Technische Universität München

Fakultät für Informatik

The Wearables Development Toolkit

Dr. rer. nat. Juan I. Haladjian Madrid

Habilitation

Members of the Habilitation Committee:

Univ.-Prof. Dr. Bernd Brügge
Univ.-Prof. Dr. Tobias Nipkow
Prof. Daniel Siewiorek, Ph.D

Abstract

Over the last two decades, several wearable devices have been developed with applications in the fields of health, sports, daily activity monitoring and animal welfare. While these applications highlight their potential benefit to different user groups, the development of wearable systems that are ultimately accepted by users remains a challenging task. To some extent, this is because of the lack of tools specifically designed to support their development. Integrated development environments are already available to support the development of mobile and desktop device applications. In contrast, there is up to date no integrated development environment for wearable devices.

This Habilitation presents the Wearables Development Toolkit, an integrated development environment we created to facilitate the development of wearable device applications. The Wearables Development Toolkit consists of a repository of reusable software components and a set of tools. The reusable components facilitate the development of activity recognition applications with wearable devices by encapsulating functionality commonly used to develop such applications. The set tools support common tasks developers usually have to engage in when developing activity recognition applications. These tasks include the collection and annotation of data, the analysis of data needed to devise recognition algorithms, the implementation of recognition algorithms and the assessment of their computational and recognition performance.

We demonstrate the ease of use of the toolkit by describing step-by-step the development process we followed to create a smart glove able to recognize goalkeeper training exercises. Furthermore, we provide evidence about the toolkit's generalizability to different application domains by describing how the components available in the toolkit are used to create two applications from different domains. The first one recognizes activities of daily living (e.g. sitting, walking) and the second one tracks the rehabilitation progress of patients after a hip replacement surgery. We append to this document a list of other applications developed with the toolkit.

Contents

Publication Preface

This Habilitation is based on the following publications:

Publication [1]

Haladjian, J. (2019). The Wearables Development Toolkit: An Integrated Development Environment for Activity Recognition Applications. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), 3(3)
Accepted. To be published in November 2019.

Publication [2]

Haladjian, J., Schlabbers, D., Taheri, S., Tharr, M., & Bruegge, B. (2019). Sensor-based Detection and Classification of Soccer Goalkeeper Training Exercises. Proceedings of the ACM Transactions on Internet of Things (TIoT),
Accepted. To be published in November 2019.

Publication [3]

Haladjian, J., & Bruegge, B. (2019). Teaching wearable device development with the wearables development toolkit. CEUR Workshop Proceedings, 2308, 27–28.
Proceedings: http://ceur-ws.org/Vol-2308/

Publication [4]

©ACM 2018. Reprinted with permission.
Echterhoff, J., Haladjian, J., & Bruegge, B. (2018a). Gait Analysis in Horse Sports. In Proceedings of the Fifth International Conference on Animal-Computer Interaction (p. 3).
DOI: https://doi.org/10.1145/3295598.3295601

Publication [5]

©ACM 2018. Reprinted with permission.
Echterhoff, J., Haladjian, J., & Bruegge, B. (2018b). Gait and Jump Classification in Modern Equestrian Sports. In Proceedings of the 2018 ACM International Symposium on Wearable Computers (pp. 88–91).
DOI: https://doi.org/10.1145/3267242.3267267

Publication [6]

Reprinted with permission.

Haladjian, J., Haug, J., Nueske, S., & Bruegge, B. (2018). A Wearable Sensor System for Lameness Detection in Dairy Cattle. Multimodal Technologies and Interaction, 2(2), 27.

DOI: https://doi.org/10.3390/mti2020027

Publication [7]

©IEEE 2018. Reprinted with permission.

Haladjian, J., Bredies, K., & Bruegge, B. (2018). KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries. In Proceedings of the 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN) (pp. 9–12). IEEE.

DOI: 10.1109/BSN.2018.8329646

Publication [8]

©ACM 2017. Reprinted with permission.

Haladjian, J., Ermis, A., Hodaie, Z., & Bruegge, B. (2017). iPig: Towards Tracking the Behavior of Free-roaming Pigs. In Proceedings of the Fourth International Conference on Animal-Computer Interaction (pp. 10:1--10:5). New York, NY, USA: ACM.

DOI: https://doi.org/10.1145/3152130.3152145

Publication [9]

©ACM 2017. Reprinted with permission.

Haladjian, J., Hodaie, Z., Nueske, S., & Bruegge, B. (2017). Gait Anomaly Detection in Dairy Cattle. In Proceedings of the Fourth International Conference on Animal-Computer Interaction (pp. 8:1--8:8). New York, NY, USA: ACM.

DOI: https://doi.org/10.1145/3152130.3152135

Publication [10]

©ACM 2017. Reprinted with permission.

Haladjian, J., Scheuermann, C., Bredies, K., & Bruegge, B. (2017). A Smart Textile Sleeve for Rehabilitation of Knee Injuries. In Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers (pp. 49–52). New York, NY, USA: ACM.

DOI: https://doi.org/10.1145/3123024.3123151

Chapter 1

Introduction

The first computing devices were as large as an entire room, so expensive that only large companies and universities could afford them and required human operators to physically manipulate their mechanical plugs and switches. The miniaturization of electronics, the mass production of electronic parts and the advances in the research fields of pattern recognition and human-computer interaction, have given place to smaller, cheaper and better usable computing devices than ever before.

Today, users are no longer required to actively interact with a computing device. Instead, computing devices infer information about the user and her context autonomously. For example, today's smartphones are already able to measure the location of the user, count steps and recognize whether the user is walking, running or riding a bike regardless of the user's location and without demanding her attention. This information can be used to adapt a user interface to the user's needs or to help athletes improve their skills. The ability of a computer system to retrieve information from the environment and adapt accordingly is called *context awareness*.

The term *context awareness* was first used in 1994 by Schilit and Theimer [88] to refer to the ability of a system to infer and make use of the location of a user to adapt a user interface. Until the late 90s, the location of the user remained the main source of information used by context-aware applications. In the following years, different applications started to take advantage of other contextual information such as the orientation of a device [30], the time [83] and the user's emotional and mental condition [26]. In 1999, Anind Dey extended the definition of context to: "*any information that can be used to characterize the situation of an entity*" [2]. He used the term *entity* to refer to the user of an application, a place or any object related to the interaction between the user and the application. The 21st century has seen a large number of context-aware applications developed for mobile, ubiquitous and wearable devices. The focus of this thesis is on context-aware applications that infer information about the user using wearable devices. In the rest of this document, we refer to applications that leverage wearable sensor signals to extract information about the wearer as *activity recognition applications*.

Activity recognition applications based on wearable sensors have been developed for a variety of fields. In the field of *medicine*, multiple applications have been developed that study the gait of a patient (e.g. patients of Parkinson's Disease [73, 15, 63]) and others that support the rehabilitation process after an injury [45, 38] or a stroke

[11]. In the *sports* field, a variety of applications have been developed that compute performance parameters and give feedback to athletes during or after training to help them improve their skills. Concrete sport types include: table tennis [18], soccer [112], swimming [12], cricket [55] and ski jumping [33]. In the more general field of *daily activity monitoring,* applications have been developed to recognize activities such as drinking [86], eating [5, 7] and falling [21]. Furthermore, a growing interest in the wellbeing of animals have made them an interesting target user of activity recognition applications as well. Target animal users of activity recognition applications include livestock (e.g. cows [44, 40, 46], pigs [42], sheep [102], goats [68]), animals that participate in sport applications (e.g. horses [27, 28]) and domestic animals, mostly dogs [56, 98].

While the broad variety of applications introduced by the research community has already highlighted their potential benefits to end-users, developing activity recognition applications with wearables that are ultimately accepted by end users is still a difficult task. The development of such applications requires specialized knowledge in multiple disciplines, including hardware design, firmware development and data science. Furthermore, activity recognition applications are bound to the computational constraints of the wearable device (memory, processing capability), as well as to additional requirements set by the end user (small, comfortable, accurate recognition, long-lasting battery). Design decisions span across hardware (e.g. CPU, memory, sensors), software (e.g. sensor ranges, recognition algorithms) and hardware-software mapping (e.g. doing computations on the wearable vs delegating them to a mobile device). As it is often impossible to find a design that satisfies every requirement, different designs usually have to be assessed and compared before a decision for a particular design can be made. Unfortunately, the suitability of a particular design can rarely be assessed in advance, without collecting data, annotating it, developing a recognition algorithm and assessing its recognition performance and computational requirements.

Many activity recognition applications rely on similar functionality such as signal processing methods and machine learning classifiers and are developed with similar development processes. The typical development lifecycle of an activity recognition application involves the collection and annotation of data, the implementation of one or more recognition algorithms and the assessment of the performance of the algorithms. Nevertheless, activity recognition applications with wearable devices are still being developed with general-purpose programming languages, tools and frameworks such as Matlab, Python, R, C++ and WEKA [47] that do not directly support the development lifecycle of activity recognition applications. As these technologies were not originally designed for activity recognition applications, their entrance barrier to this kind of applications is still high.

In this Habilitation, we present the Wearables Development Toolkit (WDK), an Integrated Development Environment (IDE) specifically designed for activity recognition applications with wearables. The WDK consists of a set of reusable components with common functionality used across activity recognition applications. It also offers four tools to: 1) annotate sensor data, 2) study the data and the effects of different signal processing methods on it, 3) develop activity recognition algorithms by reusing the set of reusable components within a textual and visual programming interface and 4) assess their recognition and computational performance. We present the design of the WDK, its different tools and demonstrate its usage with three different demonstrations from different domains.

1.1 Research Process

We developed the WDK following a formative research process. Formative research processes iteratively assess a project or method in order to guide its development [92]. In contrast, summative processes evaluate the performance of a method or project after their implementation or execution. A formative research process enabled us to iteratively assess the IDE and refine its requirements accordingly. This was necessary because the requirements for an IDE for wearable devices were not defined when we started the development of the WDK.

We started the development of the WDK by identifying abstractions based on our experience in the field of wearable computing and activity recognition. We also performed a systematic literature review to validate and extend our set of abstractions. Next, we refined and extended these abstractions by iteratively developing a family of wearable device applications. The development of these applications enabled us to refine the existing abstractions and to identify new requirements for the IDE. Since 2016, we have developed more than five different wearable device applications with the WDK. To make the WDK available for other developers to create their own applications, the WDK has been made open source under the MIT license[1].

1.2 Outline

This is a publication-based Habilitation. It contains a summary of our contribution and a copy of ten conference and journal publications. The summary includes an overview of the field of activity recognition with wearable sensors (Chapter 2), a detailed description of the WDK (Chapter 3) and an evaluation demonstrating how different applications are created with the WDK (Chapter 4). In addition, Chapter 5 discusses future research directions to continue our work. In Chapter 6, we include a reprint of the publications this Habilitation is based on.

