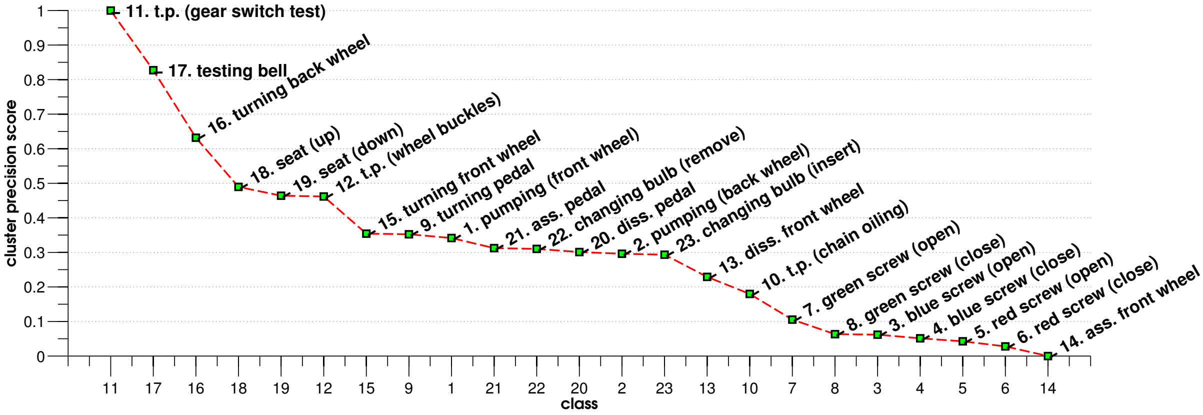


(c) car-workshop experiment.

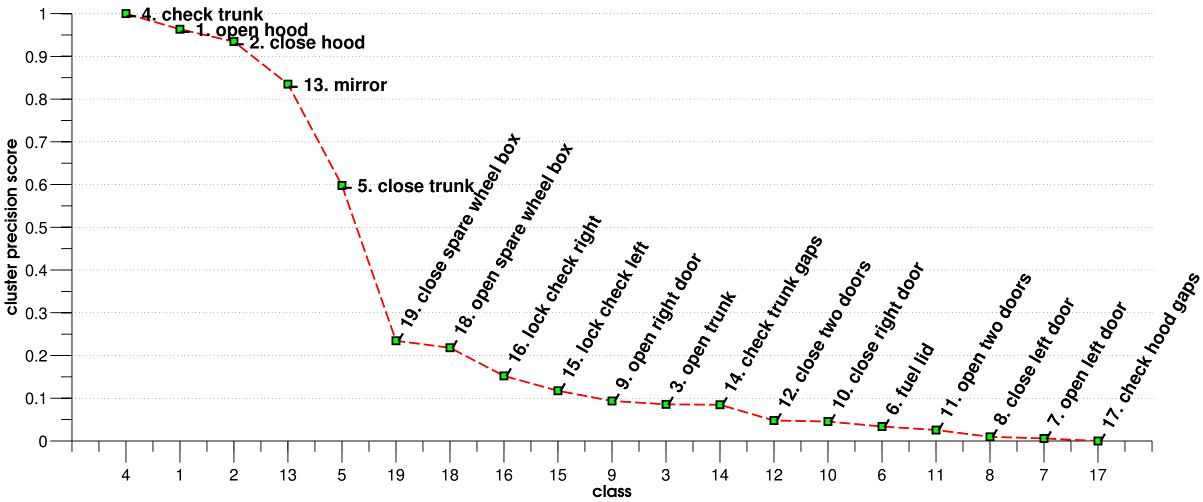
Figure 4.4: Cluster precision: best channels of the three experiments. High values of the *cluster precision* sum indicate low inter-class similarities.



(a) drink and work experiment.



(b) bicycle experiment.



(c) car-workshop experiment.

Figure 4.5: Cluster precision: best classes of the three experiments. High values of the *cluster precision* sum indicate low inter-class similarities.

Table 4.3: Best class order (1 = best, 23 = worst) by cluster precision (Figure 4.5b), best *invariants classification* result with sweep over all parameters (Figure 6.5), *invariants classification* with known class instance borders (Figure 6.11) and *invariants classification* with activity spotting (Figure 7.7) of the bicycle repair experiment. The colors are connected to the activity classes (class 11 = dark red ...) in the best class order of the cluster precision.

classifier / best class order	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
cluster precision	cl.11	cl.17	cl.16	cl.18	cl.19	cl.12	cl.15	cl.9	cl.1	cl.21	cl.22	cl.20	cl.2	cl.23	cl.13	cl.10	cl.7	cl.8	cl.3	cl.4	cl.5	cl.6	cl.14
all possible results	cl.12	cl.16	cl.19	cl.14	cl.15	cl.10	cl.11	cl.5	cl.18	cl.9	cl.22	cl.13	cl.6	cl.21	cl.20	cl.1	cl.17	cl.3	cl.2	cl.23	cl.4	cl.7	cl.8
results with given inst borders	cl.16	cl.12	cl.15	cl.10	cl.18	cl.19	cl.5	cl.21	cl.9	cl.11	cl.20	cl.14	cl.6	cl.13	cl.1	cl.22	cl.17	cl.2	cl.3	cl.23	cl.4	cl.7	cl.8
activity spotting	cl.12	cl.18	cl.5	cl.16	cl.19	cl.21	cl.10	cl.14	cl.6	cl.3	cl.22	cl.9	cl.15	cl.20	cl.1	cl.2	cl.11	cl.17	cl.23	cl.13	cl.7	cl.8	cl.4

Table 4.4: Best class order (1 = best, 19 = worst) by cluster precision (Figure 4.5c), best *invariants classification* result with sweep over all parameters (Figure 6.7), *invariants classification* with known class instance borders (Figure 6.12) and *invariants classification* with activity spotting (Figure 7.8) of the car workshop experiment. The colors are connected to the activity classes (class 14 = dark red ...) in the best class order of the cluster precision.

classifier / best class order	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
cluster precision	cl.4	cl.1	cl.2	cl.13	cl.5	cl.19	cl.18	cl.16	cl.15	cl.9	cl.3	cl.14	cl.12	cl.10	cl.6	cl.11	cl.8	cl.7	cl.17
all possible results	cl.4	cl.13	cl.18	cl.1	cl.5	cl.14	cl.19	cl.6	cl.2	cl.11	cl.12	cl.3	cl.15	cl.9	cl.10	cl.17	cl.8	cl.16	cl.7
results with given inst borders	cl.4	cl.18	cl.5	cl.13	cl.1	cl.14	cl.19	cl.6	cl.12	cl.11	cl.2	cl.15	cl.3	cl.10	cl.17	cl.9	cl.16	cl.8	cl.7
activity spotting	cl.4	cl.18	cl.14	cl.5	cl.1	cl.13	cl.19	cl.11	cl.12	cl.2	cl.15	cl.6	cl.16	cl.3	cl.8	cl.10	cl.17	cl.7	cl.9

dicate low inter-class similarities. The diagonals are the similarities between the same classes, which means they are equal and thus bright. The angles of the right shoulder, front back and side body, and the angle of the right elbow look like a good candidate to distinguish the different drink vessels. When we sum up all *cluster precision* values over each channel, we see in Figure 4.4a that these channels are the first three ranked. We normalized the *cluster precision* sum to values between 0 and 1. High values of the *cluster precision* sum imply low class similarities between all classes of a channel which should make it easier to distinguish all classes on this channel. Figure 4.5a finally shows the summed up result over the classes. High values of the *cluster precision* sum imply low class similarities between the invariants of one class and all other classes which should make it easier to distinguish this specific class from the others. In general, the activities in the *drink and work* experiment are very similar. *Drinking from a cup* ranks lowest, next *drinking from a glass*, which are both the most similar drinking activities. The other two activities, *drinking from a mug* and *drinking from a bottle*, are already visually more distinctive and hence ranked higher. We cannot compare the *cluster precision* of the *drink and work* experiment to the classification results of our new algorithm, because we only used later two variations of this experiment, where all four classes are fused into one class. If there is only one class, the *cluster precision* cannot be calculated because we compare the similarity of different classes.

In the *bicycle repair* experiment, especially the angles show a good result, as we can see in Figure 4.3b, whereas the trajectories of the shoulders are the channels with the highest class similarities. This is also reflected in Figure 4.4b of the best channels. Furthermore, it seems that the movement of the shoulder is not important in this set of activities. This is logical because the movement of the arms and hands are more central in conducting movements like *turning a pedal* or *disassembling the front wheel.* In contrast, the angles, especially of the shoulder, which reflects the movement of the upper arm, seem to support the execution of the activity set. In Figure 4.5b we see the cluster precision ranking of the different classes. It is notable here that the opening and closing of screws have the worst results. This comes from the similarity of the activities which returns similar invariants. These invariants are easily mixed either by the opposed movement, opening versus closing or by the movement related to the different screws. In Table 4.3 we compare the results of the cluster precision (Figure 4.5b) with the best F1-scores of three classification results of our *invariants classifier*:

1. after a sweep over all parameters, the *invariants classifier* returns a set of all possible classification results which our classifier is capable of, shown in Figure 6.5.
2. after training the best parameter with known class instance borders, the results of our *invariants classifier* are depicted in Figure 6.11.
3. after training the best parameter and spotting the class instance borders, the results of our *invariants classifier* are depicted in Figure 7.7.

We see in Table 4.3 the order of the activity classes (numbers on the top: 1 = best) due to the *cluster precision* values and the *invariants classification* results of the classifier variations. The colors are connected to the activity classes with the order of the *cluster precision*. We can see that there is a similarity between the order of the classes which we get from the *cluster precision* and the *invariants classifier*. However, there are some divergences, e.g. class 5 and class 6 rank in the classification result quite good, although they are part of the aforementioned opening and closing of screws activities, which rank worst in the *cluster precision* plot. The reason why the *cluster precision* is not always similar to the *invariants classification* results is because in the *invariants classification* we select a subset of the invariants. If this subset is very different to invariants of other classes but not the complete set, the *cluster precision* varies obviously from the classification result.

We see in Figure 4.3c that for the *car workshop* experiment, similar to the other two experiments, the angle channels of the right/left shoulder and the right/left elbow show good *cluster precision* scores. However, also the z-trajectory of the right/left-hand tip provides good results, which can also be seen in Figure 4.4c of the best channel. Again the trajectories of the shoulder are not very significant, which makes sense, as the main movement is usually conducted by arms and hands for activities like opening the door. In Figure 4.5c we see the ranking of the different activity classes. The activities related to closing and opening doors show bad results which may relate to the similarity in the activity and therefore to the mixing up of the motion of these classes. Finally we also see in Table 4.4 the comparison of the *cluster precision* with three different invariant classification results. Again we can see the similarities in the order of best rated/classified activities, though some differences exist also. Before we already mentioned the problem, that in the *cluster precision* calculation we use all invariants while in the invariant classification we select only a subset of all invariants. We conclude that the *cluster precision* can give a first, basic knowledge about the distinctness of certain classes compared to other classes. For a definite assessment, the classifiers have to be trained.

4.6 Summary

In this chapter we showed the central step to cope with the problem of variability in human activities: we try to find invariant segments which describe the characteristics of an activity well. For this purpose, we use an agglomerative hierarchical clustering. We started the chapter by showing the different usage of clustering algorithms in the field of activity recognition. Then we explain our cluster algorithm in detail: the clusters are constructed on the segments of all training activity instances of a class and a signal channel. A specialty in our clustering approach is that we take all possible clusters during the bottom-up construction of the cluster tree. Then we sort the clusters using three features derived from the clusters. Finally, we stepwise remove the clusters with overlapping members until only clusters without any shared segment members exist. We also introduced a heuristic called *cluster precision*. Here we compare pairwise the invariant template segments of two classes using the similarity value returned from dynamic time warping. This allows us to see which data channels are good for distinguishing our classes. It also shows which classes tend to be mixed up due to their similarity and which classes can be differentiated well. We finish this chapter comparing the *cluster precision* values with classification results of our *invariants classifier*.

5

Invariant Models

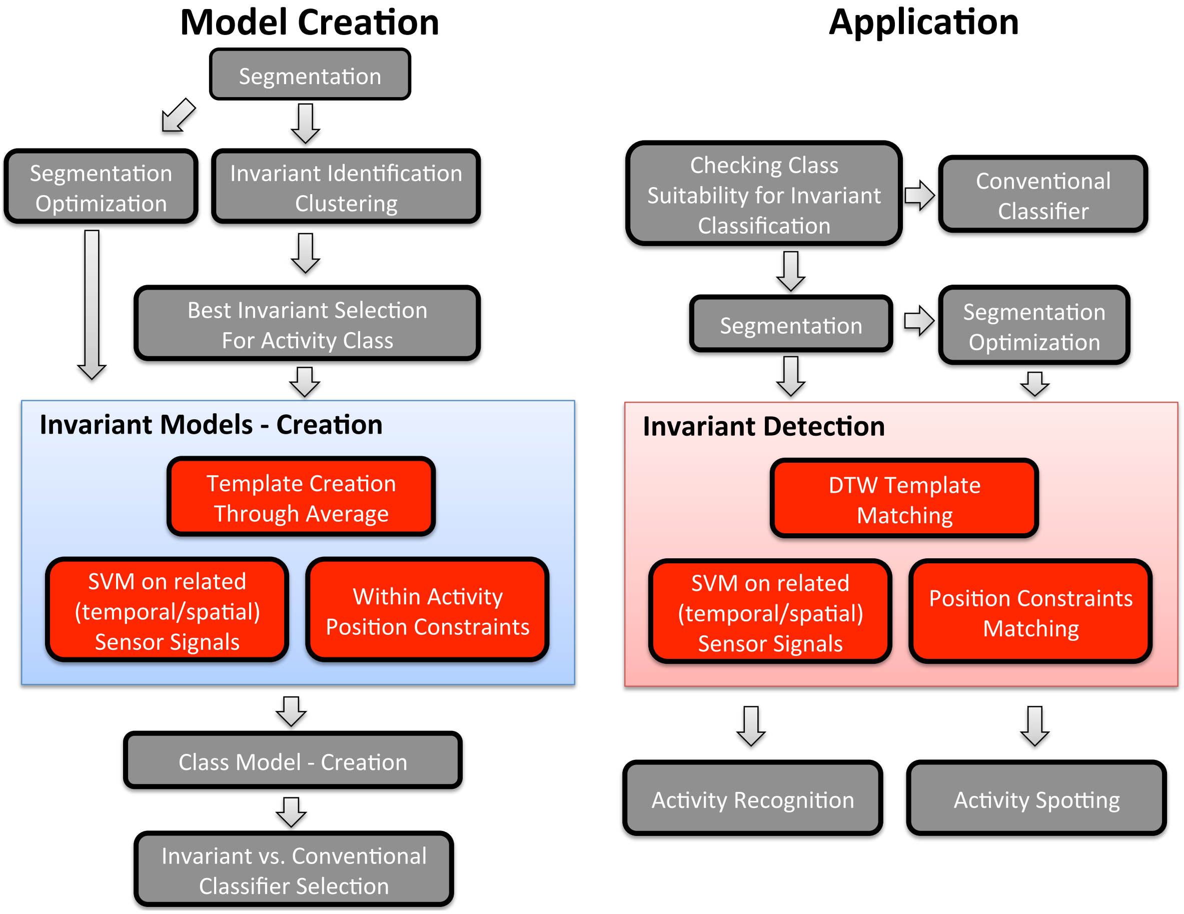

Model Creation
Application
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 5.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter.

5.1 Introduction

In the last chapter we found typical representative clusters of signal segments for each activity class, called *invariants*. Next, we want to build models from these invariants which can be applied to find segments in new sensor data which are similar to the invariant member segments. We call this process *invariant detection*. This detection of invariant segments tells us which segments are possible members of an activity and thus is one step before the final recognition of the complete activity. The most obvious invariant content to build a model are the signals of the invariant member segments themselves. Another option is to look at neighboring sensors and their signals at a time window that corresponds to the invariant. Finally, the position of the invariant member segments in the entire activity instance provides a further characteristic for determining the activity class.

This chapter takes these three ideas and builds upon three different invariant models: the first model stands for the raw signal of the invariant sub-activity which is called *template model*. The next model replenishes additional temporal/spatial information to the *template model*: we work on multi-dimensional sensor signal, and therefore it is important to add information from related sensor channels to the sensor channel of the invariant. This is achieved in the *SVM model* by training a support vector machine (SVM) with features from the related sensor channels. The third model uses the position of the invariant member segments in their activity class instance to put up certain constraints. This model is called *activity position constraints model*. All these three models are then applied to newly presented sensor signal data to detect invariant segments. The *template model* uses a template matching with a trained threshold. The *SVM model* applies the already trained SVM. The *activity position constraints model* checks the position constraints and decides whether the position variation is in the limit to accept the segment. This chapter is based on [71, 72, 73].

5.2 Related Work

The challenge of this chapter is to build a model that enables us to find certain information in a preprocessed sensor signal, identifying the activity we are looking for. In the activity recognition community, several different classification algorithms have been used. Template matching is an intuitive solution for this purpose. The similarity of an activity template signal is compared to new signals, and each signal is accepted as part of the activity if a certain threshold is met. Ko et al. [67] introduce an online context recognition method for multi-dimensional sensor data, consisting of binary, discrete or continuous signals. The signal sequences with different length are compared to the class template using dynamic time warping (DTW). Also [50] implements DTW for template matching in an online activity recognition algorithm. The templates are here selected considering the specificity of the class and the diversity to the other classes. The evaluation experiment consists here of nine different hand gestures monitored by accelerometers. A user dependent classification achieves a precision rate of 97.35% and a recall rate of 85.86%. In [95] 3D accelerometer data is discretized and then matched with activity templates using the longest common subsequence (LCSS) method. This approach is compared to template matching utilizing DTW because DTW is prone to noise in time series [127]. It is shown in the evaluation on a daily activity dataset that the accuracy of LCSS here is better than DTW. In our proposed *invariants classification* we also use DTW, but as we only work on specific, short sub-activity signals, the problem with data noise is not severe. When the DTW similarity measure would be used on complete activity instances the danger of outliers is much higher due to the variation in the activity execution.

Another classification method is the hidden Markov model (HMM). It represents a statistical model consisting of a Markov chain with invisible states and transition probabilities, and visible outputs with a probability distribution, depending on the invisible states. In [3] HMMs are used to classify arm movements related to food and fluid intake. The inertial sensor signal data is here first segmented, and then several features of these segments are checked against a threshold and finally given to the HMM to decide the class membership. Detection accuracy of 87% is achieved. Lukowicz et al. [83] train HMMs to recognize different hand gestures from assembly tasks in a wood workshop. Sensor data from two accelerometers therefor return the information about human activities. An accuracy of 84.4% is reached in the activity classification. Zhu et al. [141] also apply HMMs for hand gesture recognition on IMU sensor data. Here even hierarchical hidden Markov models are used to model certain gesture constraints and improve the classification result of the single gesture HMMs. Related to HMMs are conditional random fields (CRF), another statistical model taking the temporal sequence of activities into account. Vail et al. [124] compare the performance of CRFs and HMMs activity recognition in a multi-agent scenario, showing that CRFs perform better to equal than HMMs. In [48] they use a skip chain CRF to model interleaving and concurrent activities.

A Support Vector Machines tries to construct a possible large margin without objects between the objects of the different classes. In [21] an electrooculography system is used to detect three different eye movements (saccades, fixations, and blinks), using an SVM. In the person independent classification, a precision of 76.1 and recall of 70.5 is reached. Anguita et al. [5] implement a multi-class SVM applied on inertial sensors of a smartphone. To save computational power, memory, and energy the SVM algorithm is adapted to calculate with fixed-point arithmetic. He et al. [51] apply an SVM to recognize four daily activities (running, still, jumping and walking) with one 3D accelerometer put in the trouser pocket of the subject. A discrete cosine transformation is conducted on the raw data, followed by a Principal Component Analysis (PCA) and a multi-class SVM. The classification result achieves 97.51% accuracy.

A decision tree constructs a tree structure where every branch represents a decision rule. Bao et al. [9] apply a C4.5 decision tree to recognize human activities with the help of acceleration sensors. The experiment consists of 20 everyday activities from a set of 20 subjects. The classification results show an overall accuracy rate of 84%. In [102] a binary tree recognizes six activities (lying, sitting/standing, walking, running, and cycling) analyzing 3D accelerometers. 86.6% accuracy was reached and after a classifier personalization even 94.0%.

The k-nearest neighborhood algorithm (k-NN) classifies by using a majority vote of the k nearest labeled objects to the test object. In [74] Tai Chi movements are analyzed with eight accelerometer and gyroscope sensors. A k-NN algorithm classifies 85% of two different Thai movements correctly (user dependent). It also detects in 76% of all cases the subjects out of three correctly. In [101] a kNN and a naive Bayes classifier are used to detect 20 activity classes with the help of seven motion sensors, 16 force sensing resistors (arm muscle monitoring) and a high accuracy indoor location system. The naive Bayes algorithm applies the Bayes' law for the classification.

5.3 Invariant Models Creation

The first step in the detection of segments which are similar to the member segments of an invariant is the creation of an invariant model. Starting from the clustering result, we generate three different models from the invariants. The first model is a template segment, representing all invariant member segments. The second model is an SVM based classification model with features from other sensor channels on a time window corresponding to the invariant's template segment. The third model is a position constraint of the invariant member segments in their class activity instances. These three models allow us to detect later segments belonging to an activity class which we call *invariant detection*.

5.3.1 Template Creation – Average

This is the base model, while the other two following models are optional in the later *invariant detection* process. Here we create a template segment which is derived from the clustering step and which forms an average segment of all invariant members. In order to achieve this, we take the cluster center of the invariant as template segment. The cluster center is the medoid of the invariant, i.e it is determined by choosing the cluster member segment with the highest average similarity to all other member segments. In the clustering we used the DTW similarity, hence we use the DTW similarity in the cluster center determination too.

5.3.2 SVM – Sensor Signal Features

In this model, we want to add extra information by adding supplemental sensor channels to the channel of the invariant. Many wearable sensor setups for human activity recognition use several sensors. Each invariant is related to exactly one sensor channel. This channel already provides relevant information related to the activity class which we want to identify: the invariant *template model*. However, there is possibly more information hidden in one of the other sensor signals in the same time frame. Therefore we calculate features on selected sensor channels with the time frame of the invariant template, then apply a feature selection and train a classification model, in our case a support vector machine.

The experiments we analyzed in our algorithm evaluation consist of a set of sensors providing multi-dimensional data signals. The data signals are trajectory and angle signals of upper body limbs. The selection of the supplemental sensor channels applies the following strategy. If the invariant's channel is a trajectory axis, we use the data channels of the two other trajectory axes and all angles of the same body side (right/left) as an additional information source. If the invariant's channel is an angle, we use the data channels of all other angles of the same body side (right/left). We train an SVM with features calculated on segments of the additional sensor channels. These segments have the segmentation borders of the invariant's sensor channel, on the data points of the supplemental channels. The used feature set consists of several signal oriented features and features derived from the polynomial approximation of the signal (see [43, 39]):

1. start point trajectory-axis/angle
2. end point trajectory-axis/angle
3. variance trajectory-axis/angle

4. mean trajectory-axis/angle
5. variance central difference quotient of trajectory-axis/angle
6. max central difference quotient of trajectory-axis/angle
7. min central difference quotient of trajectory-axis/angle
8. mean central difference quotient of trajectory-axis/angle
9. segment length
10. polynomial approximation of trajectory-axis/angle signal: coefficient 1 – equivalent to the slope of the signal
11. polynomial approximation of trajectory-axis/angle signal: coefficient 2 – equivalent to the curve of the signal
12. polynomial approximation of trajectory-axis/angle signal: coefficient 3 – equivalent to the change of curve of the signal
13. polynomial approximation of trajectory-axis/angle signal: coefficient 4 – equivalent to the change of the change of curve of the signal

Next, all features (feat) are normalized by subtracting their mean and dividing by the standard deviation (SD) of the feature values of all train segments of this class, which is known as standard score:

$$feat_{norm} = \frac{feat - mean_{feat}}{SD_{feat}} \tag{5.1}$$

We end up with a large set of features per segment: the number of channels times the number of features. To reduce the size of the set of features, we implement the feature selection measure called information gain [31, 135] and select the 18 best features from the chosen segments. In the process of the information gain calculation, we have to discretize the numeric values of our features [56]. Finally, we train an SVM which implements a radial basis function kernel (RBF kernel) with the selected features. The selection of the training set is explained in detail in Section 5.4.2, as we construct it from results of the template matching of Section 5.4.1.

5.3.3 Activity Position Constraints model

Here we build a model representing the position of the invariant in relation to the entire activity. The cluster center is used, described in the *template model*, as a typical representative segment of one activity class. We want to get a value describing the position of the template segment in its class instance. For this purpose, we determine the start and the end position in percentage of all invariant member segments in their belonging activity class instance. Then we calculate the mean and the standard deviation of the start and the end position in percentage of all invariant member segments. The mean of the start values and the mean of the end values show us the position. The standard deviation of the start values and the standard deviation of the end values tell us if the position is stable, i.e. if all invariant member segments are found more or less at the same position or not. We construct two different activity position models, the *simple segment position model* and the *estimated segment position model*. In the *simple segment position model* we calculate the position in percent of the invariant segment's start and end point in relation to the corresponding class instance. In the *estimated segment position* method we estimate the start and the end point of the class instance in relation to the invariant segment.

5.4 Invariant Detection

In the *invariant detection* step, we want to detect segments with high similarity to the segment members of an invariant. Therefore we apply the constructed invariant models to new segmented sensor signals. For each activity class, we build the three invariant models on a set of found invariants. These models show us then for a given segment if it is a member of the activity class or not.

The *invariant detection* step returns only segments of a certain activity class. This means that we need an additional step which combines the detected segments and returns the complete found activity class instance. In the next two chapters, we explain how we determine the entire activity class instances. But first, we show the segment detection of the three invariant models in detail.

5.4.1 Template Matching

Our first *invariant detection* strategy tries to determine if a test segment is part of an activity class, by comparing the test segment with a typical representative class segment, which is the template segment of the *template model.* If the similarity between these two segments is higher than a threshold value, then the test segment is accepted as a member of the class. In our algorithm, the similarity value is the result of the dynamic time warping algorithm (DTW) applied to two signals. The smaller the DTW value, the higher the similarity, e.g. a DTW value of zero signifies that both values have equal data point values (the length of the signals can still differ). The *template model* is, as described before, the cluster center we got from the clustering beforehand. The detection threshold is the value so that x percent of the cluster member segments have a less or equal similarity to the cluster center. As an invariant is a group of representative segments of a certain activity class, we need a high similarity to these segments. So we chose the value of x to be 95.

5.4.2 SVM

We refine our result from the *template matching* by adding another detection step: the application of the *SVM model.* In the creation of the *SVM model* we use only segments, which were positively classified by the *template matching.* This means that we first apply the *template matching* to the set of segments which we use for the construction of the *SVM model.* For every segment, we know the labeling from the data recording which implies that we have a labeling and thus know the correct activity class. So we can separate the segments into true positives (segments which were detected correctly as an appropriate invariant member) and false positives (segments which were detected falsely as an appropriate invariant member). This is necessary because we need positive and negative patterns for the training of the SVM classifier. To get a balanced training set, we adapt the number of true positive and false positive segment features. This is achieved by randomly selecting the members from the bigger set so that there is a maximum difference of the factor of 1.5 between the two sets. Now having a set of segments for the training of the SVM we calculate several features, as described in the *SVM model* section, and train the SVM with a subset of these features. The features are calculated on the related sensor channel signals of the invariant used in the *template model.* The segment borders for the feature calculation are the segment borders of the found segment of the *template matching* on the signals of the related sensors.