Chapter 2 provides an overview of the field of activity recognition with wearable sensors. Activity recognition applications recognize user activities using sensor data. We describe what kind of sensors have been used in previous applications and how patterns can be extracted from sensor data by performing a sequence of computations known as the *Activity Recognition Chain*. We discuss in detail common computations done, most of which are available as reusable components in the WDK. The chapter also describes the typical development process of an activity recognition application, which is supported by the WDK. Finally, we compare the WDK to previously created

[1]https://github.com/avenix/WDK

toolkits that ease the development of ubiquitous computing applications.

Chapter 3 describes the WDK's design, including its design goals, architecture and object design. An important goal of the WDK was to lower the entrance barrier to the development of activity recognition applications with wearables. To this end, the toolkit offers a set of high-level reusable components that have been refined based on the several applications we developed with the WDK. We also present how we organized the reusable components in the WDK in a repository architecture. The repository of reusable components is designed as layered architecture with two main layers. The *Runtime Components* layer contains reusable components that are used by activity recognition applications at runtime and the *Development Components* layer contains functionality used during development (e.g. methods to assess the performance of an algorithm). Finally, this chapter presents the different tools offered by the WDK. These tools facilitate common development tasks such as the annotation of time series together with a reference video.

After having presented the WDK, Chapter 4 demonstrates its usage with three different applications. With this evaluation, we pursue two goals. First, we demonstrate how easily activity recognition applications can be developed with the WDK. To this end, we describe step-by-step how we used the WDK to develop an application that recognizes goalkeeper training exercises using an inertial sensor inserted in a goalkeeper's glove. Second, we demonstrate the versatility of the WDK to support the development of two activity recognition algorithms from different domains. In particular, we chose to re-create an activity recognition algorithm proposed by Bao and Intille [14] that recognizes daily activities such as walking, jogging and climbing up stairs and another one that tracks the rehabilitation exercises performed by patients after a hip replacement surgery [41].

Chapter 5 summarizes our contribution and discusses future research directions. Our main contribution is an integrated development environment including a repository of reusable components and a set of tools to support the development of activity recognition applications with wearables. The WDK facilitates the development of manually-crafted activity recognition algorithms, but does not support the development of applications that rely on these algorithms. Future work could investigate how to automatize the development process to enable the creation of recognition algorithms automatically based on an annotated data set and identify domain-specific abstractions to ease the development of applications that rely on recognition algorithms.

Chapter 6 provides a summary and copy of the publications this Habilitation is based on. For each publication, we summarize its main research contribution, describe how it relates to the WDK and mention how the author of this Habilitation contributed to them. Publications 1 [36] and 3 [39] describe the WDK and every other publication we present describes an activity recognition application developed with the WDK.

Chapter 2

Background

This chapter provides an overview of the field of activity recognition. Section 2.1 discusses the state of the art in activity recognition, including the terminology used in the field, common recognition methods and concrete applications. It also describes the typical development lifecycle of an activity recognition application. Section 2.2 provides an overview of previous work to facilitate the development of physical devices and discusses how these relate to our toolkit.

2.1 Activity Recognition

Activity recognition is often referred to as *Human Activity Recognition*, abbreviated as *"HAR"*. In this document, we avoid the term *human* because the target user of an activity recognition application might be an animal [44, 102, 56, 98, 42]. Activities include (but are not limited to) hand gestures [31, 8, 71], activities of daily living (e.g. walking, sitting, jogging) [14, 95, 81], specific motions in sport applications (e.g. serves in table-tennis [18], shots and passes in soccer [89] or batting shots in cricket [55]) and abnormal conditions (e.g. falls [1, 21]). Applications developed so far have used a variety of sensors, including inertial [44, 28, 96], pressure [112, 71], muscle electrical activity [6, 66, 76], electrocardiogram (ECG) [54, 69] sensors and microphones [104, 61, 109]. Besides recognizing activities, applications often need to compute *quality metrics* about the recognized activities, such as the stability while walking [41, 73], the "smoothness" of a turn performed by a horse in dressage [96] or the velocity and jump length in a ski jumping application [33]. Usually, activities have to be recognized before quality metrics can be extracted from them. While some applications consider every activity performed by the wearer as relevant to the application [14, 28, 95], many applications need to detect sporadic events signaling relevant activities first [44, 18, 55, 89, 24]. Accurately detecting the relevant events in a stream of sensor data while avoiding the detection of irrelevant ones can be challenging, depending on the sparsity of the relevant activities. Most applications require executing activity recognition algorithms in real time, while others perform the recognition *offline*, after the entire data is available.

2.1.1 The Activity Recognition Chain

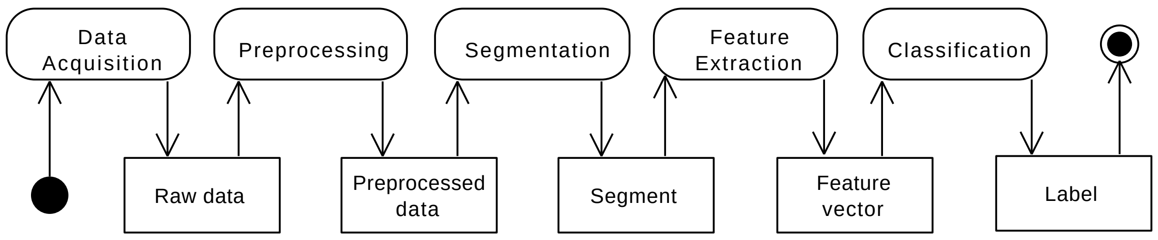


Figure 2.1: UML activity diagram shows the typical sequence of computations performed in activity recognition applications.

Most activity recognition applications process sensor signals in a sequence of *stages* (represented as activities in Figure 2.1). This sequence is often referred to as the *Activity Recognition Chain*, often abbreviated as ARC and is also referred to as *manually-crafted recognition chain*. In the first stage of the recognition chain, the sensors are read and the signals produced by them are stored in memory. Each signal consists of several samples and the set of signals acquired from the different sensors before performing any processing is called *raw data*.

A *Preprocessing* stage is often applied to prepare the *raw data* for feature extraction *[19]*. Common goals of the preprocessing stage are to eliminate noise in the acquired signal, to synchronize sensor signals in case they were acquired by different sensors and possibly at different sampling frequencies and to compute new signals by aggregating raw signals. Low-pass filters are a common preprocessing technique used to eliminate high-frequency noise in a signal produced by an accelerometer sensor [24, 96, 41]. Furthermore, several activity recognition applications aggregate raw accelerometer, gyroscope and possibly magnetometer signals and compute the linear acceleration, gravity vector and quaternion using sensor fusion techniques such as a *Kalman* or *Complementary* filter. In fact, these computations are so common, that several commercial Inertial Measurement Units already compute these values directly on the device.

The goal of the *Segmentation* stage is to split the *preprocessed data* into *segments*, from which information about the underlying activity can be inferred. In order to enable the next stages to accurately infer information from a segment, the segment should ideally contain samples corresponding to a single activity. Applications that process periodic or static signals can usually rely on a sliding window to create segments [14, 95]. In contrast, applications that need to detect the occurrence of sporadic events first, usually create segments around the detected events [18, 44, 43, 89, 96].

In the *Feature Extraction* stage, a set of values are computed from each *segment* and grouped under a so-called *feature vector*. The goal of the *Feature Extraction* is to produce *feature vector*s that are similar among instances of the same activity and different among instances of different activities. While most applications rely on a set of common feature extraction algorithms, tailoring feature extraction algorithms to a particular application can lead to improved recognition performance. Feature extraction algorithms can be categorized into those that operate on the time-domain and those that operate on the frequency-domain. Common time-domain feature extraction algorithms include statistical measurements such as the *mean*, *median*, *variance*, *minimum*, *maximum* and other mathematical functions such as the *zero-crossing rate* and the difference between the maximum and minimum values of a signal. Frequency-domain feature extraction algorithms include those that rely on the coefficients of a Fourier transform (e.g. largest Fourier coefficient) and those that rely on the coefficients of a Wavelet transform (e.g. sum of Wavelet coefficients).

Finally, a classifier is used to determine what activity took place based on a *feature vector* and produces a *label* as result. Due to the large number of classification methods used in the literature, we dedicate the entire next section to discussing them.

2.1.2 Classification Methods

Activity recognition borrows classification methods from the more general field of *pattern recognition*. Pattern recognition methods have been categorized as: *template methods*, *statistical methods*, *syntactic methods* and *neural networks* [53]. *Template methods* recognize patterns by comparing the signals to templates using a similarity measure. For example, to recognize strides in the signal produced by an inertial sensor, data segments can be compared against a stride template and discarded if their similarity falls below a threshold [15]. *Statistical methods* represent patterns as points in an n-dimensional space. Ideally, the points corresponding to the pattern to be recognized are clustered together and lie as far away as possible from points that do not correspond to the pattern. Then, statistical methods learn how to *"draw the lines"* (i.e. identify the optimal boundaries) to separate the data points - this can usually be done automatically based on the data. *Syntactic methods* represent patterns hierarchically: each pattern is composed of simpler patterns, which are in turn composed of simpler patterns. The simplest form of a pattern is called a *primitive*. To recognize patterns, syntactic methods apply a set of rules to derive patterns from the primitives. For example, an electrocardiogram (ECG) signal can be represented with vertical and diagonal line primitives, which can be aggregated into more complex representations by applying a set of rules [97]. These complex representations can be used to determine whether an ECG signal corresponds to a healthy or unhealthy individual. *Neural networks* are weighted directed graphs where the nodes and edges are analogous to neurons and connections between neurons in a human brain. An important characteristic of neural networks is that they have the ability to learn complex relationships between inputs and outputs by adapting their weights.

Most activity recognition applications developed so far are based on *statistical methods*. Commonly used statistical methods include machine learning classifiers such as Support Vector Machines [41, 43, 82, 20, 4], k-Nearest Neighbors [94, 24, 10, 77] and decision trees [14, 65, 17]. For signals that exhibit a temporal dependency, probabilistic methods (mostly Hidden Markov Models) have been studied in different applications [64, 25, 60, 87]. *Template methods* have been used mostly to recognize hand gestures [3, 49, 99, 100] and in applications that rely in short activities with temporal dependencies (e.g. gait strides [15]). A popular template method across activity recognition applications is Dynamic Time Warping. In general, machine learning classifiers have achieved higher recognition performances than other statistical and template methods [19]. *Syntactic methods* are less common and have been used mostly in pattern recognition applications based on different kind of signals than those typically produced by wearable devices, such as video [52] and image data [16]. In the field of activity recognition with wearable sensors, syntactic methods have been applied for detecting spikes and classifying states in EEG [101] and ECG [97] signals.

Until the last decade, only a few activity recognition applications relied on *neural networks*, mostly feedforward neural networks [93, 103, 29]. Motivated by the success of deep neural networks in other pattern recognition applications, different research groups started studying their application to activity recognition with wearables. Deep neural networks studied so far include Convolutional Neural Networks (CNN) [23, 35, 111, 48, 59], Recurrent Neural Networks (RNN) [75, 48, 72] and Long-Short Term Memory (LSTM) [110, 75, 48, 72, 59]. Deep neural networks are of high interest to the research community for two main reasons. First, they have been shown to surpass manually-crafted recognition chains in terms of recognition performance in several applications. Second, deep neural networks promise to free developers from having to devise a chain of computations manually, which requires experience in signal processing, feature engineering and application domain knowledge.

While deep neural networks have become the state of the art recognition method in other research fields (e.g. computer vision and natural language processing), at least two limitations makes them impractical for most activity recognition applications with wearable devices. First, deep neural networks usually achieve high recognition performance only when fed with larger amounts of data than usually available when developing an application for a wearable device. In contrast to other applications that process a single kind of signal (e.g. image, video, text), wearable device applications vary in the kind of signals they process. As different wearable systems are based on different sensors, which might be configured differently (e.g different ranges or sampling frequencies) or placed at different positions of the body, the data collected for a particular application can rarely be reused in other applications. Second, deep neural networks are usually more computationally complex than other pattern recognition methods commonly used in activity recognition with wearables. To cope with the added computational complexity, a wearable device might need to host a higher-end CPU, more memory and a larger battery to remain functional for as long as required by the application. However, a higher recognition performance might not always justify the need for bulkier, heavier and more expensive wearable device. As a consequence, most activity recognition applications with wearable devices still rely on manually-crafted recognition chains, which the WDK is based on.

2.1.3 Architectures

In addition to wearable devices, activity recognition systems might involve mobile devices and servers in the cloud. The stages in the activity recognition chain can be mapped to the aforementioned devices in different ways. Siewiorek et al. [91] defined three common architectures for activity recognition systems: *dynamic*, *low-bandwidth* and *centralized aggregation* architectures. The *dynamic architecture* maps every stage of the activity recognition chain to the wearable device, which transmits only the results of the recognition to external devices. Among the different architectures, this architecture usually leads to the lowest data transmission rates between the wearable and external devices. As transmitting data wirelessly outside of a wearable device is usually associated with high energy costs, this architecture usually also leads to the lowest energy consumption rates among the different architectures. Another advantage of this approach is that it does not require external devices (e.g. a mobile device) to be at a close range to receive data from the wearable device. This can be convenient in different applications for usability reasons (e.g. a swimming tracker, or a tracker for free-roaming animals). Two disadvantages of this architecture is that the performance of the recognition algorithm might be limited by the computational resources available in the wearable device and that both, the raw data and feature vectors are discarded and hence, not available for further data analysis purposes.

The *low bandwidth architecture* performs every stage in the activity recognition chain until the *Feature Extraction* in the wearable device and transmits feature vectors to external devices. This architecture usually results in higher data transmission rates than the dynamic architecture. However, as this architecture makes the feature vectors available outside of the wearable device, it enables the computation of further metrics from the feature vectors, which might be relevant to application users (e.g. the energy of a tennis serve). The feature vectors can also be used to improve the performance of the recognition algorithms (e.g. by means of online machine learning).

The *centralized aggregation architecture* transmits the raw data outside of the wearable device. Depending on the sampling frequency of the different signals and amount of signals to be transmitted, systems designed based on this architecture might suffer from high energy consumption rates and possible loss of data. On the other hand, this architecture enables more complex data processing tasks by delegating the entire algorithm execution to more computationally powerful devices. An additional advantage of this architecture is that the recognition algorithm might require less effort to develop, as it does not have to be optimized for execution on an embedded device with limited computational resources.

2.1.4 Development Lifecycle

Figure 2.2 illustrates the development process of an activity recognition application. The development of an activity recognition application consists of several tasks. Developers usually start developing an activity recognition application by collecting and annotating data. The annotation is done to indicate what activity took place at each moment in time. Annotations might comprise a start and an end timestamp, indicating the occurrence of an activity with a duration in time, or a single timestamp corresponding to a sudden event. The annotation of data can be a cumbersome and time-consuming task if it requires a human to annotate the data manually [18, 27, 32].

However, some applications can use external information as annotations. For example, an application that recognizes lame cows from non-lame cows can rely on a data set that uses the cow's health condition as a reference [43, 44]. When the data has to be annotated manually, the annotations might be performed during the data collection, or after it. Annotating data after its collection enables a more precise determination of the timestamps and is commonly done using video as a reference. The annotated data constitutes a *ground truth*, which is used during the rest of the development lifecycle to devise, develop and assess the performance of activity recognition algorithms.

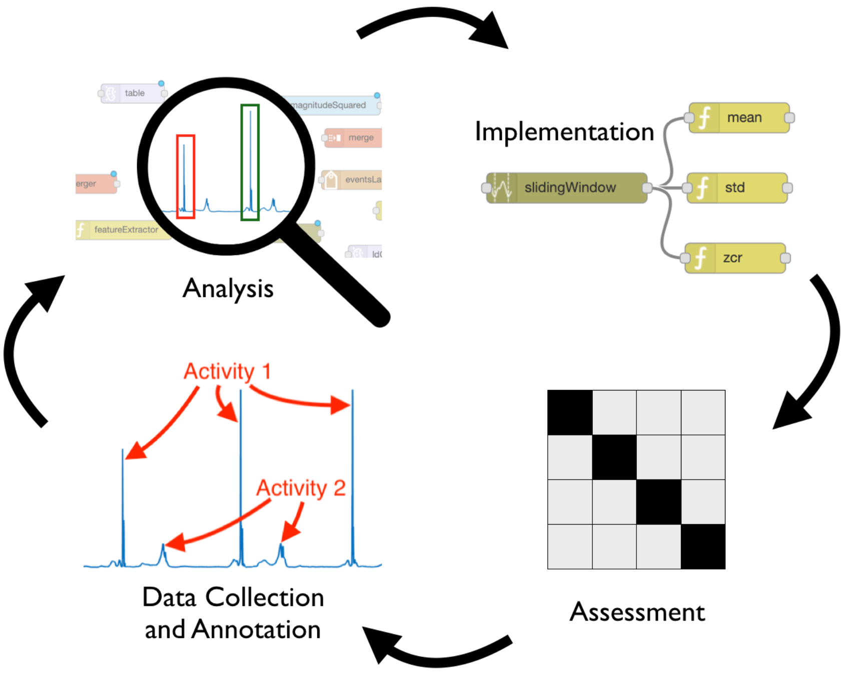


Figure 2.2: Development lifecycle of an activity recognition application.

During the *Analysis* task, developers study the data and come up with possible methods to process it. While the data analysis is mainly based on experience and intuition, developers can rely on different strategies for this task. For example, to devise recognition algorithms that effectively discriminate between different activities, developers can compare (e.g. plot) the signals corresponding to each activity. This comparison can then be used to identify properties of the signals that are similar within an activity and different among activities. To gain a better understanding about the discriminative power of a particular property, developers can compute its statistical distribution (e.g. mean, variance, histogram) for each activity. The ideal property has disjoint distributions of values among activities. In addition, developers might find it convenient to study data plots together with reference videos to understand how different parts of the signals relate to user activities, or how user activities are represented in the signals.

Once developers have decided for a recognition algorithm, they *implement* it. As many activity recognition applications rely on signal processing, statistical methods and machine learning algorithms, commonly used development languages and tools include those designed for such tasks (e.g. *Matlab* and *R*), and those that offer external libraries for such tasks (e.g. *Python* and *C++*). Most of these programming languages are interpreted, which usually makes the implementation of algorithms faster, at the cost of computational performance. For this reason, recognition algorithms implemented and tested in a development environment often have to be re-implemented on the target platform and optimized for execution on the target device.

The *Assessment* of a recognition algorithm is done to quantify its performance and determine whether it fulfills the requirements of the application. The *recognition performance* of an algorithm is quantified with different metrics, including the accuracy (i.e. percentage of correctly recognized relevant and non-relevant activities), precision (i.e. percentage of actually relevant activities among those predicted as relevant), recall (i.e. percentage of relevant activities recognized as relevant) and F1-score (harmonic mean of the precision and recall metrics). Developers might find the confusion matrix useful to gain further insight into which activities can be accurately recognized and which ones can't. Furthermore, a frame-by-frame comparison between the raw data and the recognition results can help developers understand the specific reason why an activity is poorly recognized. Recognition algorithms are usually *trained* with data to recognize activities. Since assessing the performance of an algorithm with the same data used for training it might bias the assessment results, algorithms are usually assessed with a different subset of the data than the one used during training. We call the different ways to train and assess the performance of a recognition algorithm *validation*. A common validation strategy is to exclude a part of the data set from training and use it to study the performance of a trained recognition algorithm, which is known as *hold-out validation*. A disadvantage of this method is that a fraction of the data cannot be used for training, which might lead to lower recognition performances if a machine learning classifier is not able to infer its optimal parameters based on the smaller amount of data used for training. As the data sets used in activity recognition applications tend to be small, a more common validation strategy is the *cross-validation*. Cross-validation, validates a recognition algorithm in a series of iterations. In each iteration, a different fraction of the data - a so-called *fold* - is excluded from training and used to assess the performance of the algorithm. The recognition performance of an algorithm is then determined by averaging the performance results of every fold. A particular variation of the cross-validation used in several activity recognition applications is the *leave-one-subject-out* cross-validation, which uses the data collected from different people as folds. Besides the recognition performance of an algorithm, developers are usually interested in its *computational performance*. The computational performance of an algorithm determines the requirements of the target hardware where the algorithm can be executed and enables developers to decide for a system architecture. Developers are typically interested in the amount of memory required to execute an algorithm, the time required by the wearable device to perform a computation, the energy consumption of the wearable device and the amount of data that has to be transmitted from and to the wearable device.