In the application of the *SVM model* we detect segments of the new signal from the same sensor channel as the invariant. Here we use segments only as input which were detected as invariant segments by the *template matching.* First, we calculate the same features on the same related sensor signals like in the *SVM model* creation. The segment borders for the feature calculation are the segment borders of the signal channel of the invariant, which was chosen in the *template matching.* We use the feature set as input for the *SVM model* which returns a detection result for the examined segment.

5.4.3 Position Constraints Matching

The *position constraints matching* is tested on all segments, and also on the found segments from the *template matching* and the *SVM.* Here we check the conformity of the segment position in the whole activity instance. In the *activity position constraints model* we get the position of the invariant template in its class instance represented by four values: the mean and the standard deviation of the start and end position of all cluster members. The *activity position constraints model* is compared to the instance positions of the segment which is checked for invariant similarity. This means that we always need the class instance borders for the *position constraints matching.* Here we use the class instance borders which are given from the experiment recordings. Next, we show the two different approaches we applied for the position comparison, the *simple segment position model* and the *estimated segment position model.*

Simple segment position model

In the *simple segment position model* we compare the *activity position constraints model* with the position of the segment which we want to classify. We accept this segment as found if the difference between the segment start/end position in its activity instance and the start/end position of the *activity position constraints model* is smaller than the standard deviation of the *activity position constraints model*, multiplied by a factor $f =$ 1, 2 or 3. The segment belongs to the activity instance which covers most of the segment. Two parameters have to be trained: The first parameter is the factor f. The second parameter is whether start or end or both start and end are tested.

Estimated segment position model

In the *estimated segment position model* we estimate the activity instance start position and the activity instance end position of the segment which we want to check. We accept the segment as found if the difference between the activity instance start/end related to the invariant template and the estimated start/end of the instance is smaller than a specific threshold. This threshold is derived from the standard deviation of the template's start/end point of the *position constrain model.*

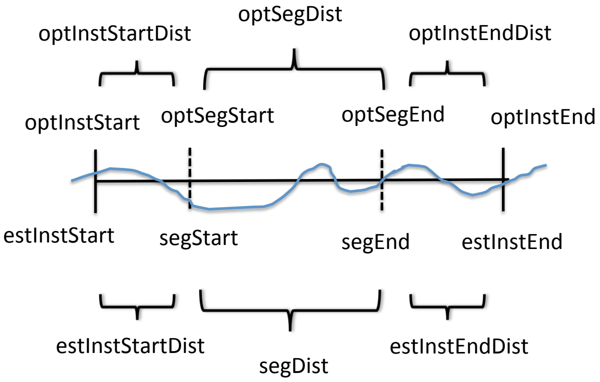


Figure 5.2: Estimated position calculation

To estimate the instance position we start with the given start/end point of the segment to check (*segStart/segEnd*) in time stamps and the optimal start/end point (*optSegStart/optSegEnd*) derived from the *activity position constraints model* in percent. Then we calculate the length of the segment to check (*segDist*) and the optimal segment (*optSegDist*). The optimal segment is here of course the invariant template.

$$segDist = segEnd - segStart \tag{5.2}$$

$$optSegDist = optSegEnd - optSegStart \tag{5.3}$$

The distance between the start point of the optimal instance (*optInstStart*) and the optimal segment (*optSegStart*) is called optimal instance start distance (*optInstStartDist*). As *optInstStart* has the value 0, the resulting distance is equal to *optSegStart*.

$$optInstStartDist = optSegStart \tag{5.4}$$

Next, we estimate the value of the estimated instance start distance (estInstStartDist), which is the distance between the start point of the segment to check (*segStart*) and the start point of the estimated instance (*estInstStart*). This can be calculated by using the relation:

$$estInstStartDist = \frac{optInstStartDist}{optSegDist} * segDist \tag{5.5}$$

Having determined this distance it is possible to determine the estimated start point of the instance (*estInstStart*) by subtracting the estimated distance between instance start and segment start (*estInstStartDist*) from the segment start.

$$estInstStart = segStart - estInstStartDist \tag{5.6}$$

The estimation of the end point of the instance is similar to the estimation of the instance's start point. The only two differences are: First, the distance between the end points of the optimal instance (*optInstEnd*) and the optimal segment (*optSegEnd*) is calculated by subtracting the optimal segment end point from one, as the optimal segment points are percentaged values. Second the end point of the estimated instance (*estInstEnd*) is the sum of the end point of the segment to check (*segEnd*) and the estimated distance between segment to check end point and estimated instance end point (*estInstEndDist*).

$$optInstEndDist = 1 - optSegEnd; \tag{5.7}$$

$$estInstEndDist = \frac{optInstEndDist}{optSegDist} * segDist \tag{5.8}$$

$$estInstEnd = segEnd + estInstEndDist \tag{5.9}$$

To accept the segment as member of the invariant and thus as member of the activity class, the following condition has to be met: the difference between the estimated instance start/end and the nearest instance start/end point of the invariant's activity class has to be smaller than the standard deviation of the position in the cluster multiplied with a factor $f = 1$, 2 or 3. The formula to calculate the standard deviation from percentage to time stamps follows:

$$SD_{Start} = \frac{SD_{optStart}}{optSegDist} * segDist \tag{5.10}$$

$$SD_{End} = \frac{SD_{optEnd}}{optSegDist} * segDist \tag{5.11}$$

Two parameters have to be trained: The first parameter is the the factor *f*. The second parameter is whether start or end or both start and end are tested.

5.5 Invariant Detection with Segmentation Optimization

In the *invariant detection* step we also vary the segmentation strategy by fusing segments with neighboring segments. We vary the segmentation strategy to find better "closed" motions. This is accomplished by using information from the clustering: When the members of a cluster cover nearly the same amount of class instances after the segment fusion, then the fusion is reasonable. The details are described in Section 3.3.3.

We test three different *invariant detection* strategies for this segment optimization. In the first two strategies, we check if a segment fusion is reasonable on found segments, after applying the *invariant detection* with the *template model* or the *SVM model*. If so, we apply the *invariant detection* using the *template model* to the before found segments, but now we extend segments before the *invariant detection*. We call both strategies *invariant detection with simple segment optimization*, or each on its own *template matching simple segment optimization* and *SVM simple segment optimization*. The third strategy trains the *SVM model* with fused segments. Then we apply this model to fused segments in the *invariant detection*. We call this strategy *SVM segment optimization*.

5.6 Results

We test our *invariant detection* on three experiments (*drink and work*, *bicycle repair* and *car workshop*) which we introduced in the Chapter 2. The *drink and work* experiment is evaluated in three different configurations varying its activity classes. The three experiments consist of one, four, 19 or 21 activity classes conducted by six or eight subjects. For the evaluation we apply a leave-one-out cross-validation: we create the invariant models with the data of *n-1* subjects and apply these models to one subject, where *n* stands for the total number of subjects in the experiment. The subjects to which we apply the models are swapped until all are covered. In our evaluation, we selected the 60 best invariants from the *invariant identification*. The found segments for each *invariant detection* strategy are divided into true positives when the majority of the found segment is part of a correct activity class instance, and false positives otherwise. The correct activity class instances are known from the labeling of the experiment data. We sum up the number of detected true and false positive segments for all classes, invariants and *invariant detection*

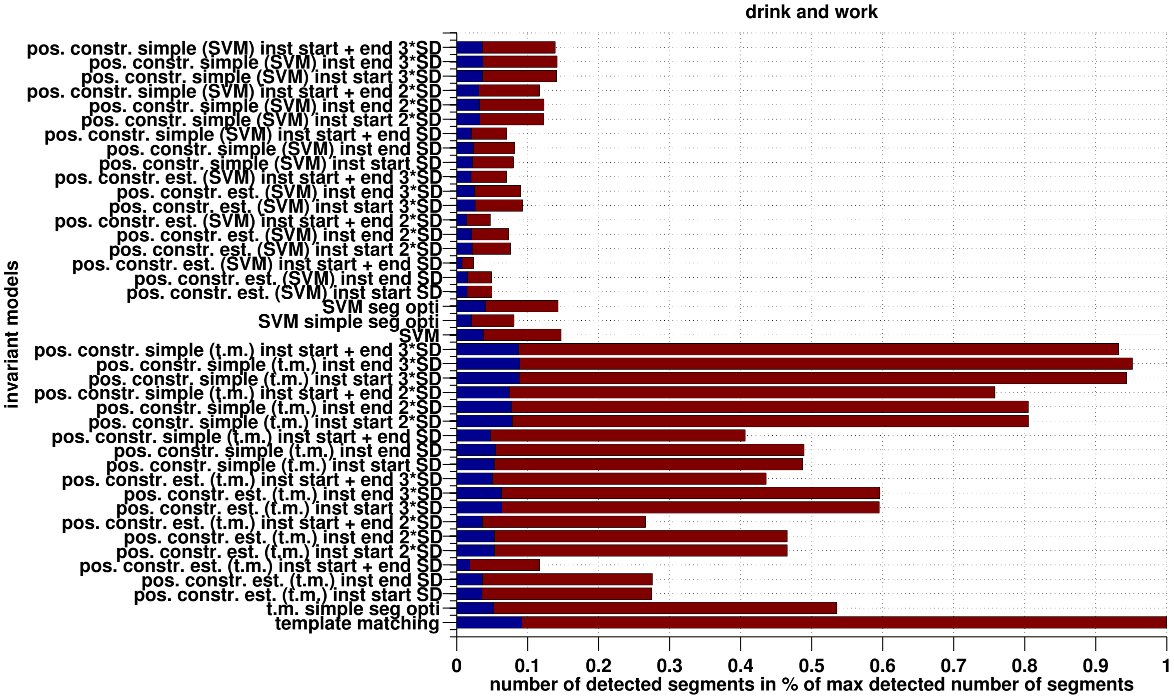


Figure 5.3: Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work experiment.

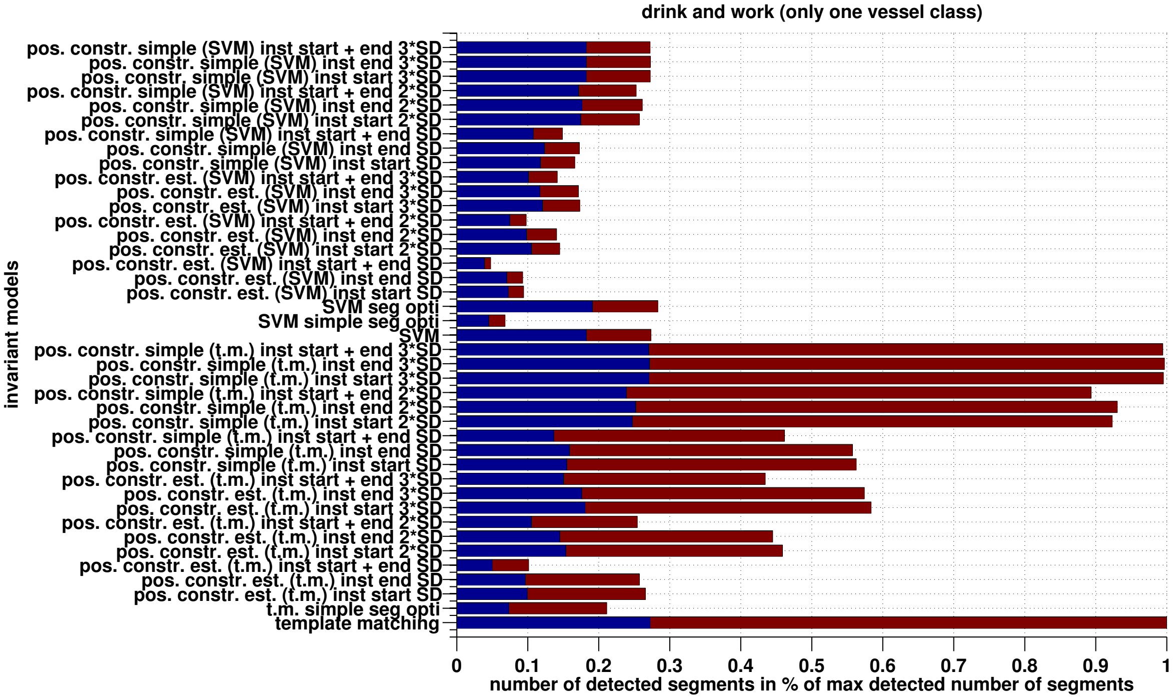


Figure 5.4: Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work (only one vessel class) experiment.

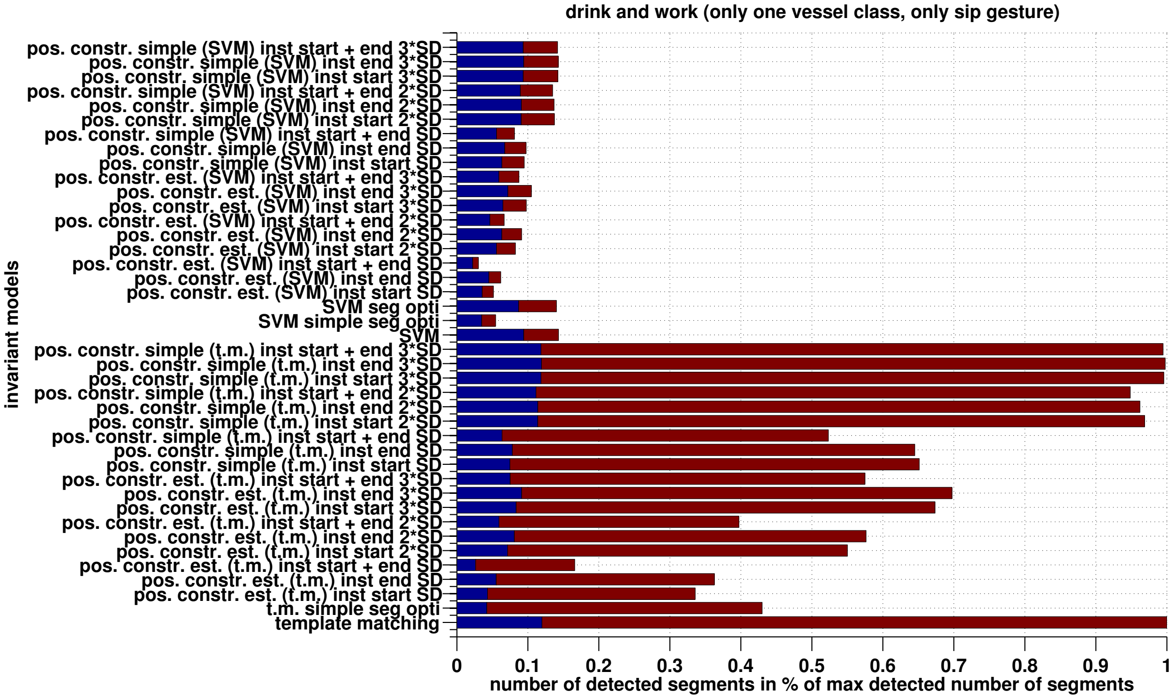


Figure 5.5: Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work (only one vessel class, only sip gesture) experiment.

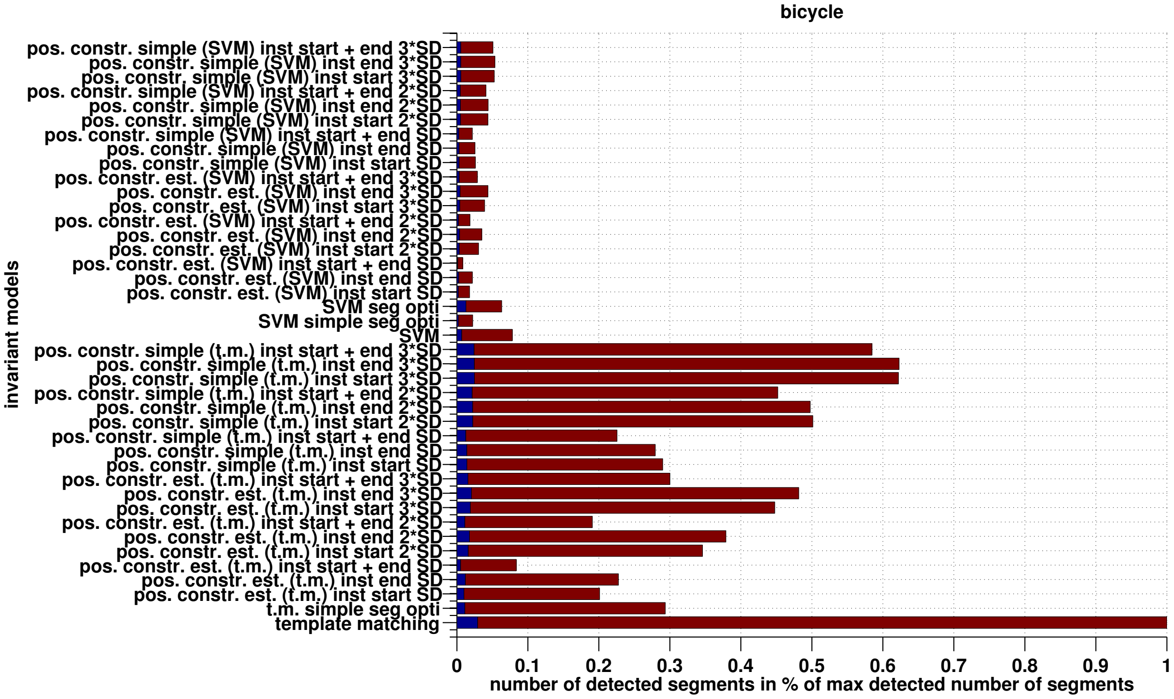


Figure 5.6: Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the bicycle experiment.

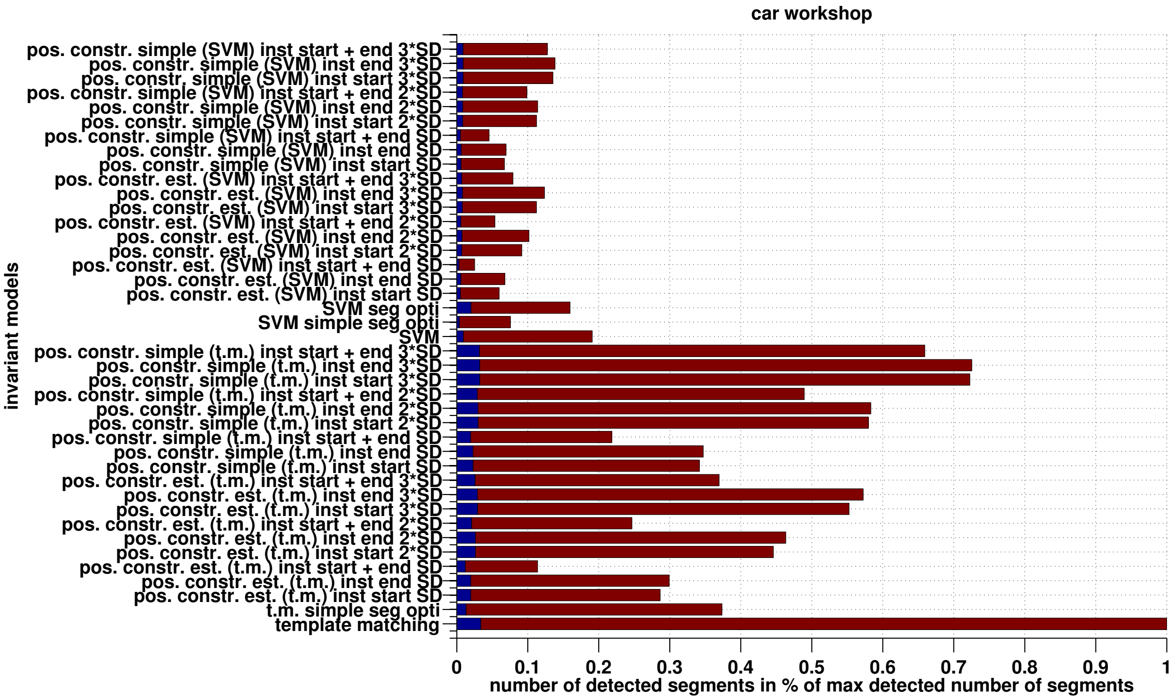


Figure 5.7: Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the car workshop experiment.

strategies from the cross-validation. Figures 5.3, 5.4, 5.5, 5.6 and 5.7 depict the segment sums of all experiments, marking the true positive segments (tps) blue and the false positive segments (fps) red. The sum of true and false positives is normalized within the range of 0 to 1 for the *invariant detection* strategies of one experiment, because relative numbers are not relevant. Starting with the *drink and work* experiment with four activity classes in Figure 5.3, we see that the basis of all *invariant detection* strategies is the *template matching*: it finds the most segments of all *invariant detection* strategies because all others build upon its results. The *template matching simple segment optimization* lowers not only the number of fps but also the number of tps compared to the *template matching*. Combining the *template matching* with the *position constraints* shows different reductions of tps and fps dependent on the factor SD which is the standard deviation of the positions. The *simple segment position model* provides a smaller reduction in both tps and fps than the *estimated segment position model*. Taking the start and the end position into account lowers the number of found segments more than checking only for the start or end position alone, which is logical, as the condition of start and end position is more strict. The *SVM* features a notable decrease of fps. This result is the basis for the remaining models (except *SVM segment optimization*). We see similar characteristic in the *SVM simple segment optimization* and *position constraints matching* like in the *template matching* case, only with the different starting basis result. The *SVM segment optimization* is different because we train a new model with optimized segments. It shows a good relation between tps and fps. The other four experiments exhibit similar relations between the *invariant detection* result. It is notable that there are much less found tps in the *bicycle repair* and *car workshop* than in the three variations of the *drink and work* experiment. This could be related to the higher challenge due to more activity classes. We show here the sum of all 60 best invariants. In the later fusion of the detected invariants, we train the best set of these invariants, meaning that not all of them are always used.

5.7 Summary

In this chapter, we introduced the *invariant detection*, which consists of the construction and the application of three different models using the proposed invariants. The aim is to detect segments on new sensor data which are similar to the member segments of the invariants which we identified before. In this way, we get segments which can be related to a certain activity class.

We start presenting related work about different classification algorithms. Next we show the construction of three invariant models: *template model*, *SVM model* and *activity position constraints model*. The *template model* constructs a template segment of the invariant which is the center of the invariant. The *SVM model* builds an SVM with a feature set from other signal channels. The key point here is that the segmentation of the invariant's signal channel is used on these other signal channels. The *activity position constraints model* determines the constraints which are related to the invariant's position in its activity instances. This is calculated by taking the mean of the positions of the invariant's cluster members. The standard deviation of these cluster member positions defines the position variation threshold. These three models are applied to segments of new sensor data signals in the *invariant detection* step. The *template model* implements a template matching with DTW similarity measure. The trained SVM is then applied only on the result of the *template matching*. Finally, the *position constraints matching* checks the results of the template matching and the SVM, and also on the raw data, if the position constraints are fulfilled. Furthermore, we show the *invariant detection with the segment optimization* which we have introduced in the segmentation chapter. Here we fuse adjacent segments and apply either the template matching or the SVM classifier on these new segments. Finally, we plot the number of correct and incorrect found segments by applying all different detection strategies to our three evaluation experiments.

6

Class Instance Model

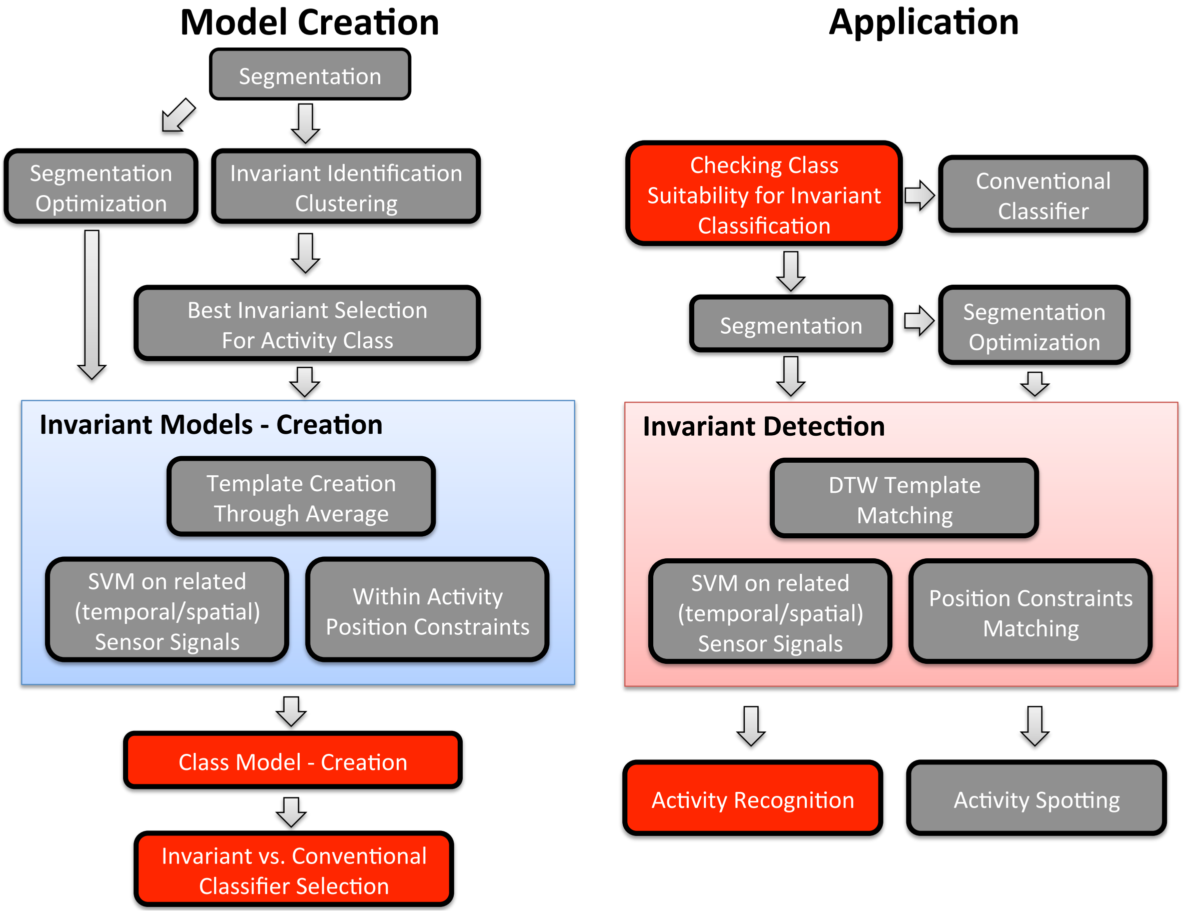

Model Creation
Application
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 6.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter.

6.1 Introduction

In the previous chapter, we built three different invariant models for each activity class invariant. Applied to sensor data segments, these invariant models decide if a segment is a member of the related activity class or not. In this chapter, we want to develop a strategy to determine the activity instances using the detected segments for a final activity classification result. Every complete activity class instance consists of several segments on each sensor channel. When we combine the single invariant models to one activity classification model, we get a final classification result of complete activities. For this purpose, we construct the *class instance model* which fuses the invariant models into one. First, we have to sort the set of invariants depending on the classification quality and then train the optimal number of used invariants and the optimal number of needed invariants. Here used invariants refer to the invariants which are fused in the *class instance model*. Needed invariants mean the number of invariants which have to be detected inside an activity instance so that it counts as found. Several other parameters have to be trained for the *class instance model*, like the invariant model type to use. In the evaluation of our *invariants classifier* we compare the classification results with conventional classifiers from publications which worked on the same experiments like we. Thus we try in the last step to combine our *invariants classifier* with the conventional classifier so that each classification modality, invariants or conventional, is used only when it is more beneficial.

In this chapter, we simplify the problem of the activity class instance detection by taking the given instance borders of the evaluation experiments and by creating artificial instances for the zero class. In Chapter 7 we address this problem again, showing a solution how to estimate activity instance borders. This chapter is based on [71, 72, 73].