The aforementioned tasks can rarely be performed sequentially. Instead, most activity recognition applications are developed iteratively. One reason for it is that the requirements for an application might not be fully understood initially. Often, developers need to explore what activities can be recognized reliably before it can be decided what activities an application should recognize. Even when the requirements are well understood, it is rarely known upfront what design will satisfy them. Design decisions to be made include how much data should a machine learning classifier rely on, how the data should be annotated and what recognition methods should be used. Often, how much data should be collected is not known before assessing the performance of an algorithm, because a low recognition performance might make it necessary to collect additional data. The annotation protocol might also change if newly collected data contains activities that were not observed previously. Another reason why change might have to be introduced to the annotation protocol is to fix inconsistencies in annotations done by different individuals. Inconsistent annotations occur often when the activities to be recognized blur into each other without a clear transition. For example, the boundary between *walking* and *sitting* might be unclear in the signal produced by an inertial sensor when a person sits down on a chair after walking towards it. Refining the annotations of one or two activities might fix frequent misclassifications between two activities. Finally, developers usually decide for a particular recognition algorithm after several iterations where they implement, assess the performance of and optimize different algorithms.

2.2 Related Work

Several toolkits and development environments have been developed with the goal to lower the entrance barrier and reduce the time needed to create interactive physical devices and user interfaces distributed among multiple devices. A comprehensive overview of previously developed toolkits and evaluation methods for toolkits is available in [58]. The existing toolkits can be grouped based on the kind of application they support:

- Toolkits to modify physical objects and add interactive behaviors to them. These toolkits usually scan physical objects with 3D cameras or using mobile devices, modify their physical structure with computer-aided design functionality offered within the toolkits and print them using 3D printers. An important feature offered by these toolkits is the ability to facilitate the integration of electronic components into the objects. Toolkits under this category include: *Pineal* [57], *Retrofab* [79], *Sauron* [84], *Makers Marks* [85], *Modkit* [67] and *WorldKit* [107].

- Toolkits that support the development of smart and interactive physical environments. The applications created by these toolkits usually enable the interaction between a human and a smart room. These applications usually recognize human activities using wearable and non-wearable devices and project visual output into walls. Toolkits that fall under this category include: *EagleSense* [106], *Physikit* [50] and the *Sod-toolkit* [90].

- Toolkits to create applications distributed across multiple wearable and non-wearable devices. These applications usually have as a goal to enable the interaction between different devices while avoiding the development of data transmission and synchronization functionality. Toolkits under this category include: *XDStudio* [70], *Weave* [22], *WatchConnect* [51] and *Panelrama* [108].

- Toolkits that support the development of applications using specific hardware technologies. For example, the *Interactex* toolkit enables the development of applications that rely on smart textiles [37]. *PaperPulse* facilitates the prototyping of applications where conductive ink is used to interconnect electronic parts on paper [80]. *PaperPixels* is a set of components and design tool to create paper-based displays [74]. *OpenCapSense* offers hardware components and a software framework to facilitate the development of applications that rely on capacitive sensors [34].

In this Habilitation, we present a toolkit to facilitate the development of other kind of applications than the ones targeted by the previously mentioned toolkits. In particular, we present a development environment for activity recognition applications with wearable devices.

Several toolkits were created that support the development of applications that recognize user gestures, such as *Exemplar* [49], *MAGIC* [9], *GART* [62] and GT^2k [105]. These toolkits are similar to the WDK in that they facilitate the development of applications that recognize patterns in the signals produced by different sensors. However, they do so with specific recognition methods rather than enabling developers to study different ones. *Exemplar* and *MAGIC* rely on Dynamic Time Warping whereas *GART* and the GT^2k use a Hidden Markov Model (HMM). However, developers of activity recognition applications usually need to assess several recognition algorithms before they are able to decide for one of them. The WDK offers a diverse set of signal processing and machine learning methods commonly used in activity recognition applications, together with tools to help developers determine which method better suits the requirements of a particular application.

The *CRN Toolbox* [13] and the more recent *Gesture Recognition Toolkit* (GRT) developed by Nick Gillian [31] are perhaps the toolkits that share most similarity with the WDK. In contrast to the previously mentioned toolkits, the CRN and GRT toolkits offer a wide variety of recognition methods. While these toolkits ease the implementation (i.e. programming) and assessment of activity recognition algorithms, they do not support other equally important tasks in the development lifecycle of activity recognition applications. This is an issue because developers rarely know upfront what algorithm to implement without engaging in the iterative process we described in Section 2.1.4. In contrast to these tools, the WDK supports the entire development process of an activity recognition application.

Chapter 3

The Wearables Development Toolkit

This chapter presents the Wearables Development Toolkit (WDK) and discusses how we designed it. We start by describing the goals that drove the decisions we made during its design in Section 3.1. In Section 3.2, we present the different subsystems of the WDK and justify why we decided to organize them in a repository architectural style with Matlab as underlying development environment. Section 3.3 presents the WDK's object design including every reusable component used at runtime and the functionality used to develop activity recognition algorithms. Finally, we describe the different tools offered by the WDK in Section 3.4.

3.1 Design Goals

The WDK was designed based on four main design goals:

Low entrance barrier. The development of activity recognition applications with wearable devices has a high entrance barrier, as it requires knowledge in multiple disciplines including data science, electrical engineering, computer science and human-computer interaction. One of the main goals of the WDK is to lower the entrance barrier to activity recognition application development with high-level interfaces that hide implementation details.

Versatility. Activity recognition applications with wearable sensors can target different user groups, including patients, athletes and animals. Despite the differences across domains, most of these applications rely on similar recognition methods, as discussed in Section 2.1.2. An important goal of the WDK was to make these recognition methods available to application developers to enable the development of applications for different domains.

Extensibility. Even if the WDK offered a set of recognition methods common across different domains, application developers are likely to need to extend this functionality to fulfill the requirements of their particular applications. For example, many applications rely on custom-tailored feature extraction algorithms [96, 78]. For this reason, a goal in the design of the WDK was to make the set of reusable functionality in the WDK extensible by developers with little effort.

Performance. Assessing the performance of a recognition algorithm can lead to computationally intensive tasks, specially when the recognition relies on machine learning classifiers that have to be trained with large data sets. To address this issue, we designed the WDK to enable developers to assess the performance of different recognition algorithms quickly.

3.2 Architecture of the WDK

The architecture of the WDK is shown in Figure 3.1. The WDK is based on a repository architectural style. The repository consists of two sets of reusable components: the *Runtime Components* and the *Development Components* which are organized in a layered architecture. The *Runtime Components* contain functionality used by an activity recognition application at runtime. An example of a runtime component is an algorithm that applies a low-pass filter to a sensor signal. The *Development Components* contain functionality commonly used to develop activity recognition algorithms, such as functionality to load a data set and to evaluate recognition algorithm using cross-validation. The applications running on the wearable devices rely on the *Runtime Components* to execute recognition algorithms. In addition, the WDK offers four tools to create, make changes to, simulate and assess the performance of activity recognition algorithms using the WDK repository.

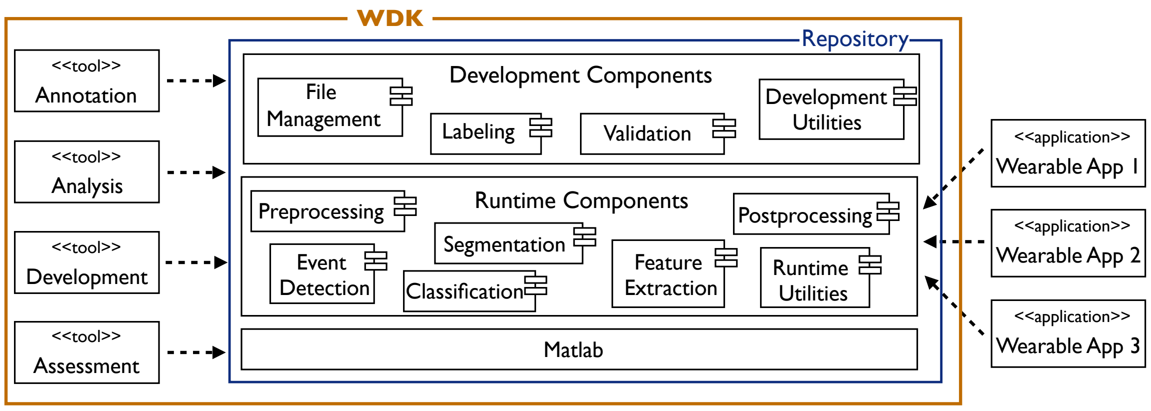


Figure 3.1: Software architecture of the WDK based on a repository organized in three layers.

The main design goal that drove our decision for a repository architecture was the *Extensibility* goal. The repository architecture decouples the reusable components from the tools, enabling the repository to be reused independently of the tools and the tools to be extended without changes to the repository. More importantly, decoupling the *Runtime Components* from the rest of the toolkit enables them to be reused within activity recognition applications running on wearable devices.

The repository itself is based on a layered architectural style with Matlab in its lowest layer. The decision to base the WDK on Matlab was mainly driven by the

Low entrance barrier goal. Matlab facilitates data analysis tasks with a broad set of functionality and native language semantics to perform arithmetic, statistical and signal processing operations with multi-dimensional arrays of data. This functionality can be used in combination with the set of reusable components in the WDK to manipulate and process data. An alternative to Matlab would have been Python in combination with third-party libraries such as *Matplotlib*, *SciPy* and *NumPy*. While we think that both languages would have been suitable alternatives, we ultimately decided for Matlab because of its proven track of usage among non-software engineers.

3.3 WDK Repository

In this section, we describe in detail the object design of the WDK's repository. We divided this section into the two layers of the repository: the *WDK Runtime Components* and the *WDK Development Components*. A full list of the reusable components in the WDK is available in our article [36].

3.3.1 WDK Runtime Components

The *WDK Runtime Component*s contain functionality commonly used in the different stages of the activity recognition chain. The functionality available for reuse in the WDK is encapsulated in subclasses of the *Algorithm* class. Analogously, every data type processed by an *Algorithm* is encapsulated in subclasses of the abstract *Data* class. Application developers typically create recognition algorithms by instantiating *Algorithm* subclasses and tailoring their properties. New algorithms can be added to the WDK by extending the *Algorithm* class and overriding its *execute()* method. Figure 3.2 shows the subclasses of the *Algorithm* and *Data* classes.

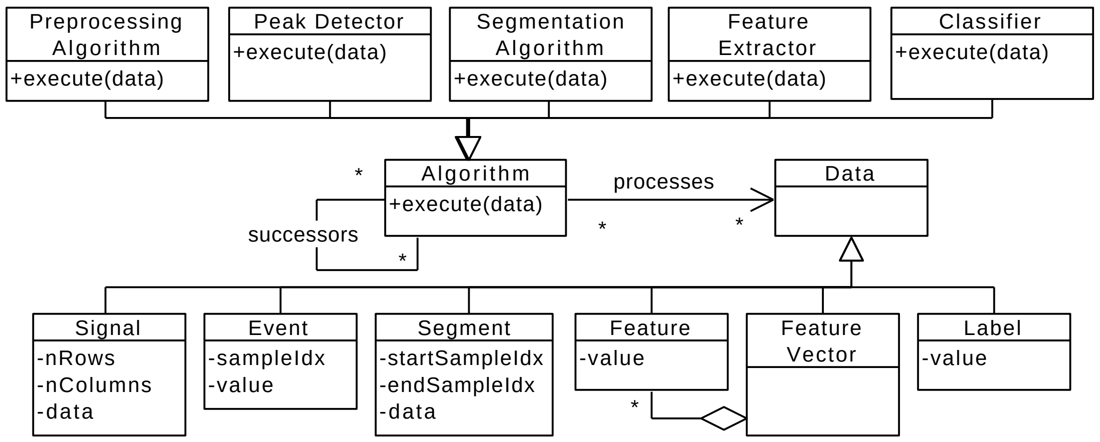


Figure 3.2: The *Algorithm* and *Data* classes are the superclasses of every other class in the WDK's repository.

A *Signal* is a two-dimensional matrix of data where the rows correspond to the time dimension and the columns correspond to different sensors. *Signals* are loaded from files and processed by *Preprocessing Algorithms*. An *Event* represents a particular sample in a signal and contains the sample's index in the *Signal* and the value of the sample. *Events* are produced by event detection algorithms (i.e. subclasses of *Peak Detector*). A *Segment* contains the data produced by a segmentation algorithm, as well as the start and end indices of the segment with respect to the *Signal* from which the segment was extracted. The *data* attribute in a *Segment* is a two-dimensional matrix of floating point values. A *Feature* is a value that describes the data in a *Segment*. An example of a *Feature* is the mean of the values in a *Segment*. *Features* are produced by subclasses of the *Feature Extractor*. A *Feature Vector* is an aggregation of instances of the *Feature* class, which are used as input to a *Classifier*. *Labels* encapsulate the values produced by a *Classifier* in an 8-bit variable.

Preprocessing

The reusable objects available in the WDK to support the preprocessing stage are shown in Figure 3.3. A *Preprocessing Algorithm* processes signals and produces another signal as output. If the *inPlaceComputation* property of a *Preprocessing Algorithm* is set to true, it's *execute* method overrides the values in the input signal and outputs the input signal. Otherwise, a *Preprocessing Algorithm* creates and outputs a new signal. Most *Preprocessing Algorithm*s receive a one-dimensional signal as input, except for the *Magnitude*, *MagnitudeSquared* and *Norm*, which require a three-dimensional signal as input.

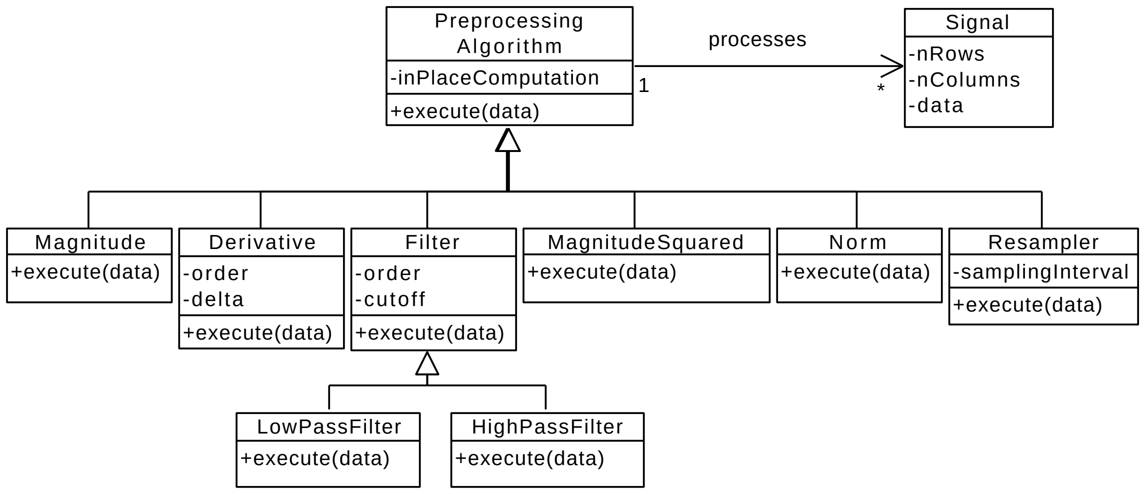


Figure 3.3: Reusable signal processing algorithms available in the WDK.

Event Detection

Some activity recognition applications need to detect the occurrence of sporadic events that are usually associated with high energies of the signals acquired by the sensors. Many activity recognition algorithms take advantage of this fact by using peak detection algorithms to detect the high-intensity events. The WDK provides two peak detection algorithms: the *Matlab Peak Detector* and the *Simple Peak Detector*. The *Matlab Peak Detector* encapsulates Matlab's functionality to detect peaks and the *Simple Peak Detector* is a greedy peak detection algorithm we adapted from [18]. Both peak detection classes inherit from the generic *Peak Detector* class, as illustrated in Figure 3.4. A *Peak Detector* takes a *Signal* as input and produces an *Event* as output.

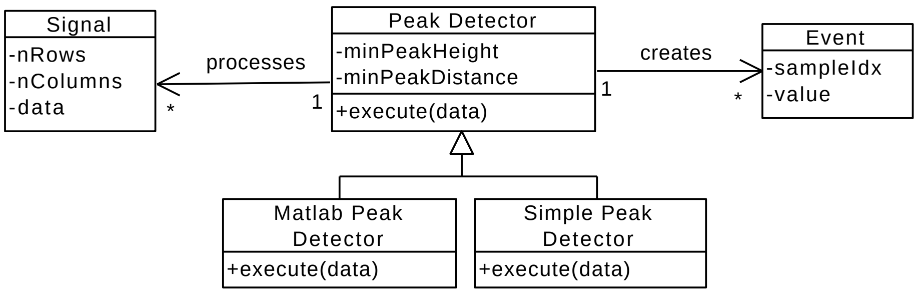


Figure 3.4: Reusable objects for event detection available in the WDK.

Segmentation

Figure 3.5 shows the classes that can be reused to segment a signal. A *Segmentation Algorithm* defines the interface for a class that creates segments from an input signal. The two subclasses of the *Segmentation Algorithm* used at runtime are the *Event Segmentation* and the *Sliding Window Segmentation.* The *Event Segmentation* creates segments around detected events in the rage: $[eventIdx - segmentSizeLeft, eventIdx + segmentSizeRight]$ where *eventIdx* is the sample index of the detected event in the input signal and *segmentSizeLeft* and *segmentSizeRight* indicate the amount of samples around the detected event to include in the generated segment. *SegmentSizeLeft* and *segmentSizeRight* are attributes of the *Event Segmentation* class that can be configured by developers.

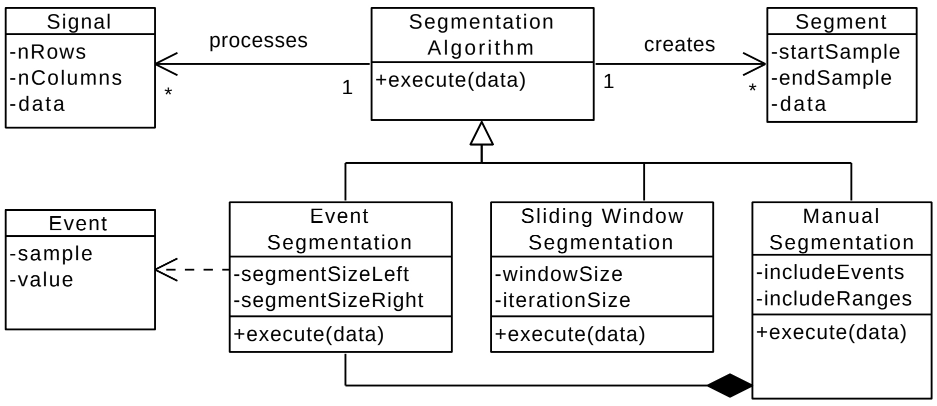


Figure 3.5: Segmentation strategies available in the WDK.

The *Sliding Window Segmentation* swipes through the entire input signal at intervals of *iterationSize* and generates a segment of size *windowSize* in each iteration.

If the value of *iterationSize* is a smaller than the *windowSize,* the generated segments overlap (i.e. the samples at the end of the segment produced in an iteration are repeated in the beginning of the segment produced in the next iteration). This segmentation strategy is suitable for cyclic activities (e.g. walking) or static postures (e.g. detecting sitting) whereas the *Event Segmentation* is suitable when sporadic activities need to be detected first (e.g. a gait stride).

The *Manual Segmentation* is a special segmentation algorithm used during development that generates segments around annotations. In particular, it converts *Range Annotations* into segments and creates segments around an *Event Annotation* using an *Event Segmentation.*

Feature Extraction

The WDK makes several common time- and frequency-domain feature extraction algorithms available for reuse. Feature extraction algorithms available in the WDK inherit from the *Feature Extractor* class, as illustrated in Figure 3.6. The *Feature Extractor* uses a *Segment* to create a *Feature.* A *Feature* encapsulates a 32-bit floating point value. *Feature Vectors* aggregate a set of extracted features and are used by the machine learning classifiers, which we discuss next.

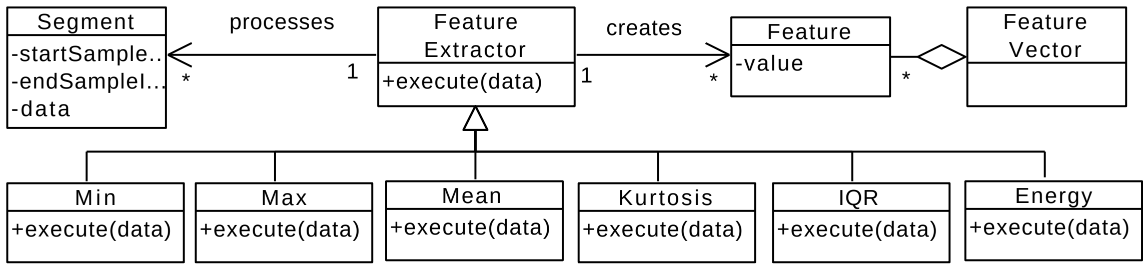


Figure 3.6: Classes available in the WDK to support the feature extraction stage. A complete list of the feature extraction algorithms available in the WDK is available in its Git repository.

Classification

Figure 3.7 shows the reusable objects offered by the WDK to support the classification stage. The superclass of every classification algorithm is the *Classifier.* A *Classifier* classifies *Feature Vector*s and produces labels as output. The *Label* object encapsulates 8-bit unsigned integer variables. The current implementation of the WDK only contains statistical pattern recognition methods. Other inference methods (e.g. template methods, syntactic methods and neural networks) can be easily added to the WDK by extending the *Classifier* class.