6.2 Evaluation

We evaluate the *invariants classification* results on three experiments: the *drink and work* scenario, the *bicycle repair* scenario and the *car workshop* scenario. In each experiment, we recorded between six and eight subjects with several repetitions of the activities. Depending on the experiment, there exist 1, 4, 19 or 23 activity classes. We classify each activity class separately.

For the evaluation, we construct the invariant models on training data from the experiments and then apply the invariant models on test data again obtained from the experiment recordings. We use a leave-one-out cross-validation where the complete data of one subject is taken as test data entity. This means that we train our classifiers on *n-1* subjects, where *n* stands for the total number of subjects per experiment, and then apply them on the remaining subject, which represents the test dataset. The test subject is changed until all subjects are covered. In the construction of the invariant models, we have to train several parameters. Here we also have to apply a leave-one-out cross-validation, but only on the training data of the first cross-validation. Figure 6.2 shows the two connected cross-validations used for the parameter training and the final classification results. The blue circles are the test and the training set for the cross-validation of the parameter training and at the same time the training set of the final classification. The green circles represent the test dataset for the final classification. The details, how we get the parameters from *n-1* parameter train results for each test dataset, is explained in detail in the next section.

We measure the classification quality using precision, recall and F1-score. These three

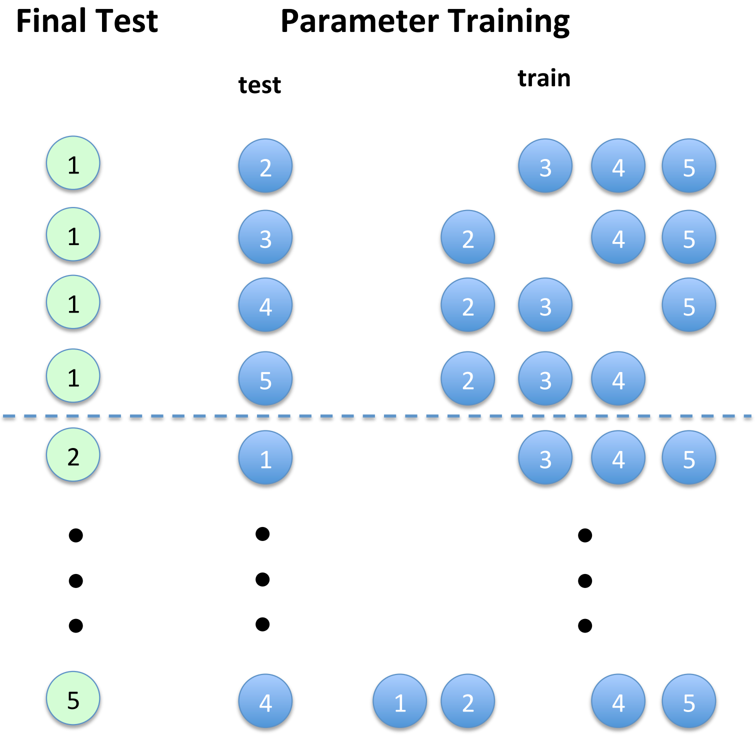


Figure 6.2: Two connected cross-validations for the parameter training (blue) and the final classification result (green: test data, blue: train data). Each circle represents the data of one subject.

values are defined as follows:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{6.1}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{6.2}$$

$$F1score = 2 * \frac{precision * recall}{precision + recall} \tag{6.3}$$

True positives here refer to the number of all class instances, which were classified correctly, while the false positives value is the number of found class instances which are not members of the class activity. False negatives are the number of all class activities instances which were not found by the classifier. The F1-score is the harmonic mean of precision and recall.

The problem which we still have to solve is that we need a list of positive classified activity instances, but our classifier returns only classified segments. An activity instance consists of several segments. However, having found a class segment we do not know the instance borders. If we want to calculate precision, recall, and F1-score, then we need the instances because we are only interested in the classification of complete activities and not parts. To simplify this issue, we take the class instance borders from the recording as known, also in the final classification on test data. Only for the zero class, we calculate artificial instances by taking the mean length of all other class instances and then separating the data stream of the zero class accordingly. When we apply a single invariant model on new segmented data, we classify an activity instance as a correct class member, when at least one class segment is found inside the borders of an instance. In the *class instance model*, we train a certain parameter (needed invariants) which defines the number of invariant models, where segments have to be found so that the instance is counted as a class member. In the next chapter, we again address the problem of class instance borders and show a solution which is independent of previously known activity labels.

To evaluate the *invariants classification* results, we compare them with already published results by using conventional classification strategies. These published results are based on the same three experiments which we took for testing our algorithm. Next, we introduce the conventional classifiers and the related articles:

The *drink and work* experiment is checked against [2]. Here, on the accelerometer and gyroscope data, the following time-domain features were calculated: sum of absolute amplitude, difference of begin and end of amplitude, number of zero-crossings, amplitude begin and end, sum of amplitude differences, mean and variance. These features were computed not only on the whole motion instance but also on three equally distributed sections of the sensor data, resulting in a set of 200 features. Next, a feature selection was conducted using the Mann-Whitney-Wilcoxon test, resulting in a group of 20 best features. These selected features have the best rank in the Man-Whitney-Wilcoxon test and a minimum correlation to already selected features. This matches the method shown in [139]. For the classification, a Feature Similarity Search (FSS) algorithm [4] is trained, looking only for the drink gesture without distinguishing the four different drink vessels. For the data, an equidistant segmentation of 0.5 Hz was used. A distance threshold and a scaling window were trained. Then for all test segmentation points, a scaling window of formerly received data was considered. The features calculated in the window were compared to the trained model by determining the Euclidean distance. This distance was used in comparison to the threshold to decide whether a drinking activity was found or not. The user-specific classification results show a parameter sweep varying the distance threshold.

The *bicycle repair* experiment is compared to [99]. Here two sensor systems were used for the activity recognition task: IMUs and two ultrasonic sensor devices (Hexamite HX900SIO) fixed on both wrists. Four base stations for the ultrasonic sensors are deployed in the experiment room. The IMUs measure the motions of the subject and the ultrasonic sensors measure the distance from sensors to the four base stations. The fusion of both sensors employs a Kalman filter and results in trajectories of the left and the right hand. Additionally, the position information is mapped to several locations where the different activities happen. This is the basis of the *location-based spotting* which uses location information to identify possible segments for all activity classes which match the spatial position of the hands. The *location trajectory-based spotting using polynomial matching cost features* step only considers segments found by the former *location-based spotting*. Here the hand trajectories are converted to discrete symbol strings by using an algorithm developed in [117, 118, 119]. For each class, representative strings are identified with an evolutionary search. These representatives string detect similar patterns with a string matching approach. An LDA classifier with polynomial features, calculated on the matching cost data, improves the spotting result. In the evaluation of our *invariants classification* algorithm with the *bicycle repair* experiment we only use IMU sensor data. Hence we compare the results of our algorithm with the *location trajectory-based spotting using polynomial matching cost features* because this method uses the least information from the ultrasonic sensors.

The *car workshop* experiment is compared with results from [142]. In this publication, a body model is used like in our algorithm. The data stream is segmented by finding local minima of the variance of each hand's movement. The segments are accepted if the pairwise combination of the minima complies with a minimal and maximal length. On these segments several features are calculated: motion primitives which describe up and down movements of the arms, push and pull movements, back and forward bending of the body, arm-twisting, a histogram on the movement direction of both hands, posture features which describe the arms' orientation towards gravity, the distance between the two hands, the hands' height, the torso's relative direction to the car, and finally location features which result from an ultra-wideband position system called Ubisense. The classification is then done with joint boosting [122].

6.3 Class Instance Model Creation

In the former chapter, we described the construction and the application of invariant models to detect a segment. We also saw in Chapter 4 that we find several invariants on different sensor channels for each activity class. To get a final activity instance classification result, we have to fuse the invariant models into a *class instance model*. This *class instance model* consists of a number of equal invariant models and several parameters. The construction of the *class instance model* is achieved by creating a certain number of invariant models and training the following parameters.

1. best invariant model configuration:
   This parameter defines the used invariant models for the creation of the *class instance model*. We use the *template model*, the *SVM model* and the *activity position constraints model*.

2. basic or optimized segmentation strategy:
   This parameter defines if the normal segmentation or the *segment optimization* is used.

3. sort strategy for the invariant models:
   This parameter defines how the invariants are sorted before the fusion of invariant models. In this way we only use the best x invariant models, where x is the parameter *number of used invariants*.

4. number of used invariants:
   This parameter defines the number of invariants which are fused in the creation of the *class instance model* for the final classification result. We use between 1 and 60 invariants per activity class.

5. number of needed invariants:
   This parameter defines the number of invariants which have to provide positive classification results inside the borders of an activity class instance so that the instance is counted as a member of the activity class. We test values between 1 and 60.

6. tolerance threshold for the activity position model:
   This parameter defines the factor for the multiplication with the standard deviation of the start/end position of the invariant segment in its class instance. This is needed to get a threshold for the maximum difference in the position in the *position constraints matching*. We test the values 1, 2 and 3.

7. start - end - start and end:
   This parameter defines if only the start point or only the end point or both start and end points have to match in the *position constraints matching.* We test all three options.

Before we start to explain our strategy to train the aforementioned parameter set, we want to show several classification results by varying these parameters. This shows the general classification capability of our *class instance model.*

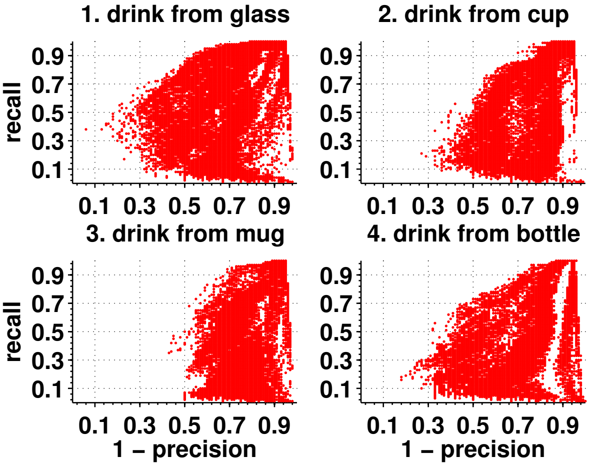
6.3.1 Possible Results

We present here two different plots of classification results for each of the three evaluation experiments. The first plot shows all *invariants classification* results with parameter variation applying the *class instance model* versus the best result of the conventional classifiers. The results of the conventional classifiers which we use in comparison to the results of our *invariants classification* algorithm, are taken from three publications: [2, 99, 142]. The second plot depicts the pareto curve of ten *class instance models* each using a different invariant detection strategy (plot color in brackets) condensing the results of the first plot:

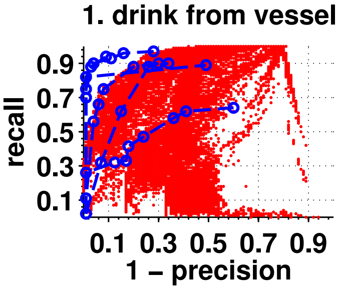
1. template matching (bright green)
2. SVM (cyan)
3. position constraints matching simple on template matching results (blue)
4. position constraints matching estimated on template matching results (black)
5. position constraints matching simple on SVM results (turquoise)
6. position constraints matching estimated on SVM results (pink)
7. position constraints matching simple on all data (brown)
8. position constraints matching estimated on all data (orange)
9. template matching/SVM with simple segment optimization (yellow)
10. SVM with segment optimization (bright red)

Plots of all classification result show the recall versus 1 - precision outcomes. The pareto curve is the line which shows only the best results for each recall versus 1 - precision step from the first plot of all *invariants classification* results.

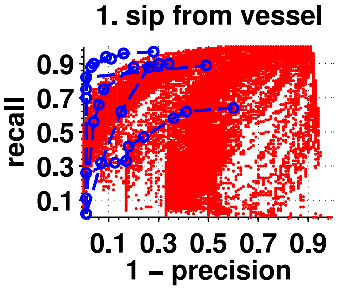
In Figure 6.3a we see all possible classification outcomes of the *drink and work* experiment, dependent on the used invariant detection strategies and the parameters. Next, we discuss the best classification outcomes of the *drink and work* experiment by using the different invariant detection strategy, visible in the pareto plot of Figure 6.4a. This experiment consists only of four classes: drinking from a cup, a glass, a mug, and a bottle. The *template matching* (bright green) returns a good recall but a bad precision result. Drinking from a glass is the best recognized class. The *position constraints matching simple on template matching results* (blue) shows quite similar results in all four classes. However, with the *position constraints matching estimated on template matching results* (black), the classification outcome of the four drinking activities has better precision. Contrary to the *template matching* and also to the *position constraints matching estimated on template matching results*, the classification with the *SVM model* (cyan) has a



(a) Drink and work.

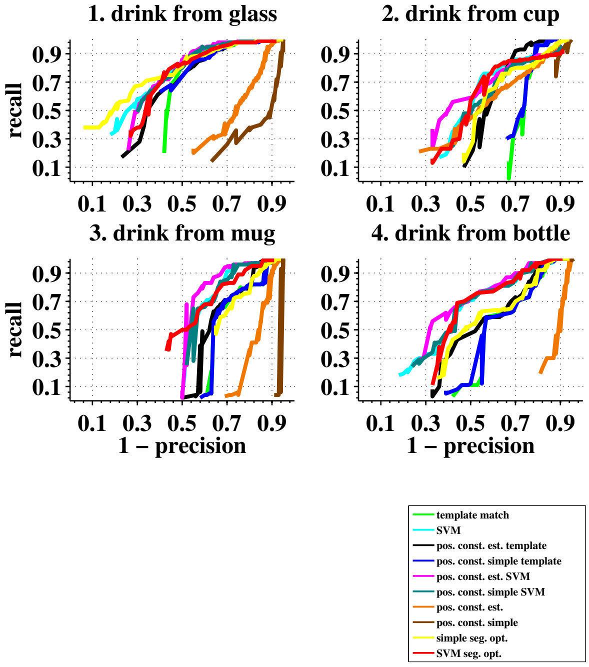


(b) Drink and work, one vessel.

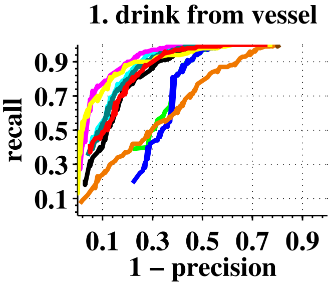


(c) Drink and work, one vessel, sip.

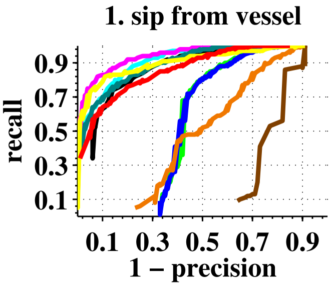
Figure 6.3: All classification results from our algorithm: red dots. Results from [2]: blue dots.



(a) Drink and work.



(b) Drink and work, one vessel.



(c) Drink and work, one vessel, sip.

Figure 6.4: Classification results with all invariant detection strategies.

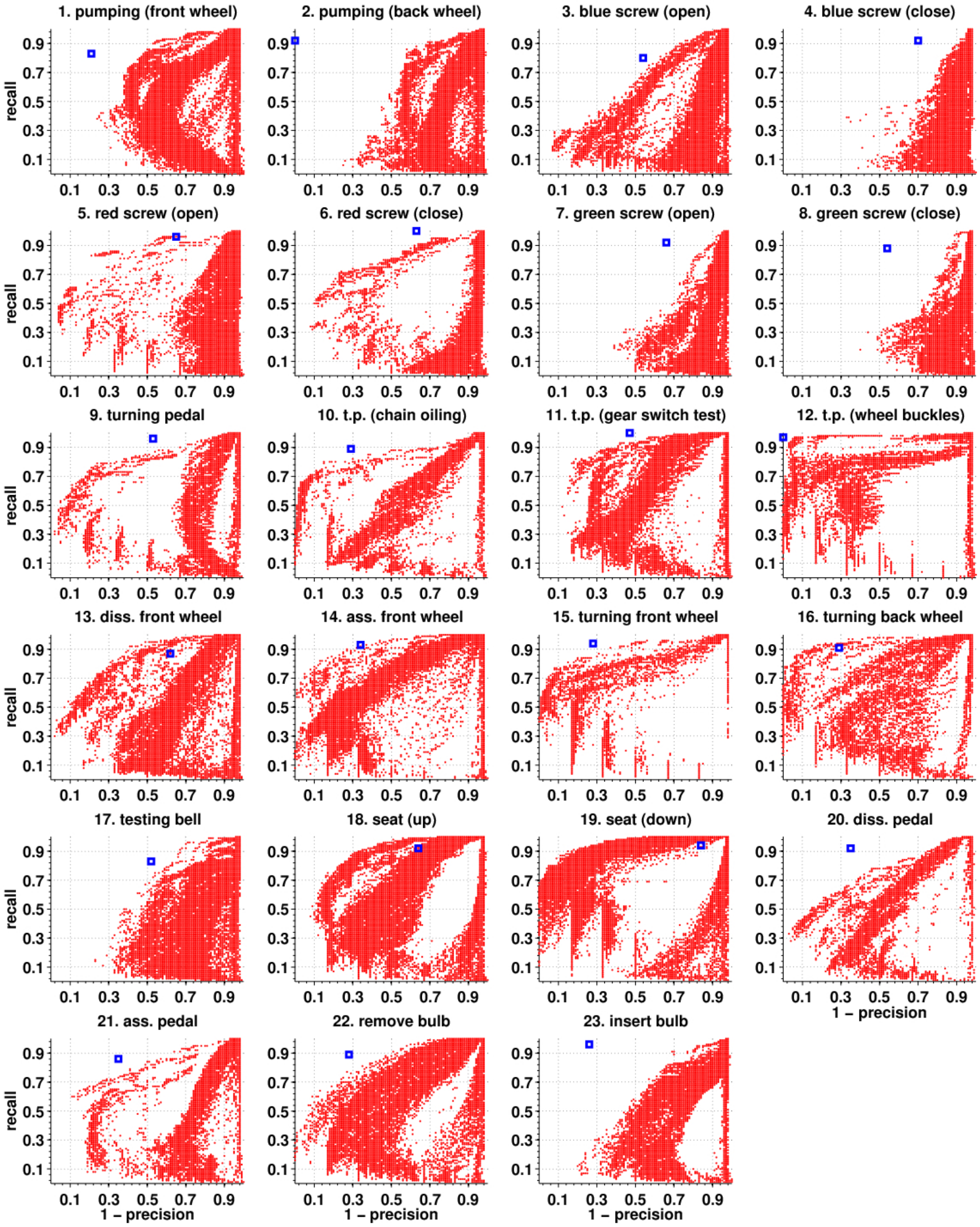


Figure 6.5: Bicycle repair. All classification results from our algorithm: red dots. Location trajectory-based spotting using polynomial matching cost features result from [99]: blue squares.

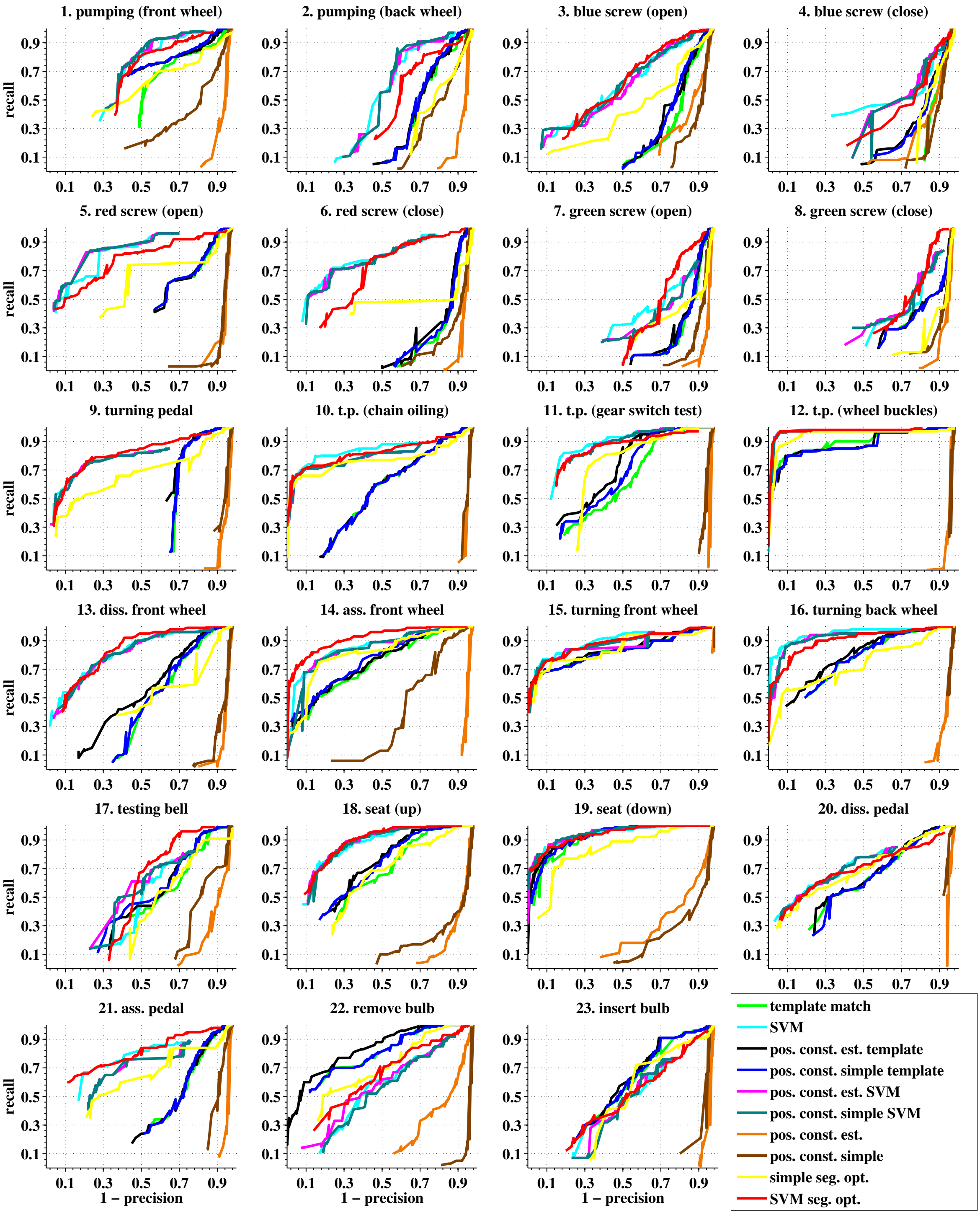


Figure 6.6: Bicycle repair. Classification results with all invariant detection strategies.

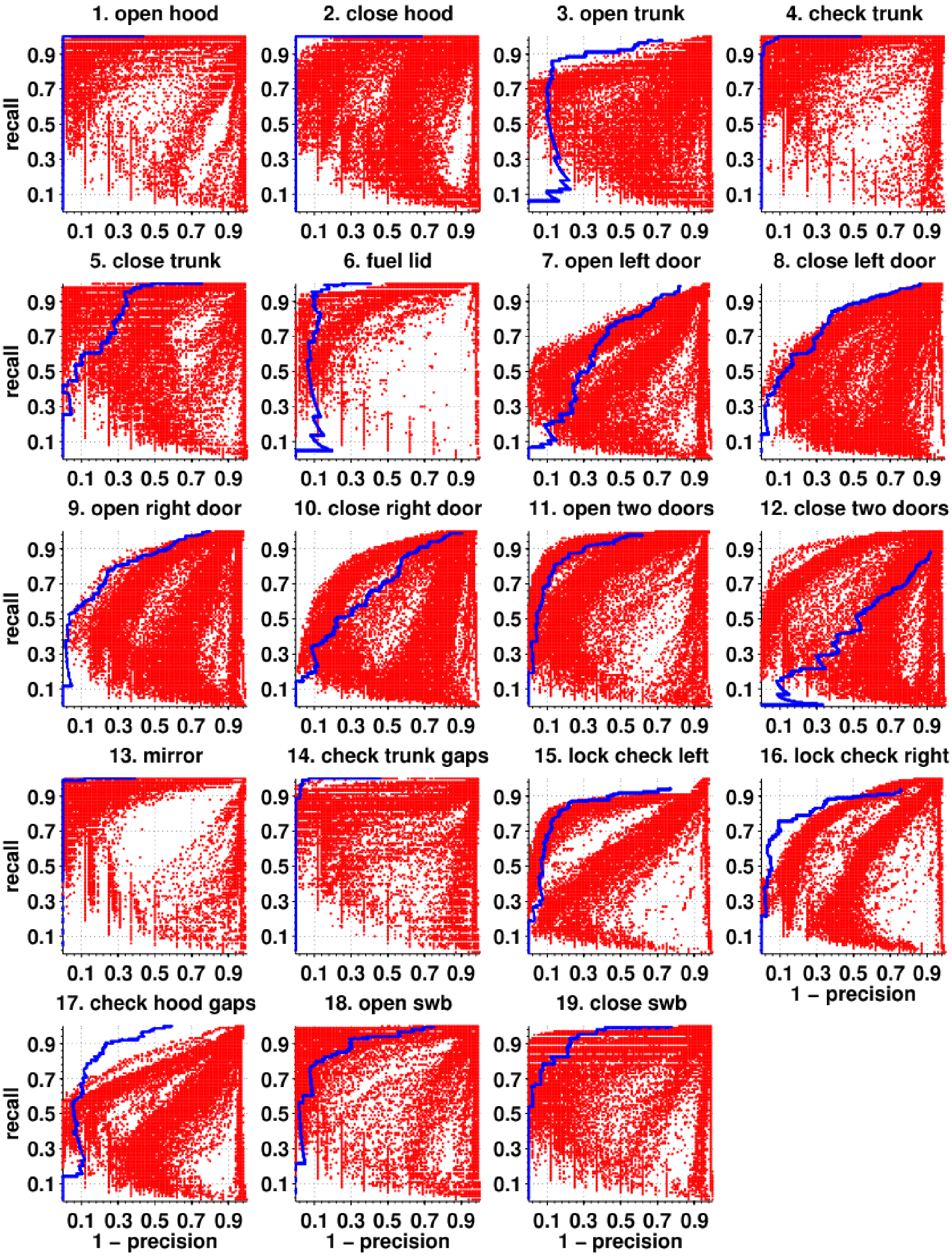

1. open hood
2. close hood
3. open trunk
4. check trunk
5. close trunk
6. fuel lid
7. open left door
8. close left door
9. open right door
10. close right door
11. open two doors
12. close two doors
13. mirror
14. check trunk gaps
15. lock check left
16. lock check right
17. check hood gaps
18. open swb
19. close swb

Figure 6.7: Car workshop. All classification results from our algorithm: red dots. Results from [142]: blue line.