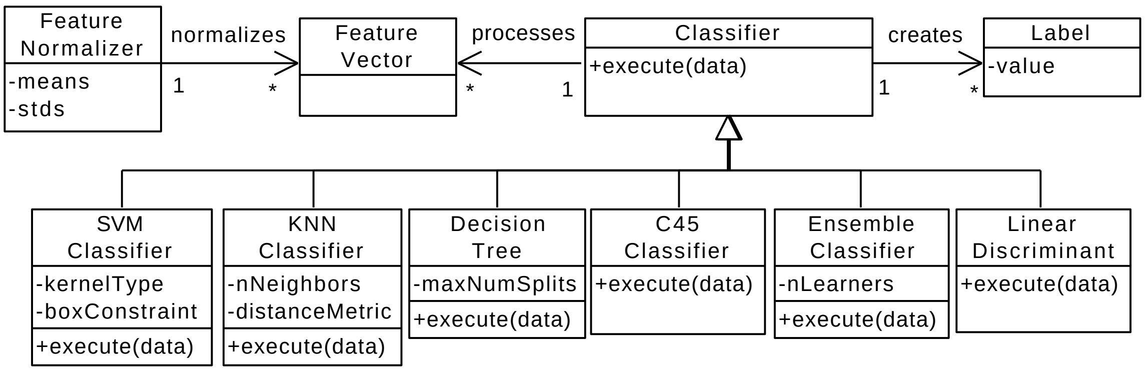


Figure 3.7: Classification algorithms offered by the WDK.

Postprocessing

Postprocessing components modify the values of the labels produced by a *Classifier* subclass. Figure 3.8 shows the reusable objects offered by the WDK to support the postprocessing of labels. The *Label Mapper* maps the labels specified in its *source-Labeling* property to the labels in its *targetLabeling* property. This can be useful to group related labels under a single category. For example, irrelevant motions included as classes in a classifier might be grouped under the same NULL-class category using this component. The *SlidingWindowMaxLabelSelector* defines a window of *window-Size* labels centered at each label in the input array. It then replaces each label with the most frequent label in the window around it or with the NULL-class if no label occurs at least *minimumCount* times within the window. This can increase the recognition performance in applications where multiple instances of an activity tend to be performed together (e.g. gait strides).

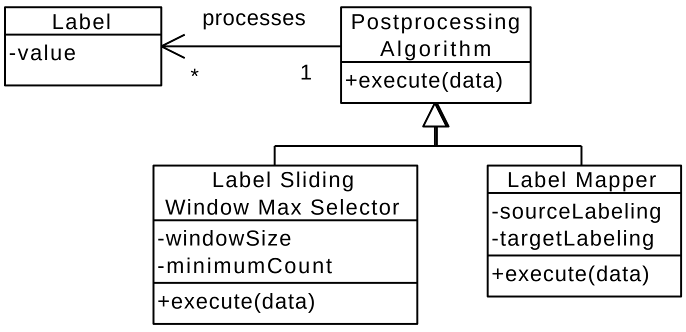


Figure 3.8: Algorithms offered by the WDK to modify the results of a *Classifier*.

3.3.2 WDK Development Components

The *WDK Development Components* layer contains functionality commonly used during the development of activity recognition applications. Figure 3.9 shows the subclasses of the *Algorithm* and *Data* classes used in this layer. In the rest of this section, we discuss each of these classes separately.

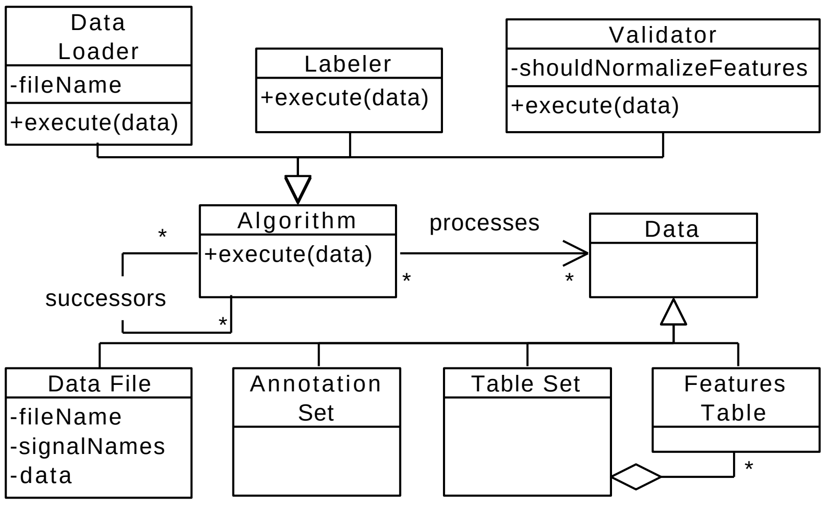


Figure 3.9: Overview of the reusable classes to develop activity recognition algorithms.

Data Acquisition

Figure 3.10 shows the reusable components in the WDK to load data files and annotations. The *File Loader* and *Annotation Loader* are *Computer* subclasses that don't take any parameter as input but produce a *Data File* and an *Annotation Set* as output, respectively. A *Data File* contains the data loaded from a file in a 2-dimensional array of floating point values and additional information about the data, including the name of the file from which the data was loaded and the names of the signals contained in the file. An *Annotation Set* contains a list of *Range Annotations* and another one of *Event Annotations*. Both annotation classes inherit from the base *Annotation* class, which encapsulates an 8-bit integer label. *Range Annotations* have a duration in time and contain the index of the start and end sample in the original signal. *Event Annotations* are sudden events represented with the index of the sensor sample in the original signal.

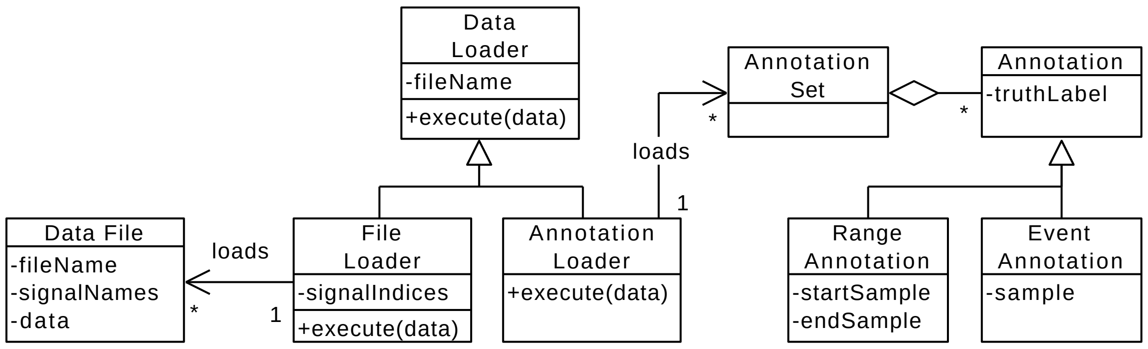


Figure 3.10: Taxonomy of annotations and reusable components to load data files and annotations.

Labeling

Labeling is the process by which the label in an *Annotation* is assigned to a segment or event. Labeled events and segments can be used during development to train a machine learning algorithm and to assess the performance of an event detection or classification algorithm. Figure 3.11 shows the classes the WDK offers to label events and segments. The *Events Labeler* labels a detected *Event* with the label of the nearest *Event Annotation* within a user-specified tolerance. The *Event Segments Labeler* labels segments that were extracted around a detected event (i.e. with the *Event Segmentation*) using an instance of the *Events Labeler* class. The *Range Segments Labeler* labels segments based on range annotations. If its *shouldContainEntireSegment* property is set to *true,* it labels each segment with the label of the *range annotation* that includes the segment entirely. Otherwise, it labels segments with the label of the *Range Annotation* that includes the sample in the middle of the segment.

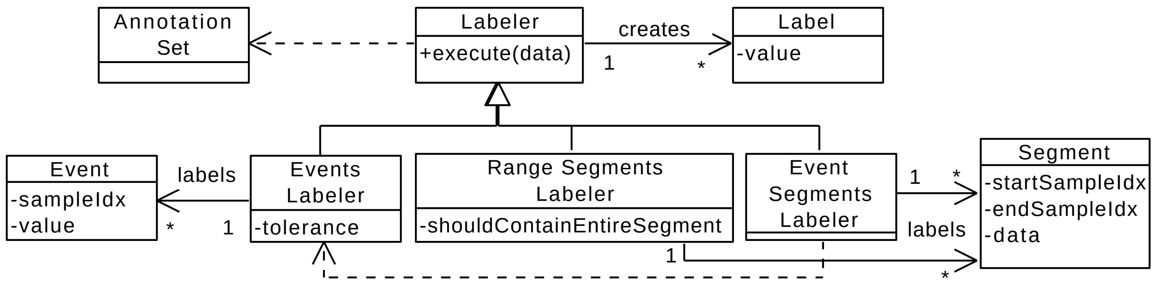


Figure 3.11: Reusable classes to label events and segments.

Validation

Figure 3.12 shows the classes the WDK makes available to assess the performance of an activity recognition algorithm. The WDK aggregates the *Feature Vectors* extracted from all of the segments of a *Data File* together with a *Label* per *Feature Vector* into a so-called *Features Table.* A *Table Set* is an aggregation of *Feature Tables.* The WDK supports two kinds of validation strategies. The *Holdout Validator* trains a classifier and predicts the labels of different subsets of *Feature Table* instances once. Which of the *Feature Tables* in a *Table Set* are used for training and for assessment is defined by the *trainIndices* and *testIndices* variables of the *Holdout Validator.* The *Leave-One-Out-Cross-Validator* trains a classifier with all but one *Feature Table*, predicts the labels of the excluded *Feature Table* and then repeats the procedure excluding a different *Feature Table.* This process is repeated until the labels of every *Feature Table* have been predicted. The *Assessment tool* in the WDK uses the predicted labels of a *Feature Set* to compute different performance metrics, as we discuss in the next section.

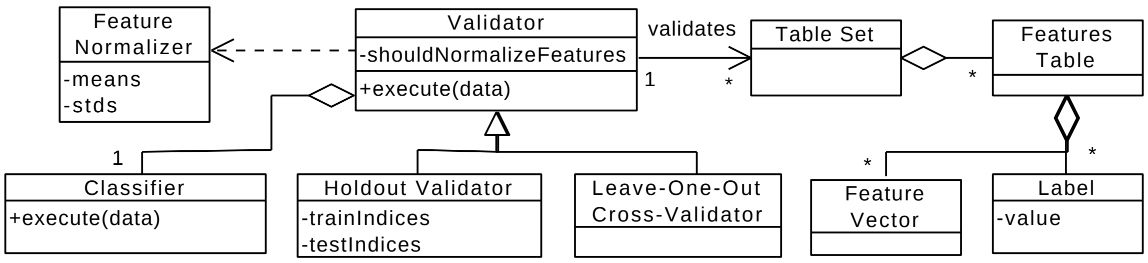


Figure 3.12: Reusable classes to assess the performance of a recognition algorithm.

3.4 WDK Tools

The WDK offers four tools to facilitate: the *annotation* of data, the *analysis* of data, the *implementation* of recognition algorithms and their performance *assessment*. We discuss each of these tools separately.

3.4.1 Data Annotation Tool

The *Data Annotation* tool provides functionality to visualize and annotate time series data from multiple sensors. The tool supports the two kinds of annotations we presented in Section 3.3.2: *Event Annotations* and *Range Annotations*. Developers introduce annotations by selecting an activity from a list and indicating a sample on the data. The tool can display video files next to the data, which can be used as a reference during the annotation of data.

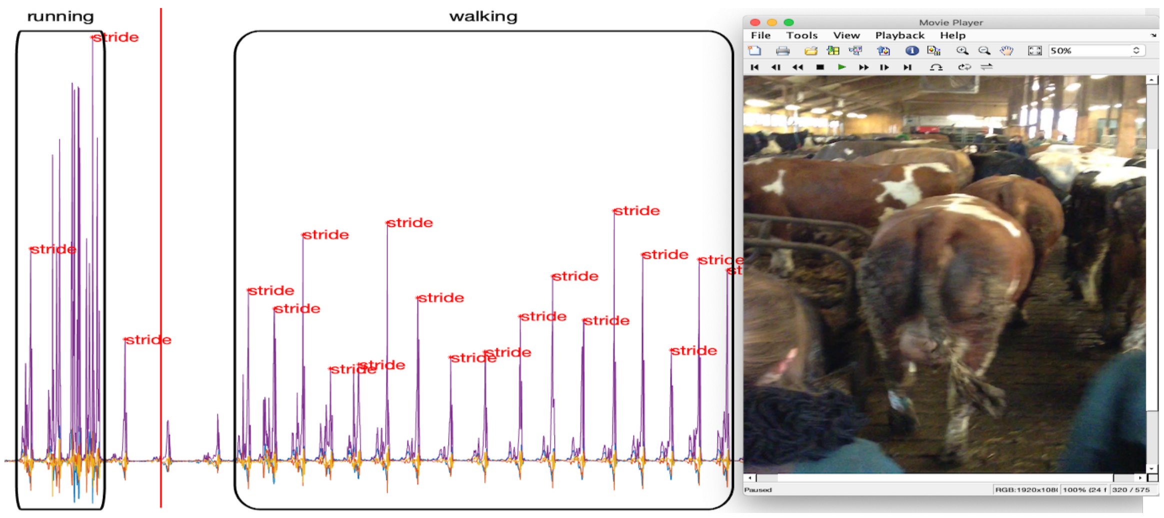


Figure 3.13: The *Data Annotation* tool displays the annotations corresponding to the gait of a cow.

The *Data Annotation* tool enables selecting and iterating through each sample of the data and frame in the video. The selected video frame and data sample are kept synchronized so that the selected data sample is updated automatically to match the currently displayed video frame and the displayed video frame is updated to match the selection of a different data sample. Figure 3.13 shows an annotated data set.

3.4.2 Data Analysis Tool

Before developers are able to implement an activity recognition algorithm, they need to study a data set containing the activities to recognize. The *Data Analysis* tool is used to study the effects of different signal processing and segmentation algorithms on the data and to devise possible feature extraction algorithms to discriminate between the activities to recognize. In particular, the *Data Analysis* enables developers to configure an activity recognition algorithm up to the segmentation stage and displays the segments produced by the algorithm grouped by activity. By visualizing the output of a segmentation algorithm, developers can study the signatures of the different activities to be recognized. By doing so, they gain an insight into which preprocessing, event detection, segmentation and feature extraction algorithms could be useful for their particular recognition problem. Figure 3.14 shows the signatures of eight different activities generated with the *Data Analysis* tool.

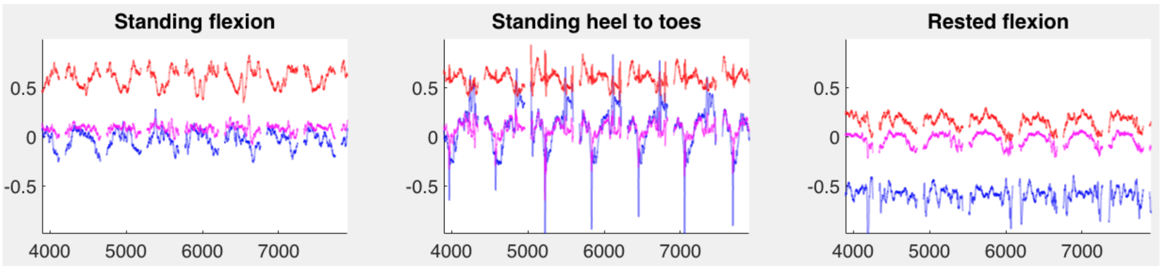


Figure 3.14: The *Data Analysis* tool shows segments that correspond to rehabilitation exercises performed by patients after a hip replacement surgery.

Furthermore, as we discussed in Section 2.1.4, the annotation of data can be challenging when different activities blur into each other without a clear boundary. By displaying segments of data generated from the annotations, developers can gain an overview of the annotations and assess their consistency. This is done in the *Data Analysis* tool by selecting the *Manual Segmentation* as segmentation strategy.

3.4.3 Algorithm Development Tool

Activity recognition algorithms can be developed in the WDK by directly reusing the components we presented in Section 3.3 within a Matlab environment. Doing so enables the reuse of the functionality available in the WDK in combination with the large set of libraries provided by Matlab. While implementing recognition algorithms within the Matlab environment provides developers with the flexibility to develop custom functionality, it requires them to be able to write source code.

To support less experienced developers, the *Algorithm Development* tool enables the development of activity recognition algorithms with a visual programming language. The tool is an extension of the Node-RED[1] visual flow-based programming platform. We have extended Node-RED by adding a Javascript implementation of every reusable component in the WDK's repository. Our extended version of Node-RED is also open-source and is available in a separate Git repository[2].

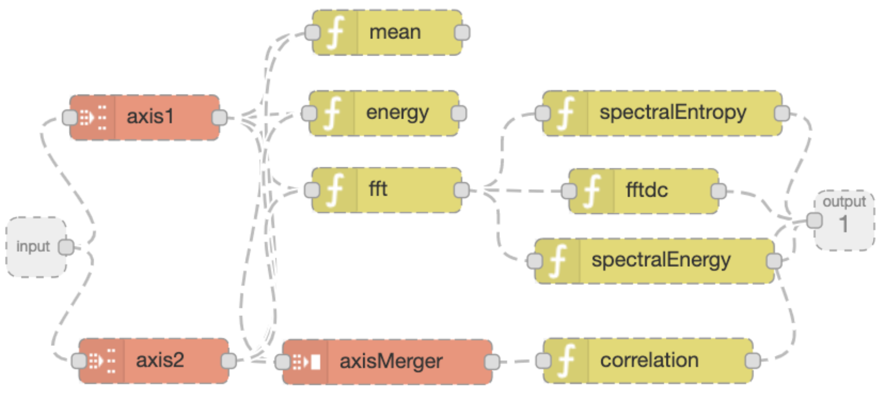


Figure 3.15: The *Algorithm Development* tool shows an algorithm to classify activities of daily living as described in [14].

3.4.4 Algorithm Assessment Tool

The *Algorithm Assessment* tool computes and displays the recognition and computational performance of an activity recognition algorithm. To compute the recognition performance of an algorithm, the tool executes the algorithm and compares the labels in the *Features Tables* generated by the algorithm against the labels in an *Annotation Set*. The recognition performance metrics computed by the *Algorithm Assessment* tool are: *accuracy*, *precision*, *recall*, *F1-Score* and *confusion matrix*. These metrics are shown per input data file and per activity. In addition, the tool displays a frame-by-frame comparison between the ground truth and the prediction of the recognition algorithm on top of the raw data used as input. This comparison is shown next to the reference video, which is automatically synchronized to the selected data sample as in the *Data Annotation* tool. The computational performance metrics computed by this tool for a recognition algorithm are: the maximum amount of memory consumed at any point by the algorithm, the total amount of floating operations performed and the amount of data output in bytes.

[1]https://nodered.org/
[2]https://github.com/avenix/WDK-RED

Chapter 4

Evaluation

This chapter demonstrates how the WDK is used to create activity recognition algorithms from three different domains: sports, daily activity monitoring and medicine. The goal of our evaluation is to demonstrate that the WDK lowers the entrance barrier to the development of activity recognition applications and that it can be used in different application domains. Section 4.1 demonstrates the ease of use of the WDK with a step-by-step walkthrough to develop an algorithm to recognize goalkeeper training exercises. Section 4.2 demonstrates how the WDK's components are used to re-create two additional activity recognition applications. The first one recognizes daily activities (e.g. walking, running, brushing teeth) using inertial sensors attached at different positions of the body. The second one uses a single inertial sensor attached at the ankle to recognize exercise repetitions performed by patients after a hip surgery.

4.1 Step-by-step Walkthrough: Goalie Glove

The first application we describe is the Goalie Glove. Goalie Glove is a goalkeeper glove with an integrated inertial sensor to help goalkeepers improve their skills. In particular, it recognizes different exercises performed by goalkeepers during a training, computes quality metrics about each exercise and provides feedback to goalkeepers over a user interface in a smartphone. In order to compute quality metrics from the training exercises, the exercises first have to be recognized in a stream of inertial sensor samples. This section demonstrates how to use the WDK to develop an algorithm to detect and classify goalkeeper training exercises using inertial sensor data.

4.1.1 Data Collection and Annotation

We started the development of this activity recognition algorithm by collecting a data set containing data from 7 goalkeepers collected during their training sessions. The data was collected at 200 Hz using an ICM20948 9-axis inertial sensor, which contains an accelerometer, gyroscope and magnetometer. The data was afterwards normalized to the range [−1, 1] by dividing each sample by the sensor's maximum ranges. On average, each training session lasted 33 minutes. We recorded the training sessions on video for annotation purposes. Figure 4.1 shows screenshots taken from the videos we collected.