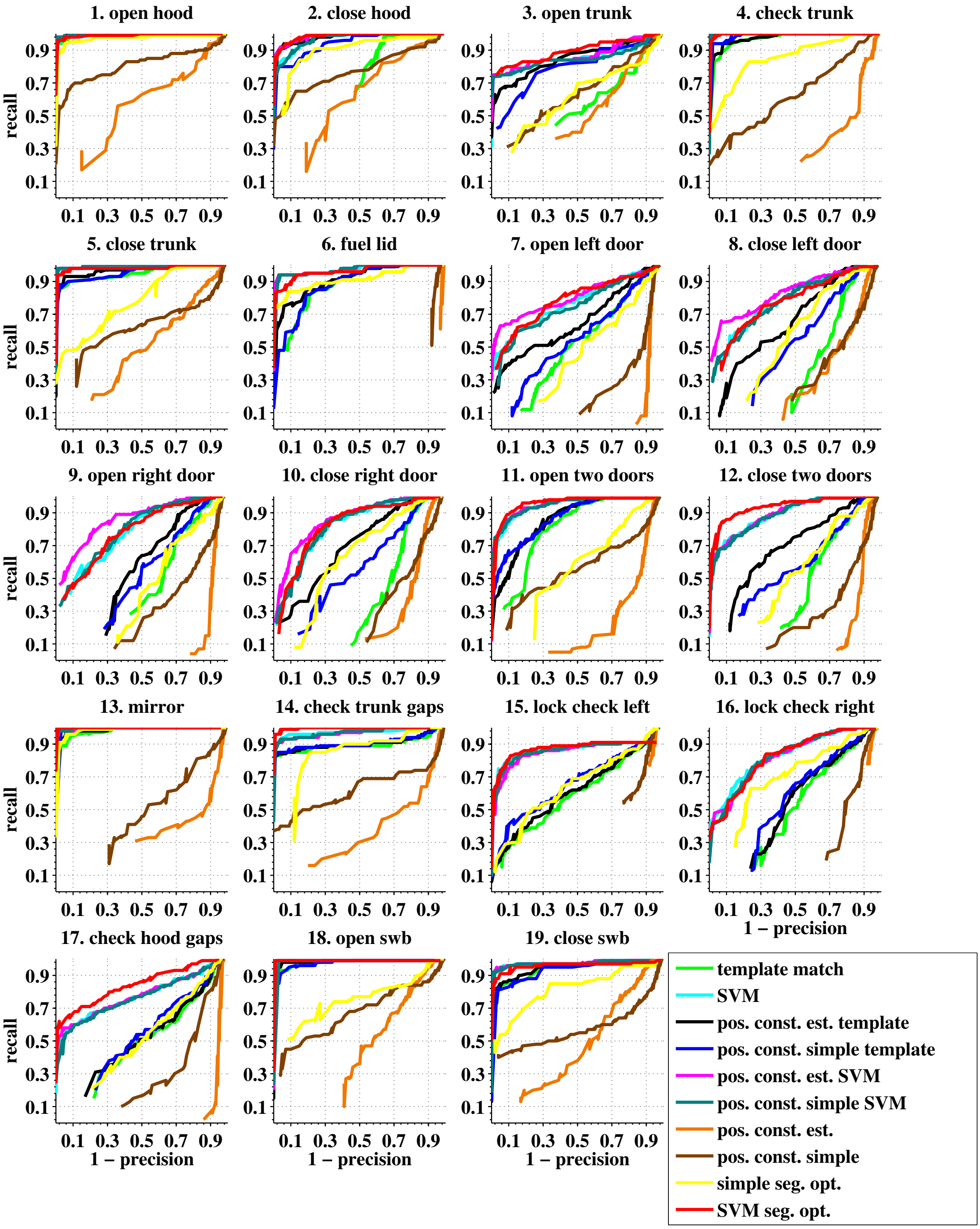


Figure 6.8: Car workshop. Classification results with all invariant detection strategies.

better precision in all classes. The *position constraints matching simple on SVM results* (turquoise) does not bring noticeable changes to the original classification with the *SVM model*, while *position constraints matching estimated on SVM results* (pink) is slightly better than the classification with the *SVM model.* The *position constraints matching simple on all data* (brown) ranks worst and *position constraints matching estimated on all data* (orange) comes second worst. Using only the instance position information seems to be not enough to classify the activities clearly. Only in the activity *drink from a cup* the *position constraints matching estimated on all data* shows a little bit better results. Finally the *template matching/SVM with simple segment optimization* (yellow) shows the best results in *drink from a glass*, while the *SVM with segment optimization* (bright red) is ranked best together with the *SVM* and the *SVM position constraints* in the other three activity classes.

Besides the original activity set of the *drink and work* experiment, we also test two other activity sets, where all four drink vessels are combined to one class. The first set, called *drink and work, one vessel*, includes the whole drink activity, like in the original experiment. The second set, called *drink and work, one vessel, sip*, includes the shortened activity only consisting of the drinking without the movements of the vessel to and from the mouth. We also analyze these two one-class activity sets because the publication [2] which serves as a basis for the comparison uses the same class arrangement. The pareto plots of the different classification strategies are presented in Figures 6.4b and 6.4c. Next, we will discuss both results together. Starting from the *template matching* (bright green), the *position constraints* brings some advancement: the *position constraints matching simple on template matching results* (blue) is quite similar to the pure *template matching* but the *position constraints matching estimated on template matching results* (black) achieves much better results. Next the *SVM* (cyan) clearly outperforms not only the *template matching* but also the *position constraints matching estimated on template matching results.* The *position constraints matching simple on SVM results* (turquoise) does not provide a better outcome for the two activity class sets, but the *position constraints matching estimated on SVM results* (pink) does provide a better result. The *position constraints matching simple on all data* (brown) and the *position constraints matching estimated on all data* (orange) return both again the worst results. Finally, the *template matching/SVM with simple segment optimization* (yellow) presents the second best classification results, behind the *position constraints matching estimated on SVM results.* The *SVM with segment optimization* (bright red) is better than the *template matching* but worse than the *SVM* and its related positon constraints. All results combined in one plot are depicted in Figures 6.3b and 6.3c. In contrast to the configuration with four classes, we can see a better classification outcome in both one class cases. The central idea of our algorithm is to find significant parts in the activity signal which describe the activity well. Maybe the problem with the four classes configuration is that the typical sensor signal snippet of the drink movement, the sipping, when the vessel is touching the mouth and then is turned so that the fluid can enter the body, is quite equal in all four drink activities. This makes it quite difficult to differentiate between the four drink classes which explains the improvements when they are combined to one class. Our results with one activity class are not as good but near to [2], probably because our result is user independent while the result of [2] is user specific.

Figures 6.6 depicts the pareto plots of the *invariants classification* results using different invariant detection strategies on the *bicycle repair* experiment. The *template matching* (bright green) shows problems in the precision values, especially in the activity classes: opening screws, closing screws, pumping front wheel, pumping back wheel, turning pedal, and assembling pedal. Only class 19, seat down, reaches nearly the recall and precision quality like in [99]. Results from the *position constraints matching simple on template matching results* (blue) and the *position constraints matching estimated on template matching results* (black) are quite similar but better as compared to the *template matching.* The *SVM classification* (cyan) provides better results in all activities than the *template matching* although we do not reach the top recall score of the *template matching* especially in classes 4, 7, 8, 9, 10, 15, 17, 20, 21, 22 and 23. Next the *position constraints matching simple on SVM results* (turquoise) and the *position constraints matching estimated on SVM results* (pink) show quite similar results as the basic *SVM.* The *position constraints matching simple on all data* (brown) and the *position constraints matching estimated on all data* (orange) return both bad outcomes with a bad precision. In classes 1, 2, 6, 7, 8, 11, 13, 14, 17, 18, 21 and 23 the *position constraints matching simple on all data* (brown) is better than the *position constraints matching estimated on all data.* The *template matching/SVM with simple segment optimization* (yellow) scores never best and is positioned in the mid-level, while the *SVM with segment optimization* (bright red) shows good results, even ranks best in class 14 and is equal with the best outcomes in classes 3, 7, 8, 9, 12, 13, 17, 18, 20 and 21. In class 22 *remove bulb* and class 23 *insert bulb* the *position constraints matching estimated on template matching results* returns the best result. All other classes are best classified by the *SVM* model alone and with its position constraints. The plot in Figure 6.5 combines all *invariants classification* outcomes from the *bicycle repair* experiment and compares them to [99]. In class 9, 12, 16, 18 and 19 we come near to the results of the comparative study. The worst classification of our algorithm can be found for classes, related to opening and closing a screw (class 3, 4, 7, 8). An explanation for our performance is that the gesture recognition in [99] builds upon the fusion of IMU sensors and ultrasonic sensors. The information about the location of the subject in the experiment is missing in our approach, so it is more challenging.

In the *car workshop* experiment all pareto plots of the *invariants classification* outcomes with different invariant detection strategies are shown in Figure 6.8. The *template matching* (bright green) already shows a good recognition in several classes. Class 18, open spare wheel box, already outperforms the comparative study [142] and also class 5, close trunk, has a better precision in the range of recall 0.9. Both the *position constraints matching simple on template matching results* (blue) and the *position constraints matching estimated on template matching results* (black) improve the basic *template matching* in most classes or at least deliver similar results. The *simple position* is only a bit better in classes 4, 13, 14, 15, 16 and 17, compared to *estimated segment position* while the other classes are little better in the *estimated segment position* case. Next the *SVM* (cyan) again improves the results we got from the *template matching* and also the related *position constraints matching.* While *position constraints matching simple on SVM results* (turquoise) nearly shows the same plot as without the position constraints, *position constraints matching estimated on SVM results* (pink) provides some improvements, noticeable in classes 2, 3, 7, 8, 9 and 10. The *position constraints matching simple on all data* (brown) provides better results than the *position constraints matching estimated on all data* (orange), although both rank worst in the comparison to the other invariant detection strategies. The *template matching/SVM with simple segment optimization* (yellow) provides classification results in the mid-level. In contrast to this, the related *SVM with segment optimization* (bright red) tops the outcomes of the other invariant detection strategies in classes 3, 11, 12, 14 and 17 and is equal to the top ranking strategies in classes 2, 13, 15 and 18. In the other classes the best results are shown by the *SVM classification* and the related *position constraints matching* strategies. In the combination of all results in Figure 6.7 we see that our algorithm outperforms [142] in classes 4, 5, 7, 8, 9, 10, 11, 12, 15, 18 and 19 while the results in classes 1, 6, 13 and 14 are equal, and 2, 3, 16 and

17 are worse.

6.3.2 Parameter Training

As explained before we have to optimize seven parameters to get a final classification result: best invariant model configuration, basic or optimized segmentation strategy, sort strategy for the invariant models, number of used invariants, number of needed invariants, tolerance threshold for the activity position model, start - end - start and end. For the training of these seven parameters, we need to find the best classification outcome of the *class instance model* which reveals the best parameter set. In the Evaluation section, we showed that we use two connected cross-validation strategies, depicted in Figure 6.2.

The first cross-validation on the training data of the second cross-validation is used for the parameter training (the blue part of Figure 6.2). The best classification outcome, measured as mean F1-scores of the first cross-validation on one training dataset, shows us the optimal parameter set for the application of the classifiers. In the second cross-validation, we determine the final classification result using the optimal parameter set.

Next, we explain the parameter training for the *invariants classification* in detail. We first sort the set of invariants, dependent on the sort strategy for the invariant models' parameter. There are four different sort strategies:

1. F1-score of the *template matching*
2. precision of the *template matching*
3. F1-score of the *SVM*
4. precision of the *SVM*

The precision or F1-score value for the ordering is determined by applying the trained invariant model on its training data. Next, we take the first m invariants, where m is the value of the parameter of used invariants. Then we sum up the number of invariants which contain found segments inside an activity instance (only one segment per invariant is counted). We compare this sum to the value k, which is the second parameter, the number of needed invariants. If the sum is greater or equal to k, then the activity instance counts as found activity class instance. We vary the two parameters in the range of 1 to 60. The parameter of used invariants is combined with the parameter of needed invariants by setting the parameter of needed invariants to all numbers between 1 and the actual parameter value of used invariants. This leaves us with 1830 ($= \frac{60^2+60}{2}$) different parameter combinations.

To decide for each activity class which is the best invariant model (*template model*, *SVM model* or *activity position constraints model*) with its parameter (tolerance threshold of the *activity position constraints model*), the best invariant models order and whether the basic or the optimal segmentation is better, we test each option of these parameters varying the used and needed variants as described before. This returns a classification result expressed in precision, recall and F1-score values, for each configuration of the seven parameters. As we use a cross-validation scheme, we get n classification results, where n is the number of train datasets. We calculate the mean precision, the mean recall, the mean F1-score, and the variance F1-score from the n results. These values are used to select the best parameter. This is done by sorting the classification results on these values. We tried the following twelve sort criteria which we call *final sort criteria*. If there is more than one sort criterion in a set, we use the second criterion for equal values in the first sort criterion and the third value for equal values in the first and second sort criterion. A descending (desc.) order for criteria means that the largest value is best and an ascending (asc.) order means that the smallest value is best.

1. I. mean F1 (desc.)
2. I. mean F1 (desc.) II. recall (desc.)
3. I. mean F1 (desc.) II. precision (desc.)
4. I. mean F1 (desc.) II. used invariants (asc.) III. needed invariants (asc.)
5. I. mean F1 (desc.) II. used invariants (desc.) III. needed invariants (desc.)
6. I. mean F1 (desc.) II.variance F1 (asc.)
7. I.precision (desc.) II. mean F1 (desc.)
8. I. precision (desc.) II. used invariants (asc.) III. needed invariants (asc.)
9. I.precision (desc.) II. variance F1 (asc.)
10. I. recall (desc.) II. mean F1 (desc.)
11. I. recall (desc.) II. used invariants (asc.) III. needed invariants (asc.)
12. I. recall (desc.) II. variance F1 (asc.)

We select the first position of each of the twelve ordered results. The best one of these twelve defines then the optimal *final sort criterion* and the related parameters for the final classification. The final *invariants classification* model is trained with all train data, diverse to the training of the parameters where we split the training dataset in the cross-validation scheme and for this purpose train the invariant models only with a subset of the training data.

6.4 Invariant vs. Normal Classifier Selection

Next, we want to generate a combination of the conventional classifier and our *invariants classification*. To calculate the *invariants classification* results, we determine the best parameters as described in the previous section. The best *class instance model*, which consists of the best set of invariant models and parameters, is then applied on segments which we want to classify. The results are classified activity instances. As mentioned before we compare our *invariants classifier* results with the results of conventional classifiers used in other publications: the *drink and work* scenario with [2], the *bicycle repair* scenario with [99] and the *car workshop* scenario with [142]. To decide whether the conventional or the *invariants classification* strategy is more suitable, we check for each class activity the F1-score of the conventional classifier against the F1-score of our *invariants classifier* on training data and then use the classifier with the higher outcome (in the final decision a factor *fac* is added beforehand to the conventional classifier). The F1-score of the conventional classifier is taken from the related publications. The F1-score of the *invariants classifier* on training data was already calculated in the parameter training. Here we perform a leave-one-out cross-validation on the training data, which leaves us for each best parameter set with n F1-scores, where n is the number of training subjects (each subject represents a test subset of the cross-validation). Therefore the final F1-score of the training data is the mean F1-score of the n F1-scores of the cross-validation

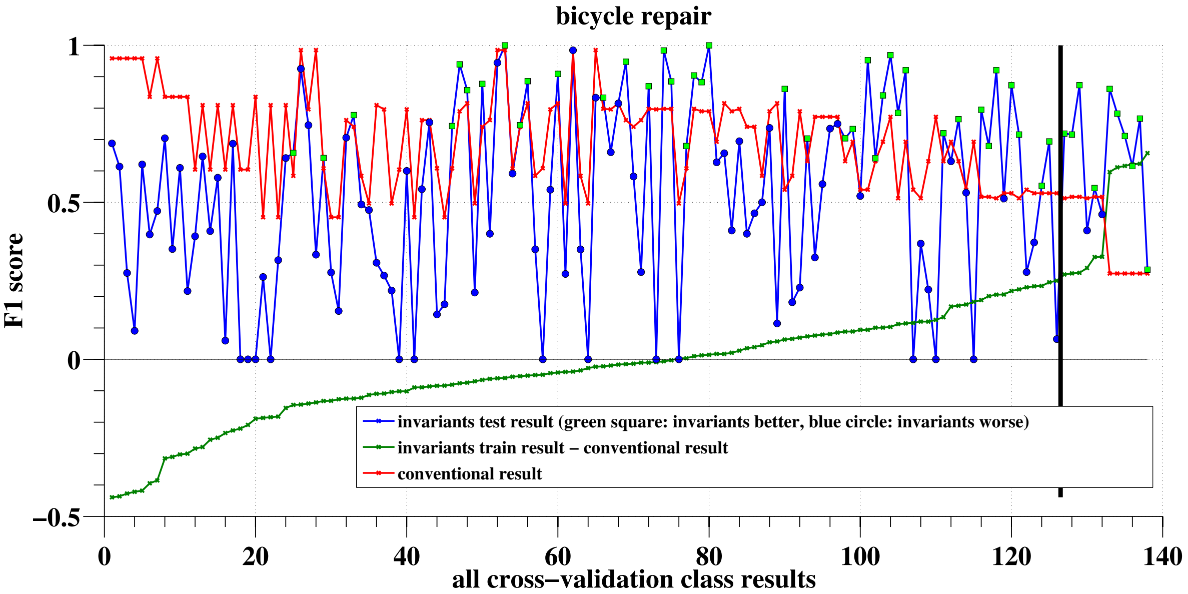


Figure 6.9: Bicycle repair experiment

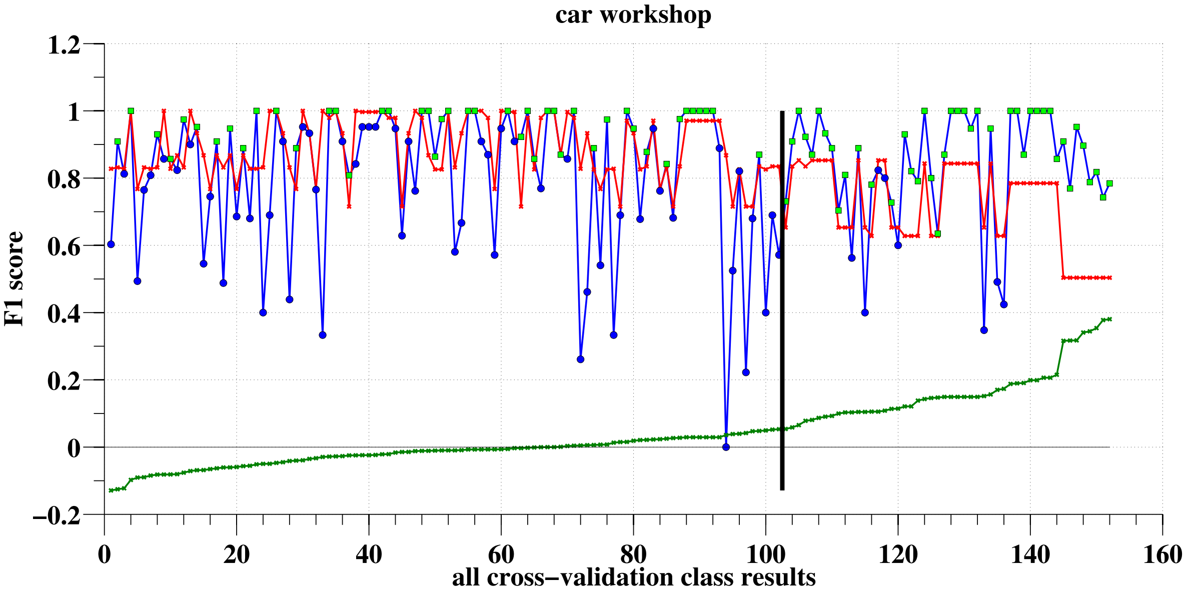


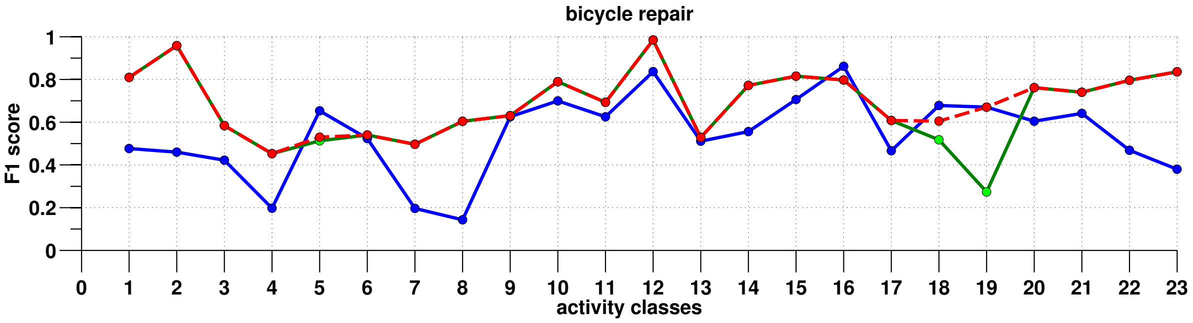
Figure 6.10: Car workshop experiment

Table 6.1: Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both applied on the drink and work, the bicycle and the car workshop experiment.

		$\varnothing pre$	$\varnothing rec$	$\varnothing$ F1
drink & work 1 ves.	invariants classifier	0.84	0.83	0.83
	conventional classifier	0.84	0.90	0.87
	fusion	0.84	0.87	0.85
drink & work 1 ves. sip	invariants classifier	0.81	0.90	0.85
	conventional classifier	0.84	0.94	0.89
	fusion	0.83	0.95	0.89
bicycle repair	invariants classifier	0.59	0.62	0.54
	conventional classifier	0.57	0.91	0.67
	fusion	0.61	0.89	0.70
car workshop	invariants classifier	0.83	0.86	0.82
	conventional classifier	0.82	0.85	0.83
	fusion	0.87	0.89	0.87

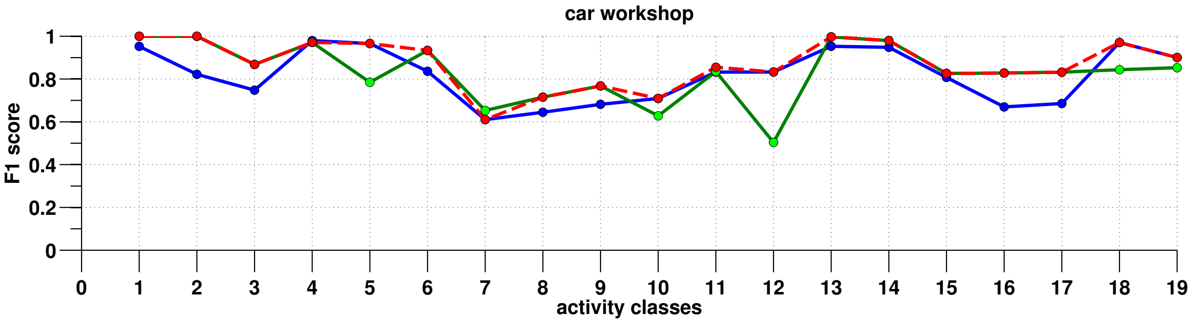
on the training data. For the final result of the *invariants classification* we use a leave-one-out cross-validation on the complete dataset. Thus we get k F1-score results, where k is the number of all experiment subjects (each subject represents a test subset of the cross-validation).

In Figures 6.9 and 6.10 the F1 scores of conventional and *invariants classifier* are plotted on the y-axis. The x-axis of the plot indicates all classes of all final cross-validation test subsets. The red line is the F1-score results of the conventional classifiers. The green line is the F1-score difference between *invariants classification* results on train data and conventional classification results. The blue line is the result of the final application of the *invariants classifier*. The green squares on the blue line show that the *invariants classifier* returns a better result than the conventional classifier and a blue dot on the blue line marks the opposite. We see in both plots, that if the F1-score difference of the *invariants classifier* on train data and the conventional classifier becomes more positive, the final application of the *invariants classifier* also returns better F1-score results than the conventional classifier. The vertical black line shows the border which we selected to get the classifier fusion outcome: everything on the right of the line is classified with the *invariants classifier* and everything left with the conventional classifier. This threshold is determined by the F1-score difference between *invariants classifier* on train data and conventional classifier plus a factor *fac*. In the final result calculation we take the twelve best *final sort criteria* results, perform for each a sweep over *fac* and take as final result the one which has the best F1-score. If there are results with same F1-score values, we sort them by the number of worse class F1-score results compared to the conventional results, the smaller, the better. If there are still equal results, then we sort by the number of better class F1-score results compared to the conventional results, the greater, the better.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
invariants classifier	0.48	0.46	0.42	0.20	0.65	0.52	0.20	0.14	0.63	0.70	0.63	0.84	0.51	0.56	0.71	0.86	0.47	0.68	0.67	0.60	0.64	0.47	0.38
conventional classifier	0.81	0.96	0.58	0.45	0.51	0.54	0.50	0.60	0.63	0.79	0.69	0.98	0.53	0.77	0.82	0.80	0.61	0.52	0.27	0.76	0.74	0.80	0.84
fusion	0.81	0.96	0.58	0.45	0.53	0.54	0.50	0.60	0.63	0.79	0.69	0.98	0.53	0.77	0.82	0.80	0.61	0.61	0.67	0.76	0.74	0.80	0.84

Figure 6.11: Mean F1 score results over all subjects per class of the invariants classifier (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
invariants classifier	0.95	0.82	0.75	0.98	0.97	0.84	0.61	0.64	0.68	0.71	0.83	0.83	0.95	0.95	0.81	0.67	0.69	0.97	0.90
conventional classifier	1.00	1.00	0.87	0.97	0.78	0.93	0.65	0.72	0.77	0.63	0.83	0.50	1.00	0.98	0.83	0.83	0.83	0.84	0.85
fusion	1.00	1.00	0.87	0.97	0.97	0.93	0.61	0.72	0.77	0.71	0.86	0.83	1.00	0.98	0.83	0.83	0.83	0.97	0.90

Figure 6.12: Mean F1 score results over all subjects per class of the invariants classifier (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment.

6.5 Results

In Table 6.1 we see the overall results of our *invariants classifier*, the conventional classifiers and the fusion of both, shown as mean precision, mean recall and mean F1-score. In the *drink and work, one vessel* experiment, the *drink and work, one vessel, sip* experiment, the *bicycle repair* experiment and the *car workshop* experiment, our invariants algorithm returns a worse result than the conventional classifiers which can be seen in the mean F1-scores. The classification outcome of our algorithm in the *drink and work* experiment is worse compared to [2] because [2] uses user-specific spotting while we provide an user-independent system. The reason why our algorithm is worse in the *bicycle repair* experiment could be the additional information from an ultrasonic sensor system, used in [99]. However, our algorithm delivers, in this case, a better precision than the conventional classification algorithms.

The fusion of both classification strategies performs worse than the conventional classifier in the *drink and work, one vessel* experiment but equal in the *drink and work, one vessel, sip* experiment. Figures 6.3b and 6.3c, which depict all possible classification results, already show that we do not reach the best classification results of the conventional classifier. The *invariants classifier* returns decent results applied on the *drink and work* experiment. However, the fusion of invariants and conventional classifier brings no im-

Table 6.2: Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment.

classify strategy	bicy. inv.	bicy. fusion	car inv.	car fusion
template match	2	0	0	0
SVM	33	25	14	18
pos. const. est. template	9	17	5	0
pos. const. simple template	1	0	1	0
pos. const. est. SVM	17	17	39	46
pos. const. simple SVM	16	17	6	10
simple seg. opt. template	1	0	0	0
simple seg. opt. SVM	4	0	0	0
SVM seg. opt.	17	25	34	26

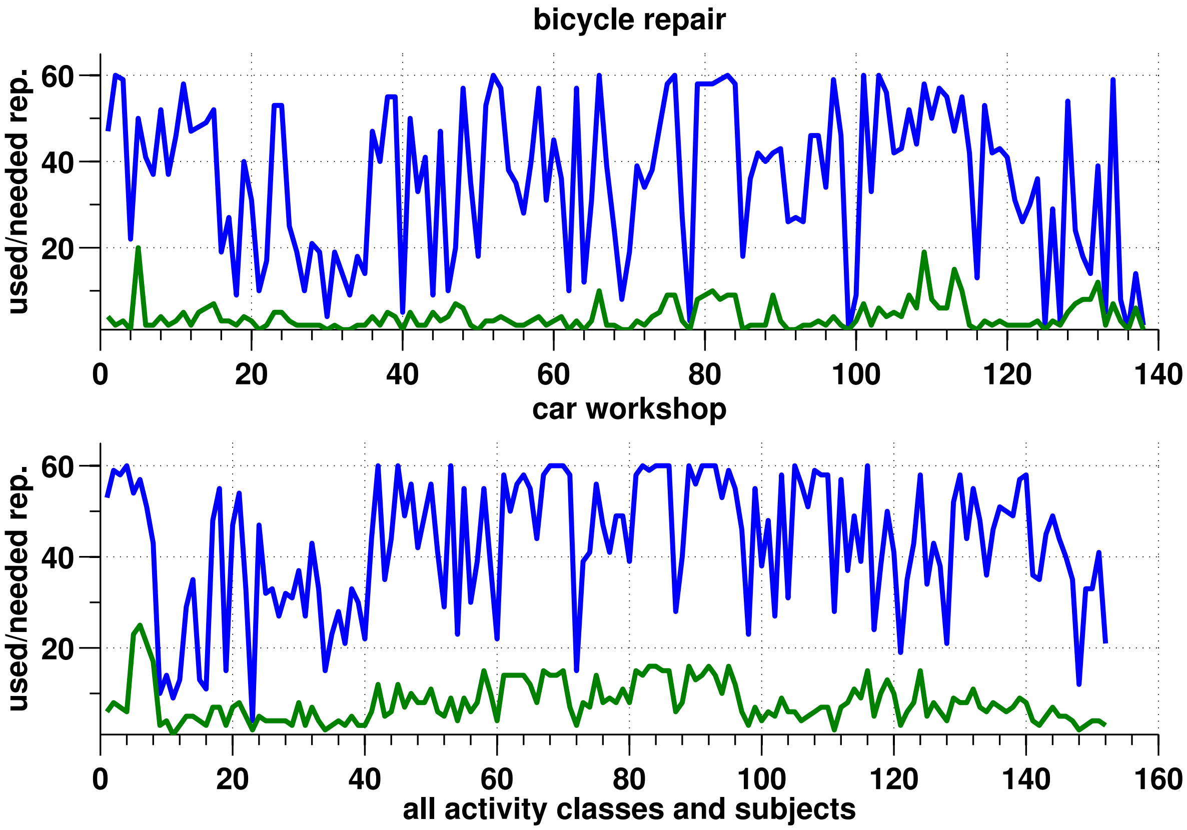


Figure 6.13: Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments.

provement so we stick to the conventional classifier. The two other experiments consist of several classes and show in several classes that the *invariants classifier* provides a better outcome than the conventional classifier. This can also be seen in the mean F1-score of all classes applying the fusion classification to the *bicycle repair* and the *car workshop* experiment (Table 6.1). Here the result is better than the result of conventional classifiers which shows that our invariants algorithm provides a benefit in the classification. To get more insight, we take a look at the classification results of the single activity classes.

We start with the *bicycle repair* experiment. In Figure 6.11, the fusion results for classes 5, 18 and 19 are better than the conventional classification results. These are the classes of the *bicycle repair* experiment: (5) red screw (open), (18) seat (up) and (19) seat (down). Especially classes 18 and 19 show a significant improvement. Figure 6.12 shows the *car workshop* experiment results of our algorithm, the conventional classifiers and the fusion for each activity class. The fusion results for classes 5, 10, 11, 12, 18 and 19 are better than the conventional classification results. These are the following activities: (5) close trunk, (10) close right door, (11) open two doors, (12) close two doors, (18) open spare wheel box and (19) close spare wheel box. The classifier fusion on classes 5, 10, 12, 18 and 19 return much better results than the conventional classifier. The results of the invariants and the conventional classifier are equal in class 11, but the fusion shows an improvement in the classifications. The reason is that in the cross-validation always the better classifier strategy (conventional/ invariants) was chosen which is reflected in the better mean F1-score. Class 7, open left door, has a slightly lower outcome in the fusion case than the conventional classifier.

In our algorithm, we tested several invariant detection strategies with three different invariant models and two segmentation strategies. For each activity class we construct a *class instance model* for the final activity classification. Every *class instance model* consists of invariant models of only one particular invariant detection strategy. In Table 6.2 we see the distribution of the invariant detection strategies which were chosen by applying the best parameter set. The above table shows the invariant detection strategies which are used in the *invariants classification* and the fusion of *invariants* and conventional classification. The *bicycle repair* experiment has most diverse invariant models and segmentation strategies among all experiments. Only the sole *activity position constraints model* is not represented, which is understandable, as the *position constraints matching simple on all data* and the *position constraints matching estimated on all data* already showed always the worst results in the plots of all possible results in Section 6.3.1. Both experiments also have more classification strategy variability in the pure *invariants classification* results. In the fusion results, the classification strategies are condensed to the best classification strategies, because only suitable classes are classified with the *invariants classification.* The best classification strategies for both experiments, according to the fusion results, are following:

- SVM
- SVM segment optimization classification
- position constraints matching simple on SVM results
- position constraints matching estimated on SVM results
- position constraints matching estimated on template matching results (only *bicycle repair*)

In all three experiments the best *final sort criteria* was the *mean F1-score (descending order).* Figure 6.13 shows the number of used and needed invariants which were selected in the training for the different classes and subjects of both experiments. The number of used invariants is relatively high while the needed invariants are in the *bicycle repair* and *car workshop* experiment usually below 20. This shows that we need only a small number of invariants to identify our activity classes.

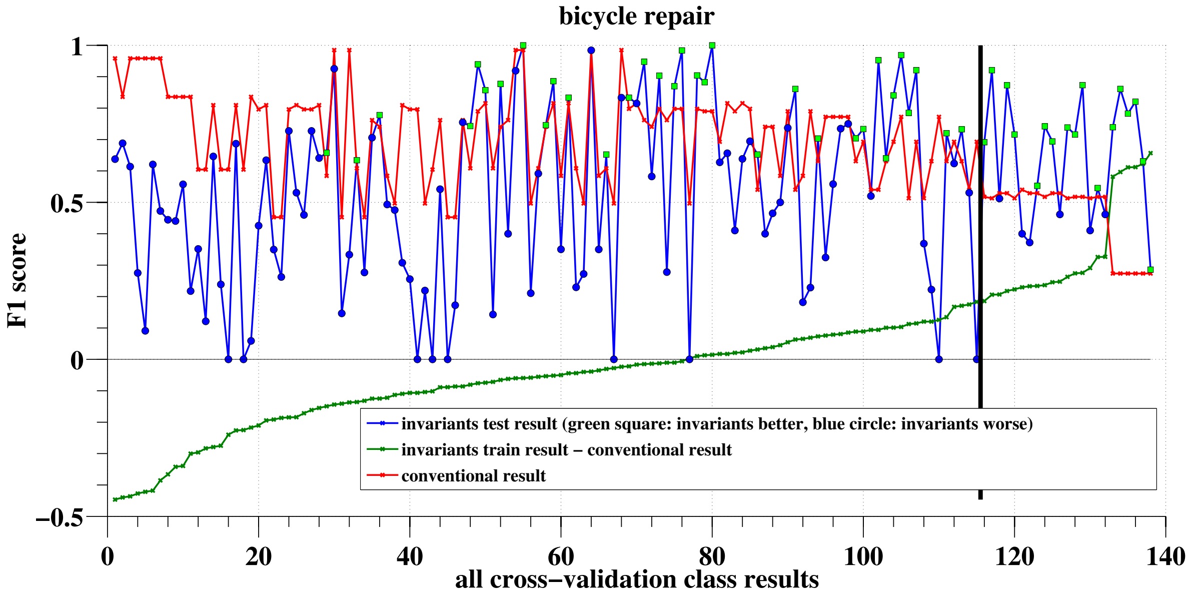


Figure 6.14: Bicycle repair with *invariant detection strategy set variation.*

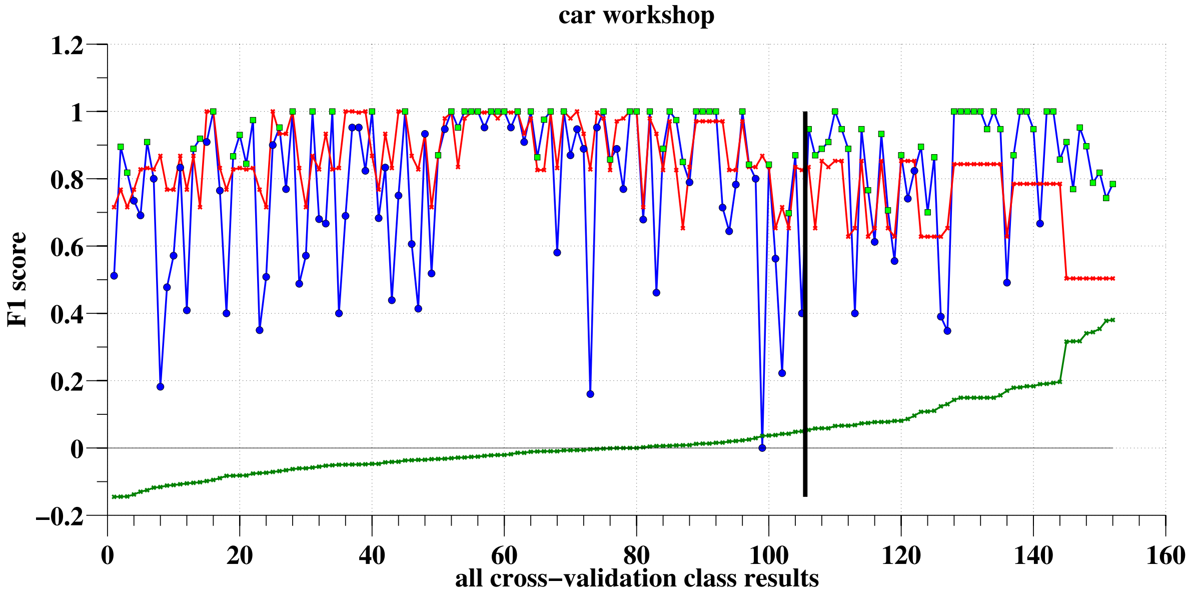


Figure 6.15: Car workshop experiment with *invariant detection strategy set variation.*

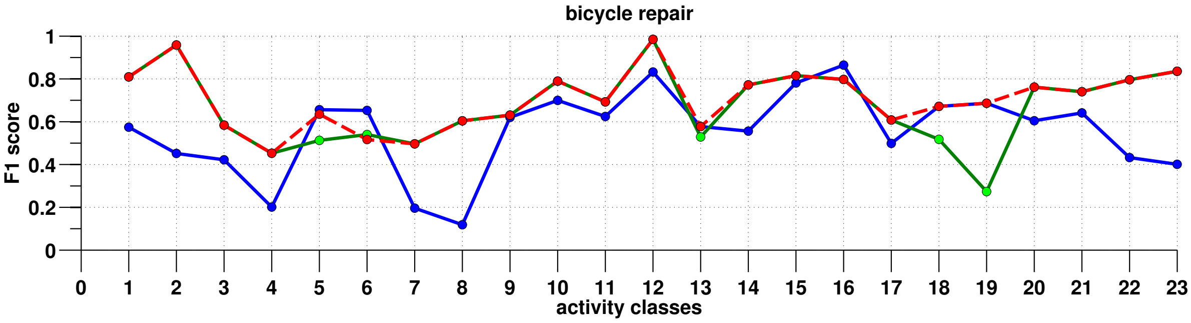
6.6 Invariant Detection Strategy Set Variation

Here we want to check what happens if we use only a subset of the invariant detection strategies when we construct the *class instance model.* In the classification fusion we compare the conventional classification results with the *invariants classification* results on training data. If we use only a subset of the invariant detection strategies, it is possible that we get better final results. When we use the complete set, it may occur that a *class instance model* with a certain invariant detection strategy returns good classification results on the training data but bad ones in the final classification. If this invariant detection strategy is excluded in the parameter training for the *class instance model*, another one is then selected which maybe returns a better final classification result. We tested the permutation of eleven invariant detection strategies:

1. template matching
2. SVM
3. position constraints matching simple on template matching results
4. position constraints matching estimated on template matching results
5. position constraints matching simple on SVM results
6. position constraints matching estimated on SVM results
7. position constraints matching simple on all data
8. position constraints matching estimated on all data
9. template matching with simple segment optimization classification
10. SVM with simple segment optimization classification
11. SVM with segment optimization

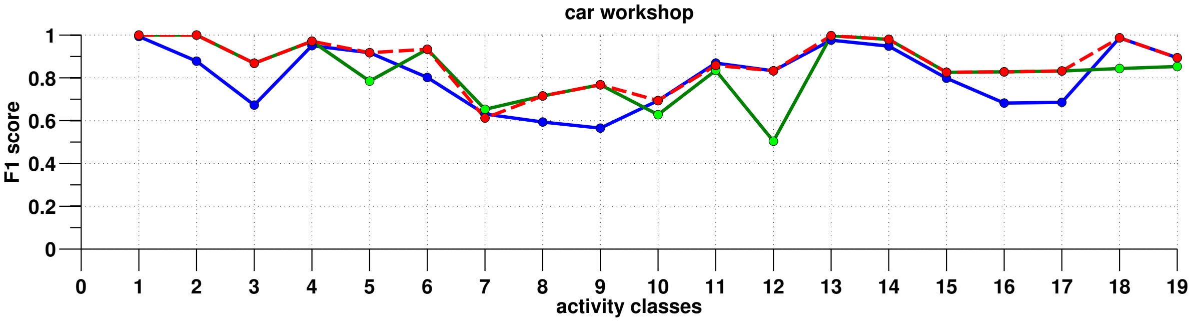
The best invariant detection strategy for the *drink and work, one vessel* and the *drink and work, one vessel, sip* experiments is the *position constraints matching simple on SVM results* combined with *final sort criteria mean F1-score (descending order).* The best invariant detection strategy subset for the *bicycle repair* experiment is the combination of *SVM*, *position constraints matching estimated on SVM results*, *SVM with simple segment optimization*, and *SVM with segment optimization.* Here also it is the only time when a *final sort criteria* differs from the normal *mean F1-score (descending order)*: the *1. mean F1-score (descending order) 2. used invariants (descending order) 3. needed variants (descending order)* returns the best final results. Finally the *car workshop* experiments achieves the best outcome with the invariant detection strategy *SVM with segment optimization* and the *final sort criteria mean F1-score (descending order).*

In Figures 6.14 and 6.15 the F1 scores of conventional and *invariants classifier* are plotted on the y-axis for the *bicycle repair* and *car workshop* experiments The x-axis of the plot indicates all classes of all final cross-validation test subsets. The red line is the F1-score results of the conventional classifiers. The green line is the F1-score difference between *invariants classification* results on train data and conventional classification results. The blue line is the result of the final application of the *invariants classifier.* The green squares on the blue line show that the *invariants classifier* returns a better result than the conventional classifier and a blue dot on the blue line marks the opposite.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
invariants classifier	0.57	0.45	0.42	0.20	0.66	0.65	0.20	0.12	0.62	0.70	0.62	0.83	0.58	0.56	0.78	0.86	0.50	0.67	0.69	0.60	0.64	0.43	0.40
conventional classifier	0.81	0.96	0.58	0.45	0.51	0.54	0.50	0.60	0.63	0.79	0.69	0.98	0.53	0.77	0.82	0.80	0.61	0.52	0.27	0.76	0.74	0.80	0.84
fusion	0.81	0.96	0.58	0.45	0.64	0.52	0.50	0.60	0.63	0.79	0.69	0.98	0.58	0.77	0.82	0.80	0.61	0.67	0.69	0.76	0.74	0.80	0.84

Figure 6.16: Mean F1 score results over all subjects per class of the invariants classifier with *classification model set variation* (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
invariants classifier	0.99	0.88	0.67	0.95	0.92	0.80	0.63	0.59	0.57	0.69	0.87	0.83	0.98	0.95	0.80	0.68	0.69	0.99	0.89
conventional classifier	1.00	1.00	0.87	0.97	0.78	0.93	0.65	0.72	0.77	0.63	0.83	0.50	1.00	0.98	0.83	0.83	0.83	0.84	0.85
fusion	1.00	1.00	0.87	0.97	0.92	0.93	0.61	0.72	0.77	0.69	0.86	0.83	1.00	0.98	0.83	0.83	0.83	0.99	0.89

Figure 6.17: Mean F1 score results over all subjects per class of the invariants classifier with *classification model set variation* (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment.

Table 6.3: Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both with *invariant detection strategy set variation* applied on the drink and work, the bicycle and the car workshop experiment.

		∅*pre*	∅*rec*	∅ F1
drink & work 1 ves.	invariants classifier	0.80	0.84	0.81
	conventional classifier	0.84	0.90	0.87
	fusion	0.84	0.89	0.87
drink & work 1 ves. sip	invariants classifier	0.81	0.90	0.85
	conventional classifier	0.84	0.94	0.89
	fusion	0.83	0.95	0.89
bicycle repair	invariants classifier	0.62	0.63	0.56
	conventional classifier	0.57	0.91	0.67
	fusion	0.64	0.88	0.71
car workshop	invariants classifier	0.82	0.86	0.81
	conventional classifier	0.82	0.85	0.83
	fusion	0.87	0.89	0.87

Table 6.4: Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment.

classify strategy	bicy. inv.	bicy. fusion	car inv.	car fusion
SVM	40	48	0	0
pos. const. simple SVM	35	26	0	0
simple seg. opt. SVM	4	4	0	0
SVM seg. opt.	21	22	100	100

We see in both plots, that if the F1-score difference of the *invariants classifier* on train data and the conventional classifier becomes more positive, the final application of the *invariants classifier* also returns better F1-score results than the conventional classifier. The vertical black line shows the border which we selected to get the classifier fusion outcome: everything on the right of the line is classified with the *invariants classifier* and everything left with the conventional classifier. This threshold is determined by the F1-score difference between *invariants classifier* on train data and conventional classifier plus a factor *fac*. Comparing Figure 6.14 with Figure 6.9, we see that we can now select the threshold which divides the conventional from the *invariants classification*, more to the left. This means, that due to better classification results, the *invariants classifier* is used more often. Figures 6.15 and 6.10 show a similar position of the frontier between conventional and *invariants classification* selection.

In Table 6.3 the final mean precision, mean recall and mean F1-score results of all three experiments using the *invariant detection strategy set variation* are depicted. We Compare these results to the results we get when using all invariant detection strategies, which is shown in Table 6.1. In the *invariant detection strategy set variation* case the mean F1-score of the *invariants classifier* improves in the *bicycle repair* experiment, stays equal in the *drink and work, one vessel, sip* experiment and worsens in the *drink and*

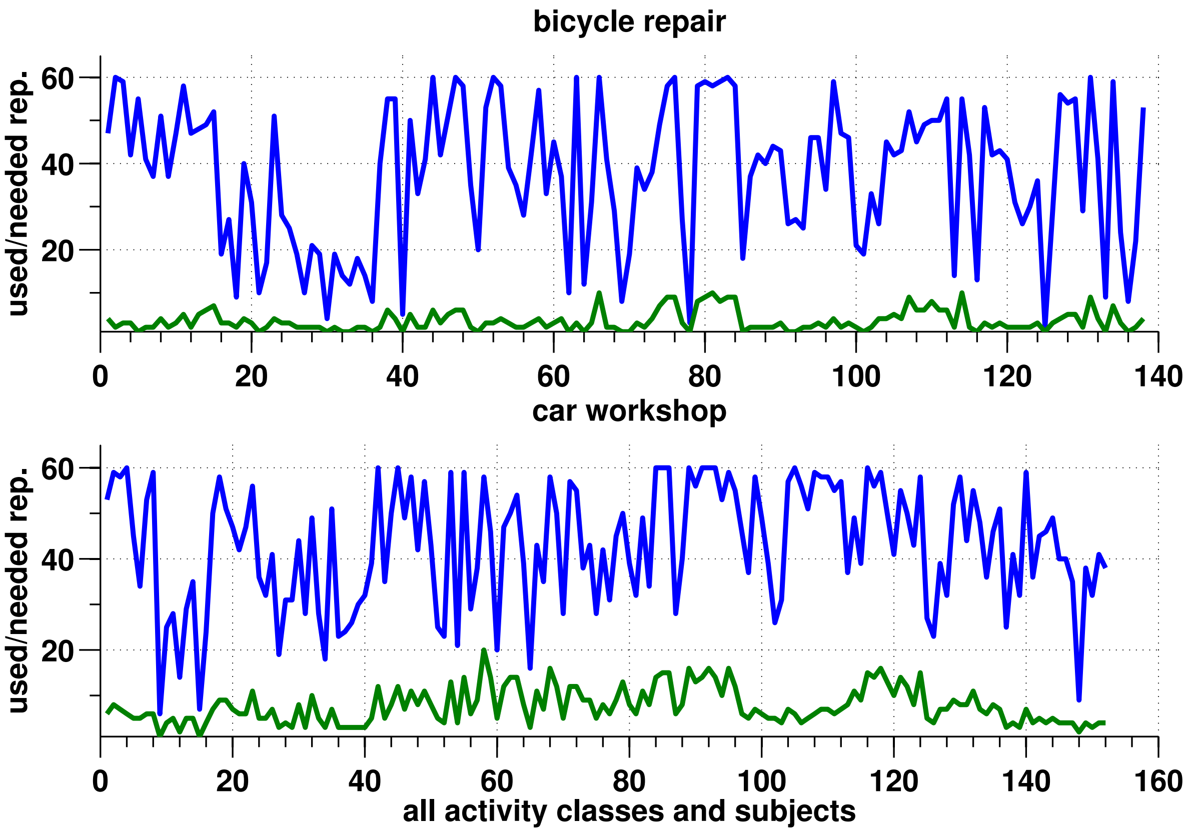


Figure 6.18: Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments (*classification model set variation*).

work, one vessel and the *car workshop* experiment. In the classifier fusion, the *invariant detection strategy set variation* shows an improvement in the F1-score in *drink and work, one vessel* and the *bicycle repair* experiment while the other outcomes stay equal. The *drink and work* experiment variations provide an equal or better F1-score outcome of the conventional classification compared to the fusion classification. This means that we can classify them with the conventional classification. The other two experiments get an enhancement by using the *invariants classifier*. Therefore we look into the outcome of the single activity classes. Figure 6.16 shows the *bicycle repair* experiment results. The fusion results for classes 5, 13, 18 and 19 are better than the conventional classification results. These are the following activities: (5) red screw (open), (13) disassembling front wheel, (18) seat (up) and (19) seat (down). All four classes show a drastic improvement. The F1-score of class 6 drops. But this class already has a bad classification which makes it insignificant in comparison to the gain from the four better detected classes. In total we see here better outcomes compared to the classification using all invariant detection strategies. We also have one more class with enhancements and one more with worsening. This explains the better mean F1-score results in the overall result. In Figure 6.17, the fusion results for classes 5, 10, 11, 12, 18 and 19 are better than the conventional classification results. These are the classes of the *car workshop* experiment: (5) close trunk, (10) close right door, (11) open two doors, (12) close two doors, (18) open spare wheel box and (19) close spare wheel box. Class 7 open left door has a slightly lower outcome in the fusion case than the conventional classifier.

In this variation of our new algorithm we preselected the set of invariant detection strategies of the *class instance model* and therefore it is interesting to see which invariant detection strategies are finally used in the *invariants classification* and the fusion classification. In Table 6.4 we see the distribution of the invariant detection strategies which were chosen by applying the best parameter set in the *class instance model* creation.

The *bicycle repair* experiment again has the most diversified invariant models and segmentation strategies. In both experiments, all preselected invariant detection strategies are also needed in the fusion case. The strategy priorities vary a bit compared to the *invariants classification* with all invariant detection strategies in Table 6.2. But still the most important strategies are:

1. SVM with segment optimization
2. SVM
3. position constraints matching simple on SVM results

Figure 6.18 shows the number of used and needed invariants which were selected in the training for the different classes and subjects of both experiments. The number of used invariants in both experiments is quite similar compared to Figure 6.13. The number of needed invariants is a little bit lower in the *invariant detection model set variation* case: below 20 in the *bicycle repair* and usually below 10 in the *car workshop* experiment.

6.7 Summary

This chapter introduces the *class instance model* which combines several invariant models for the final classification of activity instances. There exist seven parameters which have to be trained in the process of the *class instance model* creation. First, we show several *invariants classification* results in form of precision - recall plots after a sweep over all these parameters. We also depict here the classification results of conventional classifiers. This gives us a first insight into the classification capability of the *invariants classifier* after fusing several invariant models. Next, we introduced the parameter training. Here we use two connected leave one out cross-validation. The first one is for the parameter training and operates on the training data of the second cross-validation. The second cross-validation is for the final classification result. A further step is to fuse conventional classifiers with the *invariants classifier.* We decide which classifier to utilize by comparing the F1-scores of the conventional classifier with the F1-scores of the *invariants classifier* on training data. The classifier with the higher F1-score is chosen. We then present the classification results of the three evaluation experiments, showing that the *invariants classifier* provides a recognition improvement in the classifier fusion case for the *bicycle repair* and the *car workshop* experiment. As last step, we try to vary the set of used classification strategies which provides us with a small improvement in the outcome of the classifier fusion.

7

Activity Spotting

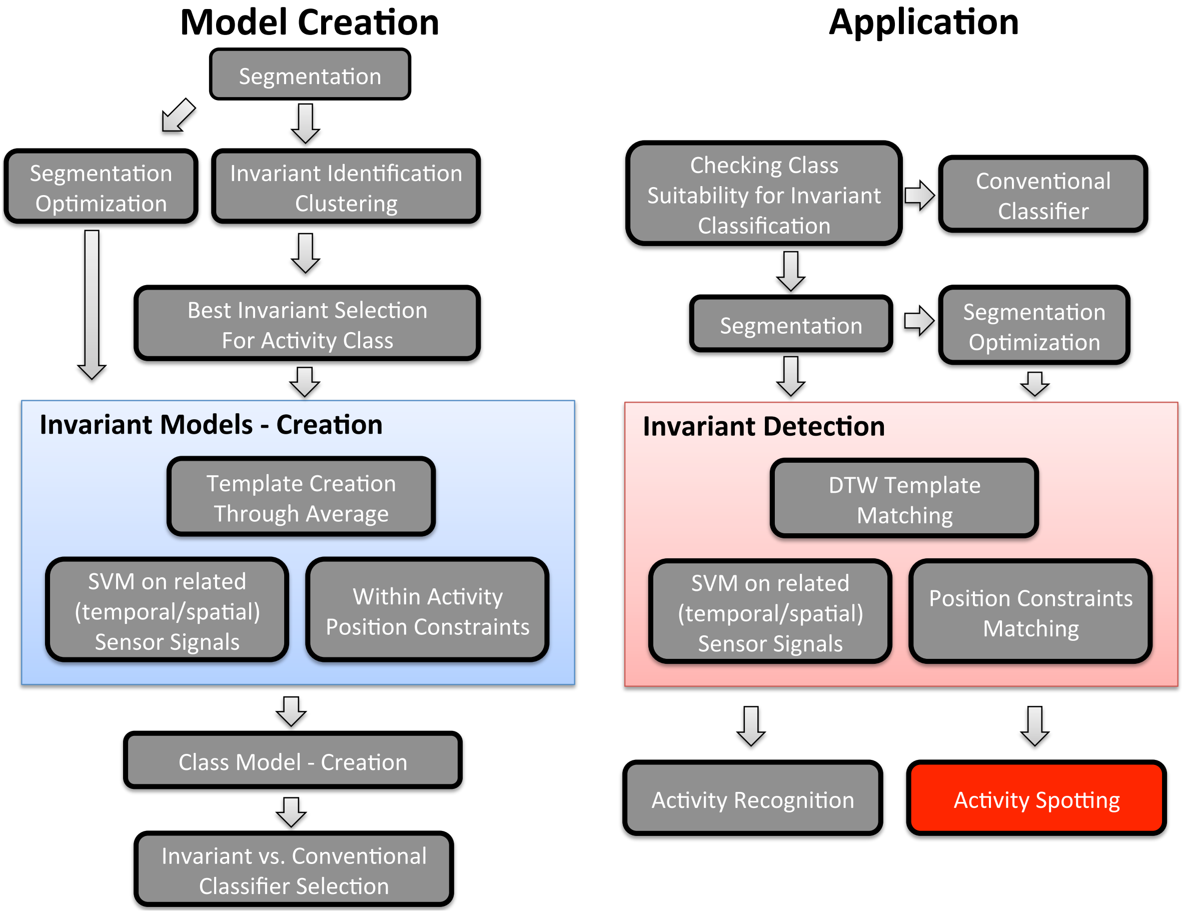

Model Creation
Segmentation
Segmentation Optimization
Invariant Identification Clustering
Best Invariant Selection For Activity Class
Invariant Models - Creation
Template Creation Through Average
SVM on related (temporal/spatial) Sensor Signals
Within Activity Position Constraints
Class Model - Creation
Invariant vs. Conventional Classifier Selection
Application
Checking Class Suitability for Invariant Classification
Conventional Classifier
Segmentation
Segmentation Optimization
Invariant Detection
DTW Template Matching
SVM on related (temporal/spatial) Sensor Signals
Position Constraints Matching
Activity Recognition
Activity Spotting

Figure 7.1: Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter.

7.1 Introduction

In this chapter we introduce the spotting and classification, where the activity class instances are detected in the continuous data stream. In the previous chapter, we introduced the *class instance model* which enables us to combine the detected class segments which we get from invariant models, to a final classified activity instance. However, for this final activity recognition result, we still used the given activity instance borders, which we got from the experiment recording. These borders are not known in an online activity spotting algorithm. Therefore we have to develop a method to estimate the instance borders of class activities from the found class segments. We can use the information of the position of the invariant members in their connected class instances to estimate these instance borders for every found class segments. When we combine the results of several invariants, we can improve the overall result by comparing the generated instance borders and finally stick to the borders which were generated by several invariants. The details for this approach, which enables our algorithm for online classification, follows next.

7.2 Evaluation

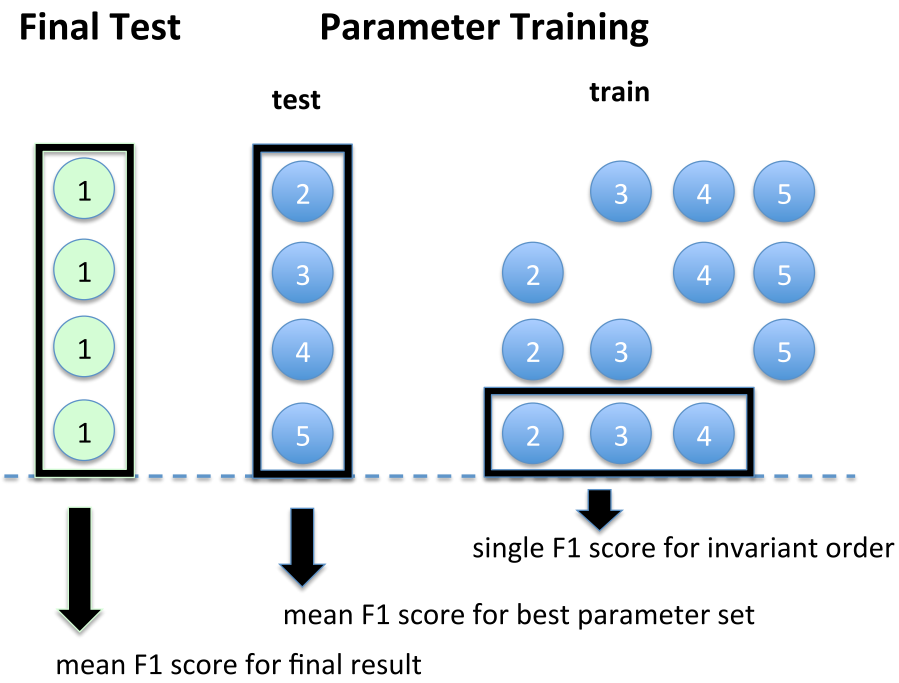

Final Test
Parameter Training
test
train
1
1
1
1
2
3
4
5
3
4
5
2
4
5
2
3
5
2
3
4
single F1 score for invariant order
mean F1 score for best parameter set
mean F1 score for final result

Figure 7.2: Two cross-validations for the parameter training. Each circle represents a subject of the evaluation experiments.

As explained in chapter 6, we use two connected cross-validations in our classification algorithm. The first cross-validation is needed to train the different parameters and the second to get the final results, depicted in Figure 7.2. We also see in Figure 7.2 three derived values from different sets of the cross-validation: the single F1-score for invariant order is generated by applying the trained invariant models on their training data. The obtained F1-score is then used to order the invariants for their fusion. It is important to use better invariant models first so that the parameters for used and needed invariants stay small. The second value in Figure 7.2 is the mean F1 score for the best parameter set. Here we calculate the mean of all cross-validation results within the same parameter set. The best mean F1-score shows us the best parameter set for the final classification. The third value, mean F1 score for final result, is a new way to calculate the final result. In the former chapter, we calculated the final result after the training of the parameters by training the invariant models with all training data of the parameter training cross-validation. We apply this strategy in the activity spotting too. But additionally, we also use all trained invariant models of the parameter training cross-validation and apply them on the test data. Then we calculate the mean over all F1-scores and thus get a second final classification result.

Next, we show all used parameters in the training phase and their variation range. One parameter mentioned in the former chapter does not appear here in the same form: We changed the name of the *tolerance threshold for the activity position model* parameter into *maxDiffFactor*, because we use the position information for the spotting and not for the classification anymore.

1. best invariant model configuration:
   This parameter defines the used invariant models for the classification. We use the *template model* and the *SVM model*.

2. basic or optimized segmentation strategy:
   This parameter defines if the normal segmentation or the optimized segmentation is used. We only use optimized segments with the *SVM model* in the *SVM with segment optimization* case.

3. sort strategy for the invariant models:
   This parameter defines how the invariants are sorted before the fusion of the single *invariants classification* results. We use the *template matching* for the *template model* and the *SVM* invariant detection strategy for the *SVM model*.

4. number of used invariants:
   This parameter defines the number of invariants which are fused for the final classification result. We use between 2 and 60 invariants per activity class.

5. number of needed invariants:
   This parameter defines the number of invariants which have to provide a positive classification result inside the borders of an activity class instance so that the instance is counted as a member of the activity class. We test values between 2 and 25.

6. maxDiffFactor:
   This parameter defines the factor for the multiplication of the standard deviation of the start/end position of the invariant in its class instance. This is needed to get a threshold for the maximum difference in the comparison of two instance start/end points. We test the values 1 and 2.

7. start - end - start and end:
   This parameter defines if only the start point or only the end point or both start and end points of the estimated instances have to match when we check if two estimated instances are equal. We test all three options.

To measure the classification quality we used in the former chapter precision, recall and F1-score. We also apply these measures for the final result after the activity spotting. In a first try, we counted all found instances with overlapping ground truth instances as true positives. When we got the F1-score greater than one, we realized that we have to refine the calculation of true positives. The problem is the fragmentation, i.e. if there exist more than one found instance overlapping the same ground truth instance. A further problem, called merge, is when one found instance overlaps more than one ground truth instance. To take these two factors into account, we assign each found instance to its ground truth instance. In the true positive calculation, we only check, if there is at least one found instance per ground truth instance. Thus we do not get false results due to fragmentation. In the merge case, we tested two approaches: we count all overlapped ground truth instances by a found instance as found. We call this *all covered ground truth*. The other option is to count only one ground truth instance per found instance. If a found instance covers more than one ground truth instance, we allocate the one with the greater overlap. We call this *only one ground truth*. Both methods are shown in the result section applied to the evaluation experiments.

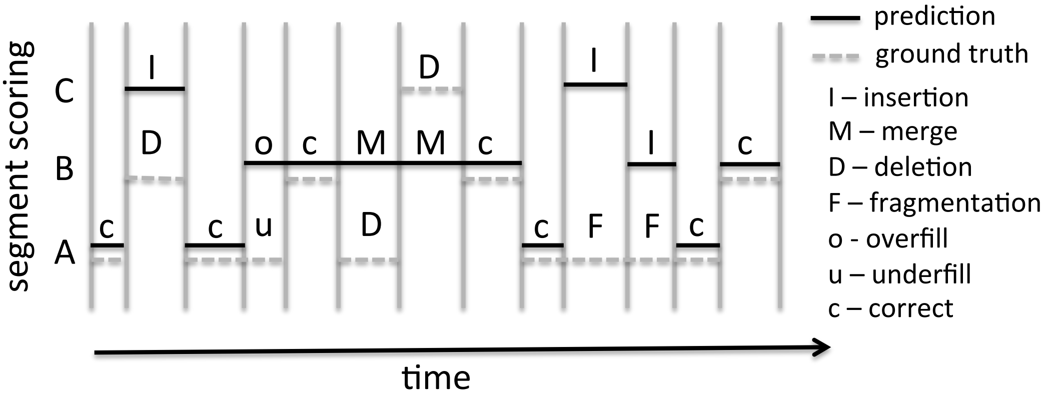


Figure 7.3: All different error categories of the SET measure shown in an example of three classes (A,B,C). This plot was taken from [129].

Activity spotting makes the classification process more difficult as we get problems like fragmentation and merge. Precision, recall, and F1-score cannot map these problems. Therefore we use additionally the segment error table (SET) measure form the publications [128, 129]. Here the signal data is segmented by each change in the prediction or the ground truth, i.e. every start or end point of a found or a ground truth instance is a new segmentation border. The segmentation is used for the prediction and the ground truth signal and then both are categorized. There are four segment categories for the prediction:

1. match:
   prediction and ground truth segment match in the activity class.

2. insertion:
   the prediction segment has a different class as the ground truth.

3. merge:
   if a prediction instance covers more than one ground truth instance of the same class, all prediction segments between the covered ground truth instances are counted as merge.

4. overfill:
   if a prediction instance starts before or ends after a ground truth instance of the same class, all prediction segments which exceed the ground truth instance are counted as overfill.

Moreover, there are four segment categories for the ground truth:

1. match:
   ground truth and prediction segment match in the activity class.

2. fragmentation:
   if a ground truth instance covers more than one prediction instance of the same class, all ground truth segments between the covered prediction instances are counted as fragmentation.

3. deletion:
   a ground truth segment has a different class as the prediction segment.

4. underfill:
   if a ground truth instance starts before or ends after a prediction instance of the same class, all ground truth segments which exceed the prediction instance are counted as underfill.

The *match* category returns for prediction and ground truth the same value. Figure 7.3 depicts the different error categories. Here the *match* category is marked with c (correct). For each category, we sum the lengths of the included segments and get then a value. From this value, we calculate the percentage in relation to the length of all ground truth activities of the appropriate class. These SET measure values help to make a statement about the classification result.

7.3 Spotting

In the spotting we want to detect the activity instance borders of the found activity segments. Figure 7.4 gives an overview of the spotting method. The starting points are the found activity segments which we have detected with the *template model* or the *SVM model*, depending on the before trained parameters for the *class instance model*. We exclude the *activity position constraints model* because we need this model for the spotting algorithm. The central idea of the spotting is the comparison of the estimated instance borders of each found segment, which is related to the *estimated segment position model*. The estimated instance borders are calculated using the segments' position information, which we get from the connected invariants. Each found segment is identified as class segment using an invariant model which relates to an invariant. The starting and the end position in percentage of all invariant member segments in their belonging activity class instance leads us to the position information. We calculate the mean and the standard deviation of the starting and the end position in percentage of all cluster member segments. With the mean value, we estimate the start and the end point of the instances for all found segment equally as in the *estimated segment position model* in Section 5.4.3. Next, we compare all such estimated instances, start versus start and end versus end points, and get a number of segments with equal estimated instance start/end points per found segment. The threshold, if two start/end points match, is the standard deviation of the invariant's position multiplied by the factor *maxDiffFactor*. The comparison e.g. of segment1 and segment2 works in the following way. We begin by calculating the start and end of

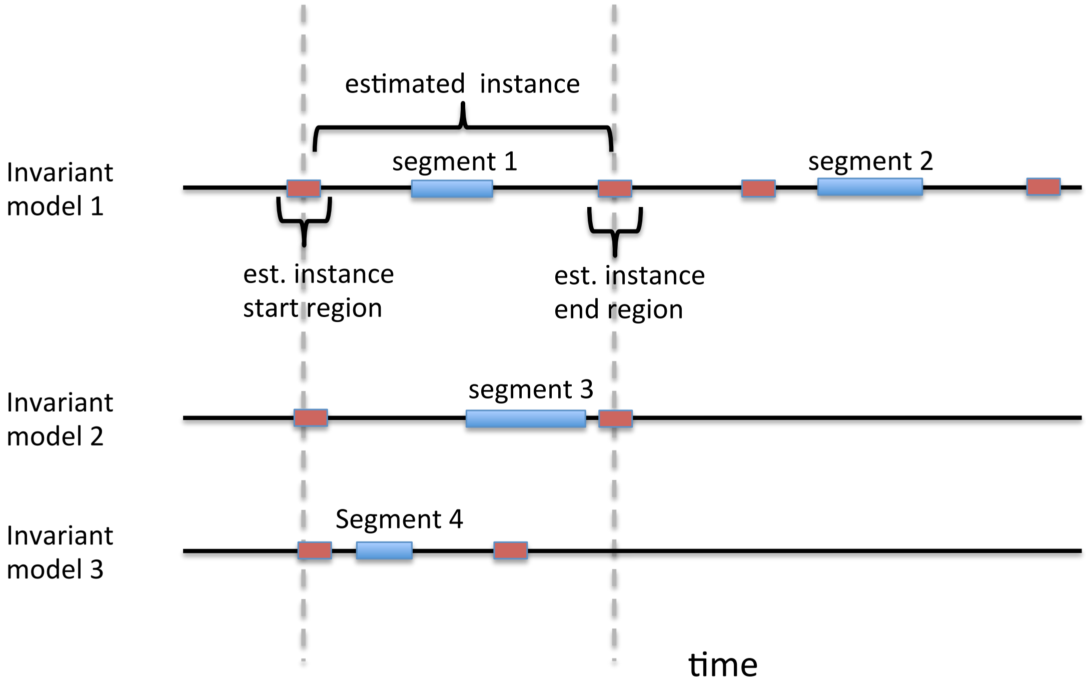


Figure 7.4: Overview of the activity spotting strategy.

the range of the estimated instance start point of segment1 ($estInstStartRegion1_{seg1}$, $estInstStartRegion2_{seg1}$). This is done by adding and subtracting the standard deviation of the invariant's position of segment1 multiplied by the factor *maxDiffFactor* from the estimated start position of segment1.

$$estInstStartRegion1_{seg1} = estInstStart_{seg1} - SD_{Start-seg1} * maxDiffFactor \tag{7.1}$$

$$estInstStartRegion2_{seg1} = estInstStart_{seg1} + SD_{Start-seg1} * maxDiffFactor \tag{7.2}$$

Next we subtract from the range start and end point of the estimated instance start point of segment1 the estimated instance start point of segment2. For both results we calculate the absolute value and then keep the smaller value, which we call $startDiff$.

$$\begin{aligned} startDiff = min(abs(estInstStartRegion1_{seg1} - estInstStart_{seg2}), \\ abs(estInstStartRegion2_{seg1} - estInstStart_{seg2})) \end{aligned} \tag{7.3}$$

To decide if both estimated instance start points are similar enough, we conduct the following comparison: if $startDiff$ is less or equal to the standard deviation of the invariant's position of segment2 multiplied by the factor *maxDiffFactor*, then the estimated instance start positions of segment1 and segment2 are counted as equal.

$$startDiff <= SD_{Start-seg2} * maxDiffFactor \tag{7.4}$$

The calculations for the estimated instance end position are analog. After comparing the estimated instances of all found segments, we get a list where the number of equal instance start or end or start and end point is given for each segment. In the parameter training of the *class instance model* we train the number of needed invariants and if the start, the end or both have to match. We keep then all found segments and their estimated instances which comply to these parameters. The last step is to fuse all estimated instances which overlap, to a new instance. In this way, we diminish the number of found instances significantly.

To address the merge and the overlap of found instances, we developed a *length adoption* strategy for the found estimated instances after the fusion. If the instance length exceeds a certain length, then we divide the instance into an appropriate number of instances with a smaller length. We also adopt the length of the instance or instances to a certain degree. First, we calculated the mean length of all found instances (*meanLength*) before the fusion. Then we test for each found instance, how often this mean length fits inside, by dividing the length of the instance *instLength* by the *meanLength*.

$$divisons = floor(instLength/meanLength) \tag{7.5}$$

Then we calculate the length of the division parts.

$$divLength = floor(instLength/divisons); \tag{7.6}$$

For each division part $divPart_i$ we calculate the center

$$newInstCenter = instStart + ((divPart_i - 1) * divLength) + floor(divLength/2); \tag{7.7}$$

The new instance start point is the center minus the half *meanLength*, the end point the center plus the half *meanLength*.

$$newInstStart = newInstCenter - floor(meanLength/2) \tag{7.8}$$

$$newInstEnd = newInstCenter + floor(meanLength/2)]; \tag{7.9}$$

We tested the estimation of the raw instances and the instances with adopted length, and show the analysis in the result section.

7.4 Results

First, we want to analyze the different outcomes by applying the two calculation strategies for precision, recall and F1 score, the *length adoption* strategy and the variation of the training set for the activity spotting with our *invariants classification*. In Table 7.1 we show all variations with their precision, recall, F1-score and the SET measure on the two evaluation experiments. The SET measure values in the table are given as a percentage of the length of all ground truth activities of the appropriate class. As explained in the evaluation section, we get different results for precision, recall and F1 dependent on the *all covered ground truth* or the *only one covered ground truth* method. The difference between both methods is that we either count all covered ground truth instances (*all covered ground truth*) or only the mostly covered ground truth instance (*only one covered ground truth*) of a prediction instance as true positives. This also explains the different results: in the *bicycle repair* experiment the *all covered ground truth* method returns better F1-score results because merges occur as we can see in the SET values. This means that there are found instances which cover more than one ground truth instance, which benefits the *all covered ground truth*. In the *car workshop* experiment there are no merges hence there are no differences between the *all covered ground truth* and the *only one covered ground truth* method. In the final activity spotting result, we chose the *only one covered ground truth* because it produces results without rewarding merges.

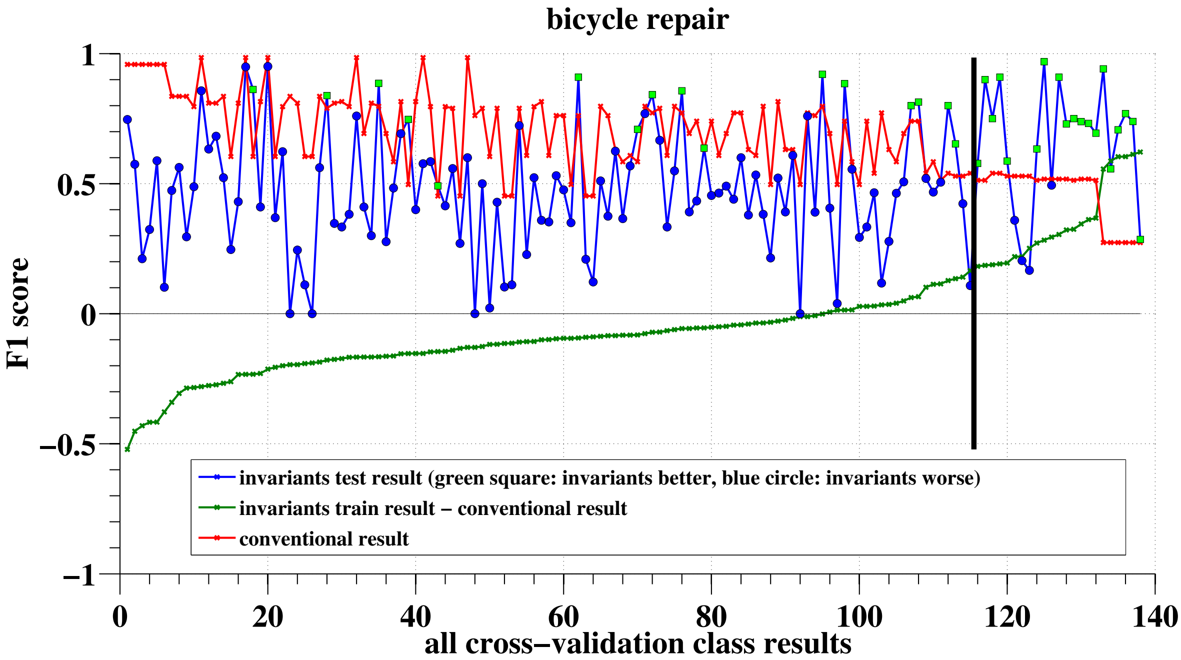


Figure 7.5: Bicycle repair experiment

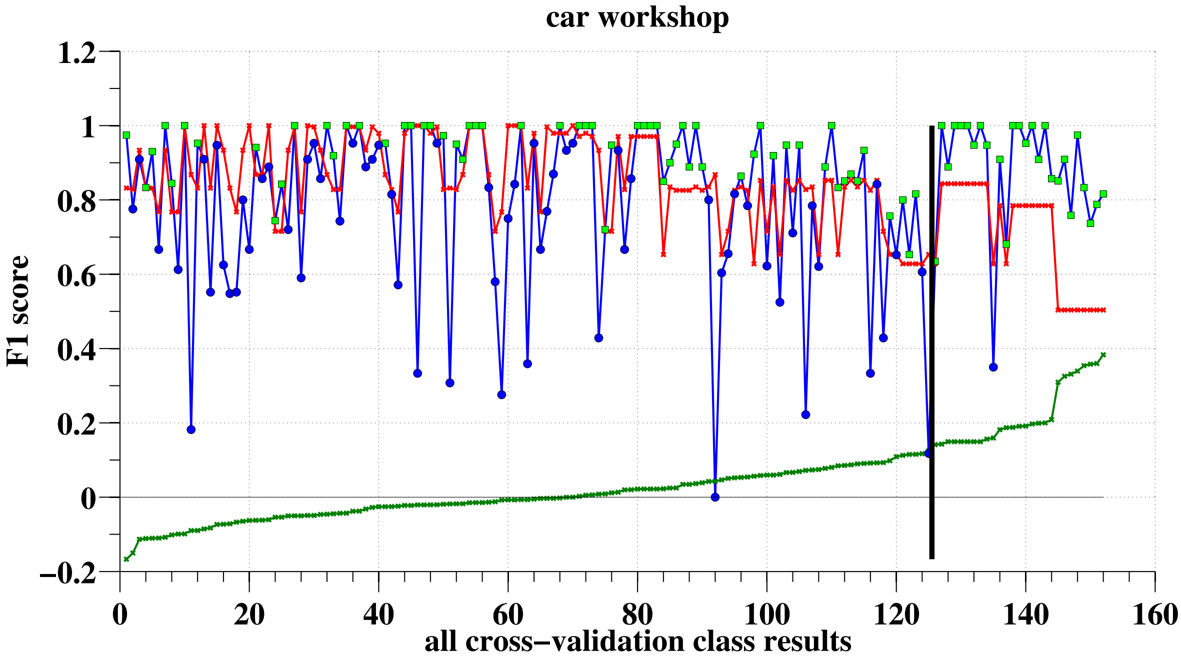
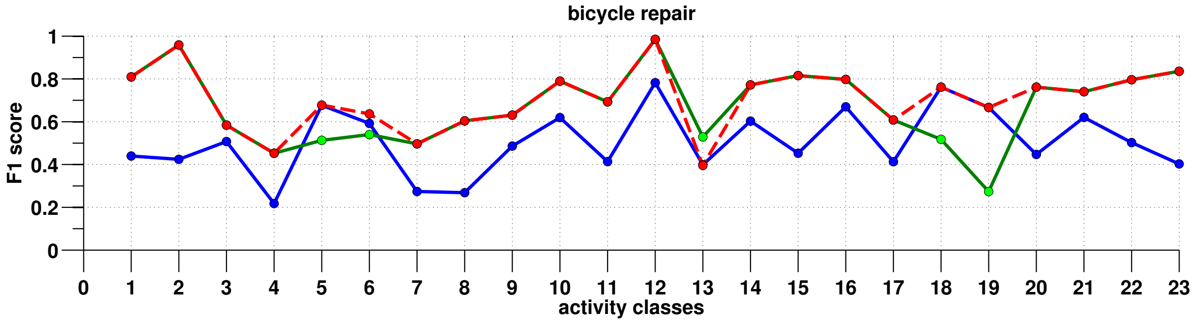
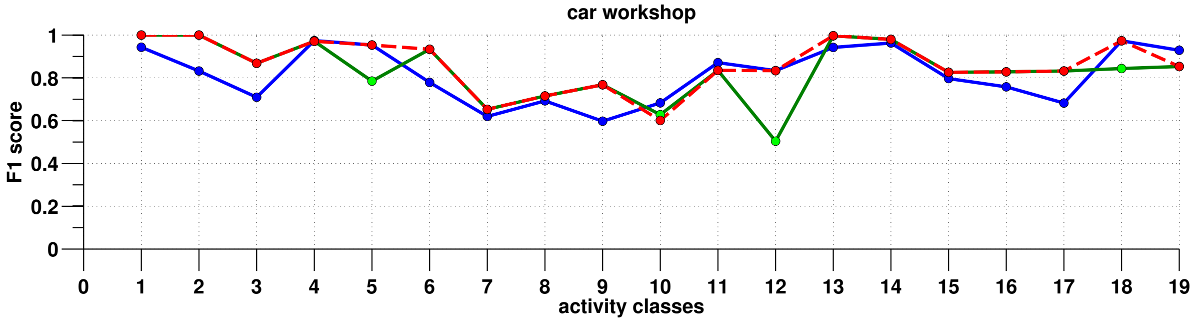


Figure 7.6: Car workshop experiment



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
invariants classifier	0.44	0.42	0.51	0.22	0.68	0.59	0.27	0.27	0.49	0.62	0.41	0.78	0.40	0.60	0.45	0.67	0.41	0.76	0.67	0.45	0.62	0.50	0.40
conventional classifier	0.81	0.96	0.58	0.45	0.51	0.54	0.50	0.60	0.63	0.79	0.69	0.98	0.53	0.77	0.82	0.80	0.61	0.52	0.27	0.76	0.74	0.80	0.84
fusion	0.81	0.96	0.58	0.45	0.68	0.64	0.50	0.60	0.63	0.79	0.69	0.98	0.40	0.77	0.82	0.80	0.61	0.76	0.67	0.76	0.74	0.80	0.84

Figure 7.7: Mean F1 score results over all subjects per class of the *invariants classifier* (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
invariants classifier	0.94	0.83	0.71	0.97	0.95	0.78	0.62	0.69	0.60	0.68	0.87	0.83	0.94	0.96	0.80	0.76	0.68	0.97	0.93
conventional classifier	1.00	1.00	0.87	0.97	0.78	0.93	0.65	0.72	0.77	0.63	0.83	0.50	1.00	0.98	0.83	0.83	0.83	0.84	0.85
fusion	1.00	1.00	0.87	0.97	0.95	0.93	0.65	0.72	0.77	0.60	0.83	0.83	1.00	0.98	0.83	0.83	0.83	0.97	0.85

Figure 7.8: Mean F1 score results over all subjects per class of the *invariants classifier* (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment.

Table 7.1: Activity spotting variation results: mean precision, mean recall, mean F1-score and mean SET measure.

experiment	variation	∅pre	∅rec	∅F1	correct	del	frag	underfill	insert	merge	overfill
bicycle repair	inter-all all cov gt	0.62	0.69	0.60	62.8	33.9	0.1	3.0	0.0	15.7	40.5
	inter-single all cov gt	0.63	0.60	0.55	53.8	41.3	0.1	3.6	0.0	9.7	30.8
	inter-all 1 cov gt	0.59	0.54	0.51	62.8	33.9	0.1	3.0	0.0	15.7	40.5
	inter-single 1 cov gt	0.60	0.50	0.49	53.8	41.3	0.1	3.6	0.0	9.7	30.8
	inter-all all cov gt new inst	0.52	0.68	0.52	55.5	34.0	2.9	7.4	16.4	1.8	24.3
	inter-single all cov gt new inst	0.55	0.60	0.49	47.3	41.4	2.3	7.8	8.9	1.3	18.5
	inter-all 1 cov gt new inst	0.52	0.68	0.51	55.5	34.0	2.9	7.4	16.4	1.8	24.3
	inter-single 1 cov gt new inst	0.55	0.60	0.49	47.3	41.4	2.3	7.8	8.9	1.3	18.5
car workshop	inter-all all cov gt	0.84	0.85	0.82	82.1	15.1	0.0	1.6	0.0	0.0	55.8
	inter-single all cov gt	0.85	0.81	0.80	78.0	19.6	0.0	1.7	0.0	0.0	48.3
	inter-all 1 cov gt	0.84	0.85	0.82	82.1	15.1	0.0	1.6	0.0	0.0	55.8
	inter-single 1 cov gt	0.85	0.81	0.80	78.0	19.6	0.0	1.7	0.0	0.0	48.3
	inter-all all cov gt new inst	0.80	0.85	0.79	72.4	15.1	2.4	9.0	5.7	0.0	22.8
	inter-single all cov gt new inst	0.82	0.81	0.78	68.9	19.6	1.6	9.2	3.8	0.0	18.5
	inter-all 1 cov gt new inst	0.80	0.85	0.79	72.4	15.1	2.4	9.0	5.7	0.0	22.8
	inter-single 1 cov gt new inst	0.82	0.81	0.78	68.9	19.6	1.6	9.2	3.8	0.0	18.5

Table 7.2: Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both applied on the bicycle and the car workshop experiment.

		∅pre	∅rec	∅F1
bicycle repair	invariants classifier	0.59	0.54	0.51
	conventional classifier	0.57	0.91	0.67
	fusion	0.64	0.87	0.71
car workshop	invariants classifier	0.84	0.85	0.82
	conventional classifier	0.82	0.85	0.83
	fusion	0.86	0.88	0.86

The spotting provides us with predicted class instances. Table 7.1 shows that overfill is the dominant error in the classification and also merge is a problem in the *bicycle repair* experiment. To overcome this, we tried to introduce the *length adoption* strategy, where we use the mean length of all found instances before the fusion to change the length of the found instances after the fusion. We have explained the strategy in detail in the previous spotting section. Compared to normally found instances, the *length adoption* provides no advancement in terms of the F1-score. This is due to a drop in the precision. The *length adoption* improves the overfill but at the same time the values for underfill and insert rise and the correct classified parts decrease. This led to the decision to directly use the prediction instances of the spotting in the final classification result without any *length adoption*. The last variation is between the *inter-all* and the *inter-single* strategy. After the parameter training, the *inter-all* strategy uses all training instances to generate the invariant models, while the *inter-single* strategy directly uses the invariant models from the parameter training, which were built with subsets of the training data. We see that the *inter-all* strategy is always better in recall and F1-score. Only in precision, the *inter-single* strategy is better. Thus we opted for the *inter-all* strategy in the final result.

Next, we show the final results with the *only one covered ground truth* strategy, the

Table 7.3: Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment.

classify strategy	bicy. inv.	bicy. fusion	car inv.	car fusion
template match	12	0	2	11
SVM	50	48	29	30
SVM seg. opt.	38	52	69	59

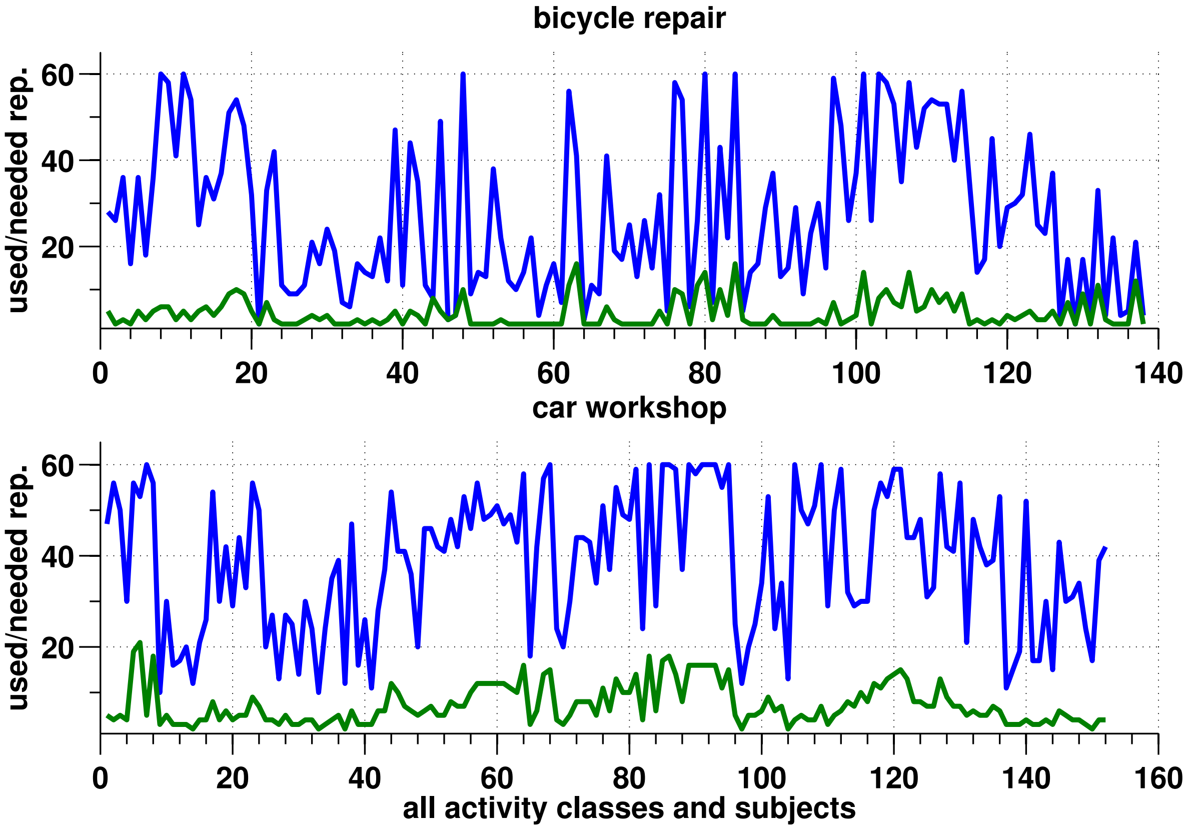


Figure 7.9: Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments.

raw spotting instances, and the *inter-all* strategy. In Tabel 7.2 we present the mean precision, recall, and F1-score over all class activities of the invariant classifier, the conventional classifier and the fusion of both on the two evaluation experiments. On both experiments, the *invariants classifier* provides a worse recall and F1-score than the conventional classifiers, but the precision is better and the fusion result indicates an improvement. So the invariant classifier is advantageous for certain classes and enhances classification results in combination with conventional classifiers. In Figures 7.5 and 7.6 we see the comparison of the difference between the F1-scores of the *invariants classifier* on training data and the conventional classifier, the F1-score of the conventional classifier and the F1-score of the *invariants classifier* on test data. Green squares on the *invariants classification* test results indicate that the *invariants classification* is better than the conventional classification, blue circles vice versa. The black vertical line indicates the fusion border: everything left is classified with the conventional classifier, everything right is classified with the *invariants classifier*. As explained in the last chapter we use the *invariants classification* results on training data to decide whether we use the *invariants classification* in the fusion

Table 7.4: Mean SET measure of invariants classifier results of the bicycle repair and car workshop experiment. SET measure values are given as a percentage of the length of all ground truth activities of the appropriate class. We calculated the SET values on: all invariants results (all); all invariants results used in the fusion (allfusion); all invariants results used in the fusion which return a better result than the conventional classifier (all pos fusion).

experiment	invariants set	correct	del	frag	underfill	insert	merge	overfill
bicycle repair	all	62.8	33.9	0.1	3.0	0.0	15.7	40.5
	all fusion	69.8	27.7	0.0	2.3	0.0	11.3	55.3
	all pos fusion	77.4	19.9	0.0	2.4	0.0	10.1	60.0
car workshop	all	82.1	15.1	0.0	1.6	0.0	0.0	55.8
	all fusion	87.3	10.1	0.0	2.0	0.0	0.0	50.1
	all pos fusion	89.4	7.9	0.0	2.1	0.0	0.0	50.9

or not. In Figures 7.7 and 7.8 we depict the results of the *invariants classifier* (blue), the conventional classifier (green) and the fusion of both (red). The fusion provides in the *bicycle repair* experiment better results compared to the conventional classifier in the classes (5) red screw (open), (6) red screw (close) (18) seat (up) and (19) seat (down). Class (13) disassembling front wheel, has a worse outcome in the fusion case compared to the conventional classifier. In the *car workshop* experiment classes (5) close trunk, (12) close two doors and (18) open spare wheel box, have a better classification result with the fusion in contrast to the conventional classifier, while class (10) close right door, drops slightly. Both experiments show the best overall classification results using the fusion of the invariants and the conventional classifiers. Thus the *invariants classification* algorithm provides a benefit for the classification of certain activity classes.

In Table 7.3 the distribution of the three used invariant models in the *invariants classification* and the fusion of invariants and conventional classifier can be seen. The order of importance of the classification models are *SVM with segment optimization*, *SVM* and *template matching*. The number of used and needed invariants is shown in Figure 7.9. When we compare this plot to Figure 6.13, there is a tendency that the number of used and needed invariants is a little bit lower in the spotting case. The last Table 7.4 compares the SET measures of three sets of *invariants classification* results. The first set are all cross-validation results on test data (the blue line in Figures 7.5 and 7.6), the second are all results which were selected in the fusion (all data points of the blue line on the right side of the black vertical line in Figures 7.5 and 7.6), and the last set are all results which were selected in the fusion and which had also a better outcome than the conventional classifier (all data points of the blue line with green squares on the right side of the black vertical line in Figures 7.5 and 7.6). Again the SET measure values in the table are given as a percentage of the length of all ground truth activities of the appropriate class. Here we see that from set one to set three in both experiments the percentage of correct parts increase while all other measure decrease most of the time. Only the overfill increases from set one to set three in the *bicycle repair* experiment, and only the underfill increases a bit in the *car workshop* experiment. However, the overall result of the SET measures is always better. In conclusion, this shows, that the invariant models which provide a better classification quality are selected in the fusion of invariants and conventional classifier.

7.5 Summary

In this chapter, we presented the last step of the *invariants classification*: identifying the borders of activity class instances, using the classified activity class segments, called *spotting*. For this purpose, we use the information from the invariants which gives us the position of the invariant segment members in their instances. This enables us to estimate the instance start and end point of detected segments when we apply an invariant model. We construct the instance estimation for each found segment and then check how many estimated instances have the same start and end points including a certain divergence. Then we train a threshold so that only the instances with a greater number than this threshold of equal start and end points are accepted as found. Next, we fuse all such overlapping found instances. To evaluate these classification results, we again calculate precision, recall, and F1-score, but also add the SET measure. This measure provides more detailed information, e.g. if the predicted instances overfill or underfill the ground truth instances, or if there is a merge or a fragmentation of predicted instances versus ground truth instances. The chapter ends with the presentation of the final *invariants classification* results and the classification results of the fusion of invariants and conventional classifiers on the *bicycle repair* and *car workshop experiment*. We show that the *invariants classification* for certain classes provides better classification results than the conventional classifiers and that a fusion of both classifiers improves the final activity classification outcome.

8

Conclusion and Outlook

A key issue in the automatic recognition of human activities with body-worn sensors is the variability of human activities and the huge range of possibilities for executing even fairly simple actions. This thesis investigates a novel human activity recognition algorithm for wearable sensor systems, called *invariants classification*, which addresses this problem. The core idea is that often even highly variable actions include short more or less invariant parts which are due to hard physical constraints. The aim is to develop an algorithm that can identify such invariant activity parts and use them to improve the classification of the respective activities. The algorithm is meant to be combined with existing classification approaches in an ensemble like fashion, being applied only to the classes for which appropriate invariant sub-activities can be found and leaving the other classes to be handled by classical methods.

The first step of the *invariants classification* is to split the continuous sensor data stream into meaningful data packets in the time domain. By understanding our dataset as time series we can also see the separation as splitting the data in shorter time series which keeps the information we want to analyze and also lowers the dimensionality for the later processing by classification algorithms. At first sight, the perfect segmentation entity in our case covers a complete activity. The problem here is the variability of human activities which would result in a big variance in the segments of one activity class. The solution presented in our algorithm is to use a segmentation strategy where the data is segregated into sub-activities. These sub-activities are defined by cutting the continuous data where its central difference quotient has a zero crossing. Thus we split complex activities in basic motions. Furthermore, we try to improve our segmentation strategy by fusing adjacent segments to a new segment. For this purpose, we use information from the next step in our processing chain, the *invariant identification*.

The *invariant identification* searches for groups of similar sub-activities in the entire activity. For this purpose, we use an agglomerative hierarchical clustering with average linkage. The clusters are constructed on the segments of all training activity instances of a class and a signal channel. A specialty in our clustering approach is, that we take all possible clusters during the bottom-up construction of the cluster tree. Then we sort the clusters using three features derived from the clusters. Finally, we remove the clusters with overlapping members stepwise along the order until only clusters without any identical segment members exist. We call these clusters *invariants*. Additionally, we calculate the measure called *cluster precision* which pairwise compares the invariants of the activity classes. This value helps us to know at an early stage which classes tend to be easy classified and which are more prone to be mixed up with other classes in the final classification, as it can be graphically shown. It also depicts the importance of the signal channels in the context of activity class confusion.

In the last step, we explain the *invariant detection*, the *classification* and the classification with spotting which we call *activity spotting*. In the *invariant detection*, we want to find class segments in new sensor data. For this purpose, we construct three invariant models: *template model*, *SVM model*, and *activity position constraints model*. The *template model* creates a template of the invariant which is the center of the invariant. The *SVM model* builds an SVM with a feature set from other signal channels. The key point here is that the segmentation of the invariant's signal channel is used on the other signal channels. The *activity position constraints model* determines the constraints which are related to the invariant's position in its activity instance. This is calculated by taking the mean of the positions of the invariant's member segments. The variance of the cluster member position defines the position variation threshold. Next, we apply these three models to new sensor data to detect class segments. The *template model* implements a template matching with DTW similarity measure. The trained SVM is then applied only on the positive result of the template matching. Finally, the position constraints matching checks the results of the template matching and the SVM, and also on the raw data, if the position constraints are fulfilled. Furthermore, we show the *invariant detection with the segment optimization*. Here we fuse adjacent segments and apply either the template matching or the SVM classifier on these new segments. As we get several invariants for each class, we can combine their models to a final classification model which we call *class instance model*. Here we want to classify the entire activity instance, not only the sub-activities. We take the activity instance borders as given from the experiment recordings and estimate instance borders of the null class. After training several parameters, like number of used invariants and number of needed invariants, we get a final classification result of entire activities. In the *activity spotting*, we estimate the class instance borders for the final classification. For this purpose, we estimate for each found class segment of the set of invariants the instance borders. This is done using the *activity position constraints model*. Then we check for each estimated instance how many instances with the same position are found. We keep then all instances which have at least a certain number of instances with equal positions. We fuse all overlapping instance and get the final classification result. A further step is to fuse the conventional classifiers with the *invariants classifier*. We decide which classifier to utilize by comparing F1-scores of the conventional classifier and the invariants classifier on training data.

8.1 Summery of achievements

This thesis aims to develop a new human activity recognition algorithm for wearable sensors. Algorithms for this purpose are structured in several processing steps. The first step usually separates the continuous sensor signal to segments. Here our algorithm achieves a sensible segmentation strategy of the sensor data with relation to the underlying activities. We introduce segment borders where the central difference quotient of the sensor data exhibits a zero crossing. In our experiments, we use trajectories derived from IMUs and the central difference quotient of one trajectory axis is equal to the velocity. If the velocity has a zero crossing, then this is a change in the direction on the axis of the specific trajectory. The trajectories are connected to body limbs of the upper body. If, e.g. the upper arm has a direction change in the x-axis this seems to be a good incision for a motion describing the smallest unit of an activity. Therefore this segmentation strategy results in a set of smallest sub-activities which is exactly the resolution we need for further processing.

The creation of an activity recognition system usually requires a training phase where the activities of interest are learned. For this purpose, the standard procedure is to record training data of the different activities with several sensors. This recording consists of the continuous sensor data stream where the start and end points of the activities are marked. These labels are usually set manually which means that imprecise label borders can occur. In our algorithm, we work only with automatically created segments. That means that we are independent of the manual labels and their possible inaccuracy.

The central idea of our algorithm is to identify a sub-activity which is unique for the complete activity. To reach this, we cluster all trained segments of activity instances to detect this sub-activity. The results here are the clusters, which are groups of such sub-activities. We call these clusters *invariants*. After the selection of the best *invariants* we have a set of sub-activities which can be used to build three different models for the classification and the spotting of the activity classes. The final classification results show us that the idea to use certain sub-activities for the classification of activity instance makes sense.

Additionally, the clustering provides us with the measure called *cluster precison*. This is the pairwise similarity of the cluster centers of the invariants. We can plot this value for each sensor signal and each class activity and have then a good overview of the pairwise confusion of the different classes in all sensor channels. When we sum up all *cluster precision* values for each class, we can show which classes are easily distinguishable and which lead to confusions. Summing up all *cluster precision* values of each sensor channel reveals the sensors which exhibit information to separate and recognize the complete activity set. Receiving such insight, already before the classification step, can bring advantages for the assessment of activity sets and sensor configurations.

One important requirement which has to be fulfilled for our new activity recognition algorithm is that it has to process several sensor systems. To deploy a net of sensors is nowadays technically possible. This makes sense because the more installed sensors exist, the more information can be processed which helps to get more information about the human being in its environment. However, a high number of input sensor signals leads us to the problem of the curse of dimensionality. This means that we cannot use all sensor information, because it makes the data processing too complex. Instead, we should only choose signals which contain the necessary information to identify the different activity classes. The solution here is automatically achieved with our algorithm. In the clustering, we search for invariants in all sensor signals. These invariants are then pre-filtered in the *best invariant selection*. When we train the parameters of the *class instance model* for the final classification, we select the best invariants over all signal channels. This means that in this step we condense the multi-dimensional data stream of sensor signal into a set of invariant models on certain sensors which solves the problem of selecting the right sensor set. Furthermore, the segmentation for each sensor signal is unique, which means that the start points of invariant sub-activities vary. To compare sub-activities, we use dynamic time warping which allows a comparison of signals with different number of data points. This enables us to process sub-activities with different length. In conclusion, these three points - motifs on several data channels, motifs don't have to start and end at the same time on different data channels, motif length is not fixed - make our strategy to a multi-dimensional asynchronous motif detection algorithm with variable motif length.

Finally, we wanted to develop an activity recognition algorithm for human activities with bodyworn sensors with a focus on the challenge of the variability of human motions. The kernel concept is to identify invariant subparts in an activity class. We found a way to build a classification process around this central idea with the introduction of *invariants*. We saw in our experimental evaluation of our new algorithm that not all activity classes are suitable for the *invariants classification*. However, when we combine our classifier with conventional classifiers, we get overall a better classification result. In the *bicycle repair* and in the *car workshop* experiment we got an improvement of the classification in 4 of 23 and 3 of 19 classes respectively, in same cases by a large margin (best case is from 27% to 67% in the first and from 50% to 83% in the second). In each dataset there is only one class for which we make the recognition worse and in both cases, it is one with poor results and a relatively small decrease (from 53% to 40% in the first and from 63% to 60% in the second). The results are achieved for an user-independent case.

8.2 Outlook

After introducing our new activity recognition algorithm, we want to present several ideas in this last section for the future work on the basis of this thesis.

In the activity spotting, we fuse the estimated instances just by combining overlapping instances. It would be interesting to see if this fusing could be improved. If two ground truth activities of the same class are close to each other, we get the problem of merging them in the classification. Maybe fusing two found estimated instances should be done only when the overlap exceeds a certain percentage of the instance length. Although the *length adoption* did not bring an improvement in the final outcome of our classification algorithm, a variation of this approach could address the problem of instance merges.

Another extension of our classifier is to prepare it for online usage. So far the *invariants classification* can be applied as online algorithm. In the classifier model construction, i.e. segmentation, invariant detection, and invariant models building, nothing has to be changed. In the application of the classifier model, we have to classify each activity class parallel on the incoming data, whereas in this thesis we analyzed our experiment data sequentially. There has to be also a kind of buffering for the found segments and the estimated instances in the data stream so that the activity spotting can generate the final results.

We mentioned in our considerations, regarding the variability of human activity, the example of closing a door with hands versus closing it with legs. In our algorithm it is possible, if there are enough training sets, that we identify *invariants* for both activity variations and also classify both variations. It would be good to extend and modify our algorithm in a way that we could identify that there are two sets of training data with their own *invariants*. Splitting this automatically in two new activity classes would surely improve the classification. Furthermore, most activity recognition research tries to identify only the activity but not the quality of its conduction. Maybe the *invariants* could also identify differences in the activity quality, e.g. the right movement to hit a tennis ball with a bracket.

The previous ideas make it necessary to collect datasets which contain these problem statements. There has to be a sensor set which surveys different body parts. Then we have to develop a scenario which features activities which can be conducted with different body parts. A comparison of activities executed by left- and right-handers could be especially interesting because this is probably the most common application of the variation of body part usage. Another scenario could address the activity quality, e.g. playing tennis by a professional, semi-professional and beginner. Here we can analyze if there are different *invariants* for the same activity among the different player types. Another aspect in the design of new datasets is the sensor usage. In our evaluation experiments, only IMUs were analyzed by our algorithm. We suppose that our algorithm works on new experiments with wearable IMUs. However, can we transfer this algorithm for activity recognition to other wearable sensor systems or in general to other sensor types?

List of Figures

2.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. 15
2.2 Segmentation example of the car workshop experiment. The lower plot is the raw signal (x-axis of a trajectory signal), the upper plot is its central difference quotient. The two colored regions of the lower plot represent activity class instances. The red dots are the segment borders (lower plot) or the zero crossing (upper plot). 17
2.3 MTx sensor from Xsense 21
2.4 MTx-sensor: earth fixed versus sensor fixed coordinate system. 22
2.5 The position of the MTx sensors and the trajectory locations. 23
2.6 Stick figure constructed of the sums of the orientation matrices T multiplied with the body part length vectors l. 23
2.7 Heading compensation. 24
2.8 Joint angles. 25
2.9 Error of arc cosine: results of $x - acos(cos(x))$, $x = 0 : \pi/1e08 : \pi$. 26
2.10 Drink activity from the drink and work experiment. 29
2.11 Activity from the bicycle repair experiment. 31
2.12 Activities from the car workshop experiment. 33

3.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter. 37
3.2 Mean length of instances in data points of the activity classes of all evaluation experiments. red: drink and work; blue: bicycle repair; green: car workshop. 41
3.3 Mean number of segments per class instance. 42
3.4 Mean number of segments per class instance for each signal channel. 43
3.5 dark/light blue: mean/SD of class instance length; turquoise/yellow: mean/SD of instance - segment difference with *basic-segmentation*; orange/red: mean/SD of instance - segment difference with *low velocity segment fusion*. 45

4.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. The red marked step is explained in this chapter. 49
4.2 The cluster tree (dendrogram) and all possible clusters, represented by the red ellipses. 52
4.3 Visual representation of cluster precision results for all sensor signals (trajectory/angle of shoulder, elbow, wrist or hand tip) over all classes of the three experiments; Bright areas indicate high inter-class similarities, whereas darker areas indicate low inter-class similarities. 56
4.4 Cluster precision: best channels of the three experiments. High values of the *cluster precision* sum indicate low inter-class similarities. 57
4.5 Cluster precision: best classes of the three experiments. High values of the *cluster precision* sum indicate low inter-class similarities. 58

5.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter. 63
5.2 Estimated position calculation 70
5.3 Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work experiment. 72
5.4 Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work (only one vessel class) experiment. 72
5.5 Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the drink and work (only one vessel class, only sip gesture) experiment. 73
5.6 Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the bicycle experiment. 73
5.7 Detected segments after applying invariant models (blue = true positive segments; red = false positive segments) to the car workshop experiment. . 74

6.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter. 77
6.2 Two connected cross-validations for the parameter training (blue) and the final classification result (green: test data, blue: train data). Each circle represents the data of one subject. 79
6.3 All classification results from our algorithm: red dots. Results from [2]: blue dots. 83
6.4 Classification results with all invariant detection strategies. 83
6.5 Bicycle repair. All classification results from our algorithm: red dots. Location trajectory-based spotting using polynomial matching cost features result from [99]: blue squares. 84
6.6 Bicycle repair. Classification results with all invariant detection strategies. 85
6.7 Car workshop. All classification results from our algorithm: red dots. Results from [142]: blue line. 86
6.8 Car workshop. Classification results with all invariant detection strategies. 87
6.9 Bicycle repair experiment 92
6.10 Car workshop experiment 92

6.11 Mean F1 score results over all subjects per class of the invariants classifier (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment. 94
6.12 Mean F1 score results over all subjects per class of the invariants classifier (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment. 94
6.13 Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments. 95
6.14 Bicycle repair with *invariant detection strategy set variation*. 97
6.15 Car workshop experiment with *invariant detection strategy set variation*. . 97
6.16 Mean F1 score results over all subjects per class of the invariants classifier with *classification model set variation* (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment. 99
6.17 Mean F1 score results over all subjects per class of the invariants classifier with *classification model set variation* (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment. 99
6.18 Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments (*classification model set variation*). . . . 101

7.1 Overview of *invariants classification* algorithm. The left side shows the creation of the classifier model. The right side shows the application of the classifier model. All red marked steps are explained in this chapter. 103
7.2 Two cross-validations for the parameter training. Each circle represents a subject of the evaluation experiments. 104
7.3 All different error categories of the SET measure shown in an example of three classes (A,B,C). This plot was taken from [129]. 106
7.4 Overview of the activity spotting strategy. 108
7.5 Bicycle repair experiment 110
7.6 Car workshop experiment 110
7.7 Mean F1 score results over all subjects per class of the *invariants classifier* (blue), the conventional classifier (green), and the fusion of both (red) applied on the bicycle repair experiment. 111
7.8 Mean F1 score results over all subjects per class of the *invariants classifier* (blue), the conventional classifier (green), and the fusion of both (red) applied on the car workshop experiment. 111
7.9 Number of used (blue line) and needed invariants (green line) for all classes and subjects in both experiments. 113

List of Tables

2.1 MTx sensor data performance specification. 21
2.2 Basis vectors for elbow and joint wrist angle calculation. 29
2.3 The four different classes contained in the drink and work dataset. 30
2.4 Signal channels of the drink and work experiment. 30
2.5 Signal channels of the bicycle repair and the car workshop experiment. . . 32
2.6 The 23 different classes contained in the bicycle repair dataset. 33
2.7 The 19 different classes contained in the car workshop dataset. 34

4.1 Number of clusters for every class and channel before (white) and after (gray) sub-/superset removal of the *drink and work* experiment. 53
4.2 Number of clusters for every class before (white) and after (gray) sub-/superset removal of all three evaluation experiments. 54
4.3 Best class order (1 = best, 23 = worst) by cluster precision (Figure 4.5b), best *invariants classification* result with sweep over all parameters (Figure 6.5), *invariants classification* with known class instance borders (Figure 6.11) and *invariants classification* with activity spotting (Figure 7.7) of the bicycle repair experiment. The colors are connected to the activity classes (class 11 = dark red ...) in the best class order of the cluster precision. . . 59
4.4 Best class order (1 = best, 19 = worst) by cluster precision (Figure 4.5c), best *invariants classification* result with sweep over all parameters (Figure 6.7), *invariants classification* with known class instance borders (Figure 6.12) and *invariants classification* with activity spotting (Figure 7.8) of the car workshop experiment. The colors are connected to the activity classes (class 14 = dark red ...) in the best class order of the cluster precision. . . 59

6.1 Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both applied on the drink and work, the bicycle and the car workshop experiment. 93
6.2 Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment. 95

6.3 Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both with *invariant detection strategy set variation* applied on the drink and work, the bicycle and the car workshop experiment. 100
6.4 Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment. 100

7.1 Activity spotting variation results: mean precision, mean recall, mean F1-score and mean SET measure. 112
7.2 Mean precision, mean recall, and F1 score of the invariants classifier, the conventional classifier and the fusion of both applied on the bicycle and the car workshop experiment. 112
7.3 Distribution of the different classification strategy (in percent) in the invariants classifier / classifier fusion of the bicycle and the car workshop experiment. 113
7.4 Mean SET measure of invariants classifier results of the bicycle repair and car workshop experiment. SET measure values are given as a percentage of the length of all ground truth activities of the appropriate class. We calculated the SET values on: all invariants results (all); all invariants results used in the fusion (allfusion); all invariants results used in the fusion which return a better result than the conventional classifier (all pos fusion). 114

Bibliography

[1] Barbara E Ainsworth, William L Haskell, Stephen D Herrmann, Nathanael Meckes, David R Bassett Jr, Catrine Tudor-Locke, Jennifer L Greer, Jesse Vezina, Melicia C Whitt-Glover, and Arthur S Leon. 2011 compendium of physical activities: a second update of codes and met values. *Medicine and science in sports and exercise*, 43(8):1575–1581, 2011. 4

[2] Oliver Amft, David Bannach, Gerald Pirkl, Matthias Kreil, and Paul Lukowicz. Towards wearable sensing-based assessment of fluid intake. In *PerCom Workshops*, pages 298–303. IEEE, 2010. 8, 29, 31, 80, 82, 83, 88, 91, 94, 124

[3] Oliver Amft, Holger Junker, and Gerhard Tröster. Detection of eating and drinking arm gestures using inertial body-worn sensors. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on*, pages 160–163. IEEE, 2005. 39, 65

[4] Oliver Amft and Gerhard Tröster. Recognition of dietary activity events using on-body sensors. *Artif Intell Med*, 42(2):121–136, February 2008. 80

[5] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop on Ambient Assisted Living*, pages 216–223. Springer, 2012. 65

[6] Alberto Apostolico, Mary Ellen Bock, and Stefano Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3-4):283–311, 2003. 9

[7] Eric R Bachmann, Robert B McGhee, Xiaoping Yun, and Michael J Zyda. Inertial and magnetic posture tracking for inserting humans into networked virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 9–16. ACM, 2001. 22

[8] David Bannach, Oliver Amft, Kai S Kunze, Ernst Heinz, Gerhard Troster, Paul Lukowicz, et al. Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game. In *Computational*

Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, pages 32–39. IEEE, 2007. 8, 39

[9] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *Pervasive computing*, pages 1–17. Springer, 2004. 3, 38, 65

[10] Gerald Bauer, Ulf Blanke, Paul Lukowicz, and Bernt Schiele. User independent, multi-modal spotting of subtle arm actions with minimal training data. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 8–13. IEEE, 2013. 39

[11] Akram Bayat, Marc Pomplun, and Duc A Tran. A study on human activity recognition using accelerometer data from smartphones. *Procedia Computer Science*, 34:450–457, 2014. 6

[12] Eugen Berlin and Kristof Van Laerhoven. Detecting leisure activities with dense motif discovery. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 250–259. ACM, 2012. 9

[13] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994. 50

[14] Gerald Bieber, Jörg Voskamp, and Bodo Urban. Activity recognition for everyday life on mobile phones. In *International Conference on Universal Access in Human-Computer Interaction*, pages 289–296. Springer, 2009. 2

[15] Ulf Blanke and Bernt Schiele. Daily routine recognition through activity spotting. In *Location and Context Awareness*, pages 192–206. Springer, 2009. 39

[16] Ulf Blanke, Bernt Schiele, Matthias Kreil, Paul Lukowicz, Bernhard Sick, and Thiemo Gruber. All for one or one for all? combining heterogeneous features for activity spotting. In *PerCom Workshops*, pages 18–24, 2010. 40

[17] Ulf Mario Blanke. *Recognizing Complex Human Activity Based on Activity Spotting*. PhD thesis, Technische Universität Darmstadt, 2010. 9

[18] Michael Buettner, Richa Prasad, Matthai Philipose, and David Wetherall. Recognizing daily activities with rfid-based sensors. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 51–60. ACM, 2009. 2

[19] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. *Journal of computational biology*, 9(2):225–242, 2002. 9

[20] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):33, 2014. 2

[21] Andreas Bulling, Jamie A Ward, Hans Gellersen, and Gerhard Troster. Eye movement analysis for activity recognition using electrooculography. *IEEE transactions on pattern analysis and machine intelligence*, 33(4):741–753, 2011. 3, 65

[22] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974. 51

[23] Raymond B Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1(2):245–276, 1966. 51

[24] Jianfeng Chen, AlvinHarvey Kam, Jianmin Zhang, Ning Liu, and Louis Shue. Bathroom activity monitoring based on sound. In Hans-W. Gellersen, Roy Want, and Albrecht Schmidt, editors, *Pervasive Computing*, volume 3468 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin Heidelberg, 2005. 7

[25] Jingyuan Cheng, Oliver Amft, and Paul Lukowicz. Active capacitive sensing: Exploring a new wearable sensing modality for activity recognition. In *Pervasive Computing*, pages 319–336. Springer, 2010. 2, 7

[26] Jingyuan Cheng, David Bannach, and Paul Lukowicz. On body capacitive sensing for a simple touchless user interface. In *Medical Devices and Biosensors, 2008. ISSS-MDBS 2008. 5th International Summer School and Symposium on*, pages 113–116. IEEE, 2008. 8

[27] Jingyuan Cheng, Mathias Sundholm, Bo Zhou, Marco Hirsch, and Paul Lukowicz. Smart-surface: Large scale textile pressure sensors arrays for activity recognition. *Pervasive and Mobile Computing*, 2016. 7

[28] Jingyuan Cheng, Mathias Sundholm, Bo Zhou, Matthias Kreil, and Paul Lukowicz. Recognizing subtle user activities and person identity with cheap resistive pressure sensing carpet. In *Intelligent Environments (IE), 2014 International Conference on*, pages 148–153. IEEE, 2014. 7

[29] Jingyuan Cheng, Bo Zhou, Kai Kunze, Carl Christian Rheinländer, Sebastian Wille, Norbert Wehn, Jens Weppner, and Paul Lukowicz. Activity recognition and nutrition monitoring in every day situations with a textile capacitive neckband. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 155–158. ACM, 2013. 8

[30] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–498. ACM, 2003. 10

[31] Thomas M Cover and Joy A Thomas. *Elements of information theory 2nd edition.* Wiley-interscience, 2006. 67

[32] Modan K Das and Ho-Kwok Dai. A survey of dna motif finding algorithms. *BMC bioinformatics*, 8(Suppl 7):S21, 2007. 9

[33] David L Davies and Donald W Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):224–227, 1979. 51

[34] Stefan Dernbach, Barnan Das, Narayanan C Krishnan, Brian L Thomas, and Diane J Cook. Simple and complex activity recognition through smart phones. In *Intelligent Environments (IE), 2012 8th International Conference on*, pages 214–221. IEEE, 2012. 2

[35] Richard W DeVaul and Steve Dunn. Real-time motion classification for wearable computing applications. *2001 Project Paper*, 2001. 38

[36] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008. 50

[37] Joseph C DunnâĂă. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1974. 51

[38] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, 1994. 50

[39] Dominik Fisch, Thiemo Gruber, and Bernhard Sick. Swiftrule: Mining comprehensible classification rules for time series analysis. *Knowledge and Data Engineering, IEEE Transactions on*, 23(5):774–787, 2011. 66

[40] James Fogarty, Carolyn Au, and Scott E Hudson. Sensing from the basement: a feasibility study of unobtrusive and low-cost home activity recognition. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 91–100. ACM, 2006. 2

[41] Erich Fuchs, Thiemo Gruber, Jiri Nitschke, and Bernhard Sick. On-line motif detection in time series with SwiftMotif. *Pattern Recognition*, 42(11):3015 – 3031, 2009. 10

[42] Erich Fuchs, Thiemo Gruber, Jiri Nitschke, and Bernhard Sick. Online segmentation of time series based on polynomial least-squares approximations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2232–2245, 2010. 39

[43] Erich Fuchs, Thiemo Gruber, Jiri Nitschke, and Bernhard Sick. Online segmentation of time series based on polynomial least-squares approximations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(12):2232–2245, 2010. 66

[44] Lei Gao, AK Bourke, and John Nelson. Evaluation of accelerometer based multi-sensor versus single-sensor activity recognition systems. *Medical engineering & physics*, 36(6):779–785, 2014. 6

[45] Aristides Gionis and Heikki Mannila. Finding recurrent sources in sequences. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 123–130. ACM, 2003. 9

[46] Agnes Grünerbl, Patricia Oleksy, Gernot Bahle, Christian Haring, Jens Weppner, and Paul Lukowicz. Towards smart phone based monitoring of bipolar disorder. In *Proceedings of the Second ACM Workshop on Mobile Systems, Applications, and Services for HealthCare*, page 3. ACM, 2012. 8

[47] Ju Han and Bir Bhanu. Human activity recognition in thermal infrared imagery. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 17–17. IEEE, 2005. 2

[48] Derek Hao Hu, Sinno Jialin Pan, Vincent Wenchen Zheng, Nathan Nan Liu, and Qiang Yang. Real world activity recognition with multiple goals. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 30–39. ACM, 2008. 65

[49] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. 50

[50] Bastian Hartmann and Norbert Link. Gesture recognition with inertial sensors and optimized dtw prototypes. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 2102–2109. IEEE, 2010. 64

[51] Zhenyu He and Lianwen Jin. Activity recognition from acceleration data based on discrete consine transform and svm. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 5041–5044. IEEE, 2009. 65

[52] Peter Hevesi, Sebastian Wille, Gerald Pirkl, Norbert Wehn, and Paul Lukowicz. Monitoring household activities and user location with a cheap, unobtrusive thermal sensor array. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 141–145, New York, NY, USA, 2014. ACM. 7

[53] Yu-Jin Hong, Ig-Jae Kim, Sang Chul Ahn, and Hyoung-Gon Kim. Activity recognition using wearable sensors for elder care. In *Future Generation Communication and Networking, 2008. FGCN'08. Second International Conference on*, volume 2, pages 302–305. IEEE, 2008. 2

[54] Tâm Huynh, Mario Fritz, and Bernt Schiele. Discovery of activity patterns using topic models. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 10–19. ACM, 2008. 39

[55] Tâm Huynh and Bernt Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 159–163. ACM, 2005. 39

[56] Keki B Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993. 67

[57] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. 50

[58] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008. 39

[59] Victor Kaptelinin and Bonnie Nardi. Activity theory in a nutshell. In *Acting with Technology: Activity Theory and Interaction Design*, pages 29–âĂŞ72, 2006. 4

[60] Sidney Katz, Thomas D Downs, Helen R Cash, and Robert C Grotz. Progress in development of the index of adl. *The gerontologist*, 10(1 Part 1):20–30, 1970. 4

[61] W Larry Kenney and Percy Chiu. Influence of age on thirst and fluid intake. *Medicine and Science in Sports and Exercise*, 33(9):1524–1532, 2001. 20

[62] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001. 10

[63] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE, 2001. 39

[64] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. *Data mining in time series databases*, 57:1–22, 2004. 39

[65] Eamonn J Keogh and Michael J Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, volume 98, pages 239–243, 1998. 10

[66] Aftab Khan, Nils Hammerla, Sebastian Mellor, and Thomas Plötz. Optimising sampling rates for accelerometer-based human activity recognition. *Pattern Recognition Letters*, 2016. 6

[67] Ming Hsiao Ko, Geoff West, Svetha Venkatesh, and Mohan Kumar. Online context recognition in multisensor systems using dynamic time warping. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 283–288. IEEE, 2005. 64

[68] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995. 20

[69] Andreas Krause, Daniel P Siewiorek, Asim Smailagic, and Jonny Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *null*, page 88. IEEE, 2003. 51

[70] Matthias Kreil, Georg Ogris, and Paul Lukowicz. Muscle activity evaluation using force sensitive resistors. In *Medical Devices and Biosensors, 2008. ISSS-MDBS 2008. 5th International Summer School and Symposium on*, pages 107–110. IEEE, 2008. 7

[71] Matthias Kreil, Bernhard Sick, and Paul Lukowicz. Dealing with human variability in motion based, wearable activity recognition. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 36–40. IEEE, 2014. 17, 38, 50, 64, 78

[72] Matthias Kreil, Bernhard Sick, and Paul Lukowicz. Coping with variability in motion based activity recognition, **best paper award**. In *Proceedings of the 3nd international Workshop on Sensor-based Activity Recognition and Interaction*. ACM, 2016. 17, 38, 50, 64, 78

[73] Matthias Kreil, Kristof Van Laerhoven, and Paul Lukowicz. Allowing early inspection of activity data from a highly distributed bodynet with a hierarchical-clustering-of-segments approach. In *IEEE BSN*, Boston, MA, 2013. IEEE Press, IEEE Press. 17, 38, 50, 64, 78

[74] Kai Kunze, Michael Barry, Ernst A Heinz, Paul Lukowicz, Dennis Majoe, and Jürg Gutknecht. Towards recognizing tai chi-an initial experiment using wearable sensors. In *Applied Wearable Computing (IFAWC), 2006 3rd International Forum on*, pages 1–6. VDE, 2006. 3, 65

[75] Kai Kunze and Paul Lukowicz. Dealing with sensor displacement in motion-based onbody activity recognition systems. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 20–29. ACM, 2008. 2, 6

[76] Kai Kunze, Paul Lukowicz, Holger Junker, and Gerhard Tröster. Where am i: Recognizing on-body positions of wearable sensors. In *Location-and context-awareness*, pages 264–275. Springer, 2005. 6

[77] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011. 6

[78] Yongjin Kwon, Kyuchang Kang, and Changseok Bae. Unsupervised learning for human activity recognition using smartphone sensors. *Expert Systems with Applications*, 41(14):6067–6074, 2014. 51

[79] Robert LiKamWa, Yunxin Liu, Nicholas D Lane, and Lin Zhong. Can your smartphone infer your mood. In *PhoneSense workshop*, pages 1–5, 2011. 8

[80] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003. 9

[81] Jessica Lin Eamonn Keogh Stefano Lonardi and Pranav Patel. Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002. 9

[82] Henk J Luinge and Peter H Veltink. Measuring orientation of human body segments using miniature gyroscopes and accelerometers. *Medical and Biological Engineering and computing*, 43(2):273–282, 2005. 21

[83] Paul Lukowicz, Jamie A Ward, Holger Junker, Mathias Stäger, Gerhard Tröster, Amin Atrash, and Thad Starner. Recognizing workshop activity using body worn microphones and accelerometers. In *International Conference on Pervasive Computing*, pages 18–32. Springer, 2004. 3, 65

[84] Jani Mäntyjärvi, Johan Himberg, and Tapio Seppänen. Recognizing human motion with multiple acceleration sensors. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 2, pages 747–752. IEEE, 2001. 6

[85] Jenny Margarito, Rim Helaoui, Anna M Bianchi, Francesco Sartor, and Alberto G Bonomi. User-independent recognition of sports activities from a single wrist-worn accelerometer: A template-matching-based approach. *IEEE Transactions on Biomedical Engineering*, 63(4):788–796, 2016. 2

[86] Uwe Maurer, Asim Smailagic, Daniel P Siewiorek, and Michael Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4–pp. IEEE, 2006. 6

[87] Sinziana Mazilu, Ulf Blanke, Michael Hardegger, Gerhard Tröster, Eran Gazit, and Jeffrey M Hausdorff. Gaitassist: a daily-life support and training system for parkinson's disease patients with freezing of gait. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2531–2540. ACM, 2014. 8

[88] David Minnen, Charles L Isbell, Irfan Essa, and Thad Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 615. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007. 10

[89] David Minnen, Thad Starner, Irfan Essa, and Charles Isbell. Discovering characteristic actions from on-body sensor data. In *Wearable computers, 2006 10th IEEE international symposium on*, pages 11–18. IEEE, 2006. 10

[90] David Mizell. Using gravity to estimate accelerometer orientation. In *null*, page 252. IEEE, 2003. 6

[91] Raúl Montoliu, Raúl Martín-Félez, Joaquín Torres-Sospedra, and Adolfo Martínez-Usó. Team activity recognition in association football using a bag-of-words-based method. *Human movement science*, 41:165–178, 2015. 8

[92] Peter Morguet and Manfred Lang. Spotting dynamic hand gestures in video image sequences using hidden markov models. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 193–197. IEEE, 1998. 40

[93] Peter Morguet and Michael Lang. An integral stochastic approach to image sequence segmentation and classification. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 5, pages 2705–2708. IEEE, 1998. 40

[94] Bobak J Mortazavi, Mohammad Pourhomayoun, Sunghoon Ivan Lee, Suneil Nyamathi, Brandon Wu, and Majid Sarrafzadeh. User-optimized activity recognition for exergaming. *Pervasive and Mobile Computing*, 2015. 8

[95] Long-Van Nguyen-Dinh, Daniel Roggen, Alberto Calatroni, and Gerhard Tröster. Improving online gesture recognition with template matching methods in accelerometer data. In *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*, pages 831–836. IEEE, 2012. 64

[96] Wei Niu, Jiao Long, Dan Han, and Yuan-Fang Wang. Human activity detection and recognition for video surveillance. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 1, pages 719–722. IEEE, 2004. 2

[97] Georg Ogris. *Multi-modal on-body sensing of human activities.* PhD thesis, University of Passau, 2009. 9, 31, 32, 33, 44

[98] Georg Ogris, Matthias Kreil, and Paul Lukowicz. Using fsr based muscule activity monitoring to recognize manipulative arm gestures. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 45–48. IEEE, 2007. 2, 6

[99] Georg Ogris, Paul Lukowicz, Thomas Stiefmeier, and Gerhard Tröster. Continuous activity recognition in a maintenance scenario: combining motion sensors and ultrasonic hands tracking. *Pattern Analysis and Applications*, 15(1):87–111, 2012. 9, 32, 80, 82, 84, 89, 91, 94, 124

[100] Georg Ogris, Thomas Stiefmeier, Holger Junker, Paul Lukowicz, and Gerhard Tröster. Using ultrasonic hand tracking to augment motion analysis based recognition of manipulative gestures. In *ISWC*, pages 152–159. IEEE Computer Society, 2005. 9, 32

[101] Georg Ogris, Thomas Stiefmeier, Paul Lukowicz, and Gerhard Tröster. Using a complex multi-modal on-body sensor system for activity spotting. In *Proceedings of the 2008 12th IEEE International Symposium on Wearable Computers*, ISWC '08, pages 55–62, Washington, DC, USA, 2008. IEEE Computer Society. 3, 33, 65

[102] Juha Pärkkä, Luc Cluitmans, and Miikka Ermes. Personalization algorithm for real-time activity recognition using pda, wireless motion bands, and binary decision tree. *IEEE Transactions on Information Technology in Biomedicine*, 14(5):1211–1215, 2010. 65

[103] Kurt Partridge and Philippe Golle. On using existing time-use study data for ubiquitous computing applications. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 144–153. ACM, 2008. 4

[104] Matej Perše, Matej Kristan, Stanislav Kovačič, Goran Vučkovič, and Janez Perš. A trajectory-based analysis of coordinated team activity in a basketball game. *Computer Vision and Image Understanding*, 113(5):612–621, 2009. 8

[105] Pavel A Pevzner, Sing-Hoi Sze, et al. Combinatorial approaches to finding subtle signals in dna sequences. In *ISMB*, volume 8, pages 269–278, 2000. 10

[106] Matthai Philipose, Kenneth P Fishkin, Mike Perkowitz, Donald J Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE pervasive computing*, 3(4):50–57, 2004. 2

[107] Gerald Pirkl, Karl Stockinger, Kai Kunze, and Paul Lukowicz. Adapting magnetic resonant coupling based relative positioning technology for wearable activitiy recogniton. In *Wearable Computers, 2008. ISWC 2008. 12th IEEE International Symposium on*, pages 47–54. IEEE, 2008. 2, 7

[108] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. 40

[109] Cliff Randell and Henk Muller. Context awareness by analysing accelerometer data. Technical Report CSTR-00-009, Department of Computer Science, University of Bristol, August 2000. 5

[110] Michalis Raptis, Iasonas Kokkinos, and Stefano Soatto. Discovering discriminative action parts from mid-level video representations. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1242–1249. IEEE, 2012. 51

[111] Daniel Roetenberg, Henk J Luinge, Chris Baten, and Peter H Veltink. Compensation of magnetic disturbances improves inertial and magnetic sensing of human body segment orientation. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 13(3):395–405, 2005. 21

[112] Don J Rude, Stephen Adams, and Peter A Beling. A benchmark dataset for depth sensor based activity recognition in a manufacturing process. *IFAC-PapersOnLine*, 48(3):668–674, 2015. 9

[113] Edward S Sazonov, George Fulk, James Hill, Yves Schutz, and Raymond Browning. Monitoring of posture allocations and activities by a shoe-based wearable sensor. *IEEE Transactions on Biomedical Engineering*, 58(4):983–990, 2011. 2

[114] Thomas Stiefmeier. *Real-Time Spotting of Human Activities in Industrial Environments*. PhD thesis, ETH Zurich, 2008. 9, 25, 33

[115] Thomas Stiefmeier, Georg Ogris, Holger Junker, Paul Lukowicz, and Gerhard Tröster. Combining motion sensors and ultrasonic hands tracking for continuous activity recognition in a maintenance scenario. In *ISWC*, pages 97–104. IEEE, 2006. 9, 32

[116] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Tröster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, (2):42–50, 2008. 33

[117] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Tröster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, 7(2):42–50, 2008. 80

[118] Thomas Stiefmeier, Daniel Roggen, and Gerhard Tröster. Fusion of string-matched templates for continuous activity recognition. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 41–44. IEEE, 2007. 80

[119] Thomas Stiefmeier, Daniel Roggen, and Gerhard Tröster. Gestures are strings: efficient online gesture spotting and classification using string matching. In *Proceedings of the ICST 2nd international conference on Body area networks*, page 16. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. 80

[120] Seyed Salim Tabatabaei, Mark Coates, and Michael Rabbat. Ganc: Greedy agglomerative normalized cut. *arXiv preprint arXiv:1105.0974*, 2011. 51

[121] Yoshiki Tanaka, Kazuhisa Iwamoto, and Kuniaki Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2-3):269–300, 2005. 10

[122] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing visual features for multiclass and multiview object detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(5):854–869, 2007. 81

[123] Alireza Vahdatpour, Navid Amini, and Majid Sarrafzadeh. Toward unsupervised activity discovery using multi-dimensional motif detection in time series. In *IJCAI*, volume 9, pages 1261–1266, 2009. 10

[124] Douglas L Vail, Manuela M Veloso, and John D Lafferty. Conditional random fields for activity recognition. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 235. ACM, 2007. 3, 65

[125] Kristof Van Laerhoven and Ozan Cakmakci. What shall we teach our pants? In *Wearable Computers, The Fourth International Symposium on*, pages 77–83. IEEE, 2000. 38, 50

[126] Kristof Van Laerhoven and Hans Gellersen. Spine versus porcupine: A study in distributed wearable activity recognition. In *IEEE International Symposium on Wearable Computers (ISWC 2004)*. IEEE Computer Society, IEEE Computer Society, October 2004. 6

[127] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 216–225. ACM, 2003. 64

[128] Jamie A Ward. *Activity Monitoring: Continuous Recognition and Performance Evaluation.* PhD thesis, ETH Zurich, 2006. 106

[129] Jamie A Ward, Paul Lukowicz, and Gerhard Tröster. Evaluating performance in continuous context recognition using event-driven error characterisation. In *International Symposium on Location-and Context-Awareness*, pages 239–255. Springer, 2006. 106, 125

[130] Jamie A Ward, Paul Lukowicz, Gerhard Troster, and Thad E Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1553–1567, 2006. 9

[131] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999. 1

[132] Jens Weppner and Paul Lukowicz. Bluetooth based collaborative crowd density estimation with mobile phones. In *Pervasive computing and communications (PerCom), 2013 IEEE international conference on*, pages 193–200. IEEE, 2013. 9

[133] Daniel H Wilson and Chris Atkeson. Simultaneous tracking and activity recognition (star) using many anonymous, binary sensors. In *International Conference on Pervasive Computing*, pages 62–79. Springer, 2005. 2

[134] Martin Wirz, Tobias Franke, Daniel Roggen, Eve Mitleton-Kelly, Paul Lukowicz, and Gerhard Tröster. Probing crowd density through smartphones in city-scale mass gatherings. *EPJ Data Science*, 2(1):1–24, 2013. 9

[135] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005. 67

[136] Lu Xia, Chia-Chih Chen, and JK Aggarwal. View invariant human action recognition using histograms of 3d joints. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 20–27. IEEE, 2012. 51

[137] Mti and mtx user manual. `http://www.xsens.com/`. 21

[138] Xsens. `http://www.xsens.com/`. 21

[139] Qianren Xu, Mohamed Kamel, and Magdy MA Salama. Significance test for feature subset selection on image recognition. In *Image Analysis and Recognition*, pages 244–252. Springer, 2004. 80

[140] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. VLDB, 2000. 50

[141] Chun Zhu and Weihua Sheng. Wearable sensor-based hand gesture and daily activity recognition for robot-assisted living. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):569–573, 2011. 65

[142] Andreas Zinnen. *Spotting human activities and gestures in continuous data streams*. PhD thesis, TU Darmstadt, 2009. 80, 82, 86, 89, 91, 124

[143] Andreas Zinnen, Kristof Van Laerhoven, and Bernt Schiele. Toward recognition of short and non-repetitive activities from wearable sensors. In *Ambient Intelligence*, pages 142–158. Springer, 2007. 40

[144] Andreas Zinnen, Christian Wojek, and Bernt Schiele. Multi activity recognition based on bodymodel-derived primitives. In *Location and Context Awareness*, pages 1–18. Springer, 2009. 40

CURRICULUM VITAE

Matthias Johannes Michael Kreil

Education

04/2013 - 05/2017	PhD student at the German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany.
09/2003 - 06/2004	Studies of Computer Science at the University of Glasgow, Great Britain. (Erasmus exchange)
09/1999 - 05/2007	Studies of Computer Science at the University of Passau, Germany. Graduation with degree *Dipl.-Inform.* Diploma thesis: *Wearable gesture recognition with force sensitive resistors* , supervised by Dr. Georg Ogris and Prof. Paul Lukowicz.

Experience

08/2007 - 07/2013	Research and teaching assistant, supervised by Prof. Paul Lukowicz. Embedded System Lab, University of Passau, Germany.
02/2006 - 05/2006	Intern at the Department of Computer Science, Universidad Autónoma de Occidente in Cali, Colombia.