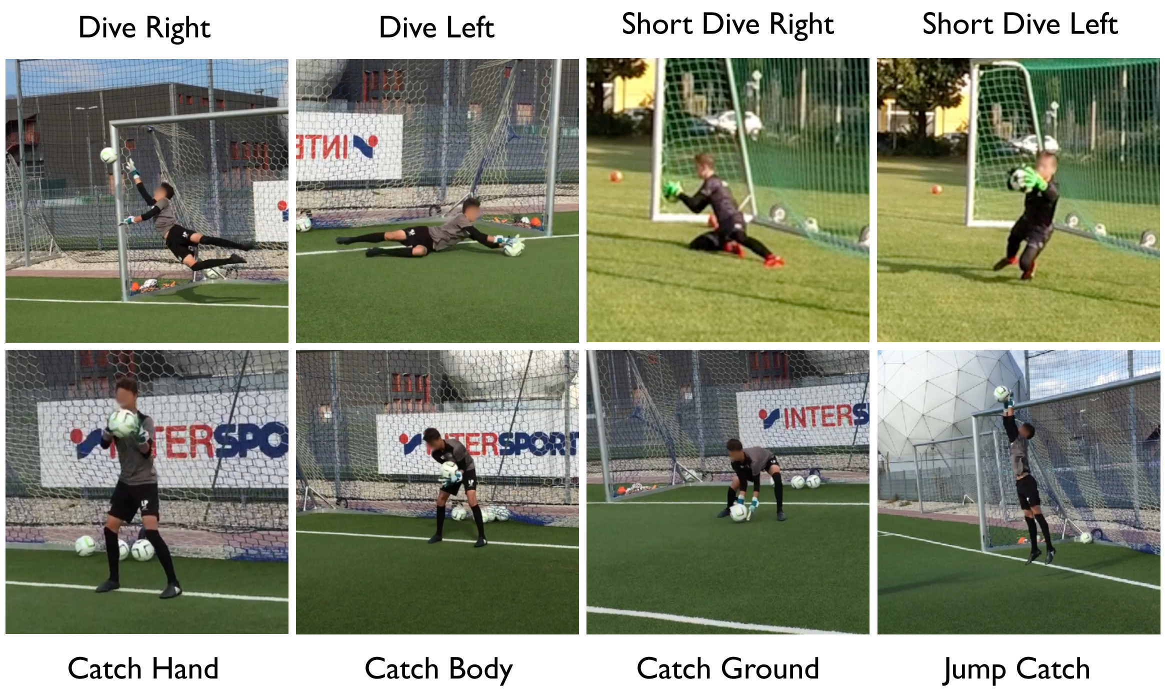


Figure 4.1: Training exercises performed by soccer goalkeepers.

As this application requires the detection of sporadic events (i.e. training exercises in a stream of inertial sensor samples), we know that our recognition algorithm will rely on an event detection method. In particular, the activities the algorithm should recognize are associated with a high energy of acceleration. Hence, we consider detecting these activities with a peak detection algorithm, as done in similar applications [18, 89, 32]. In order to be able to assess the performance of an algorithm that detects peaks of acceleration, we annotate the peaks the algorithm should detect using the *Data Annotation* tool as *Event Annotations*. In total, we annotate 18 variations of training exercises, including those shown in Figure 4.1.

We also observe that our data set contains several irrelevant motions that are likely to be detected by a peak detection algorithm, such as the motion to pass the ball to a coach. When these motions are detected, they should be recognized as false positives. In order to recognize irrelevant motions as false positives, we decide to include them as a class in a machine learning classifier. To be able to train a classifier to recognize irrelevant motions, we add the peaks associated to irrelevant motions performed frequently by goalkeepers to our annotation set. In total, we annotate 4153 *Event Annotations*, out of which 916 correspond to relevant exercise repetitions and 3237 are instances of irrelevant motions.

4.1.2 Data Analysis

We know that the peaks corresponding to training exercises can be detected with a peak detection algorithm, as discussed in the previous section. In the WDK, segments can be extracted around the *Events* detected by a peak detection algorithm with the *Event Segmentation* component. The *Event Segmentation* generates segments that contain the *segmentSizeLeft* samples to the left and the *segmentSizeRight* samples to the right of each detected event. We use the *Data Analysis* tool to decide on the values for the *segmentSizeLeft* and *segmentSizeRight* parameters. To this end, we assign both parameters to a value of 300 and compare the magnitude of acceleration of the different exercises, as shown in Figure 4.2. Based on this comparison, we observe that the characteristic motion of most exercises starts approximately 200 samples before the peak and that most exercises end shortly after it. Furthermore, the motion characteristic of some exercises (e.g. dives) might last longer than 200 samples before the peak. However, extending the segments to more than 200 samples before the peak would lead to additional memory costs and cause motion that is not characteristic of other exercises to be included in the segment. Based on these observations, we decide for parameters *segmentSizeLeft=200* and *segmentSizeRight=30.*

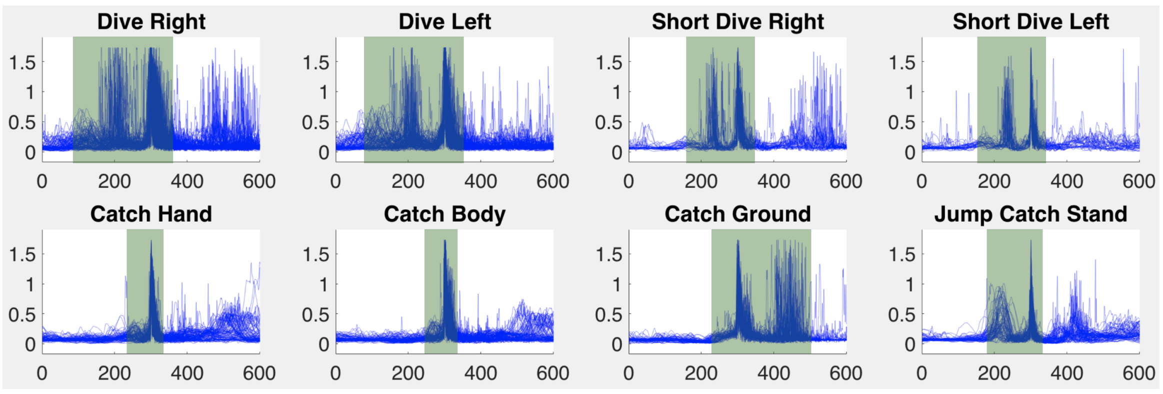


Figure 4.2: The *Data Analysis* tool displays the magnitude of acceleration of different training exercise segments plotted on top of each other. We highlighted the parts of the signal that contain motion characteristic to each exercise with a green overlay.

After having decided for a segmentation algorithm, we study what feature extraction algorithms can be used to discriminate between the different exercises. We compare the segments produced by our segmentation algorithm in the *Analysis tool*, as shown in Figure 4.3. We observe that most exercises consist of sequences of motions. For example, the dives consist of an arm swing, the actual dive and finally an impact with the ground. As these sequences of motions are individual to each exercise, they can be useful to discriminate between them. For example, the arm swing goalkeepers perform before a dive is associated to the samples in the range [1, 60]. Furthermore, the range [61, 180] can provide information to discriminate between *dives*, jump catches and other types of catches, as these exercises are associated with considerable different accelerations in this range. Finally, the range [181, 230] corresponds to the impact of a ball or with the ground, which is also different among exercises.

For example, dives are associated to more and longer accelerations in this range than the catches. Therefore, we decide to divide the segments in three sub-segments in the ranges: [1, 60], [61, 180] and [181, 230]. For each of these sub-segments, we extract different features computed on different axes of the accelerometer and magnetometer signals. In total, we extract 45 time-domain features including the *minimum*, *maximum*, *mean*, *median*, *variance*, *standard deviation* and *area under the curve.*

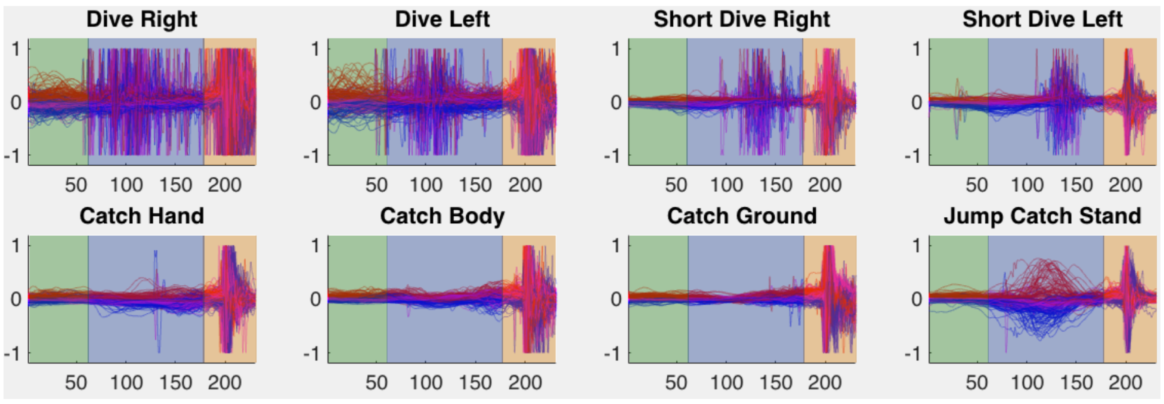


Figure 4.3: The *Data Analysis* tool shows two accelerometer signals of different exercises. We have highlighted the sub-segments [1, 60], [61, 180] and [181, 230] with a green, blue and orange overlay, respectively.

4.1.3 Algorithm Implementation

We develop the recognition algorithm shown in Listing 4.1 using the WDK components in a Matlab script. The algorithm relies on the accelerometer and magnetometer signals. First, it uses an *AxisSelector* to extract all three accelerometer axes from the input *Signal* (line 1). The resulting Nx3 *Signal* is passed to the *Magnitude* component (line 2). The *Magnitude* computes the magnitude of each accelerometer vector in the input *Signal* and passes the computed magnitude values in an Nx1 *Signal* to the *SimplePeakDetector* (line 5). The *SimplePeakDetector* detects peaks in the magnitude *Signal* and returns an array of *Events* containing the detected peaks. The *EventSegmentation* generates *Segments* around the detected peaks containing the 200 samples to the left and 30 samples to the right of each peak (line 8). The extracted *Segments* are passed to the *FeatureExtractor* which is loaded from the *features.mat* file (line 11). This feature extraction algorithm extracts the 45 features mentioned in the previous subsection for each *Segment* and outputs a *FeaturesTable*. The *FeatureNormalizer* normalizes the *FeaturesTable* such that each of its columns has zero mean and a standard deviation of one (line 15). The normalized *FeaturesTable* is passed to the *SVMClassifier* component (line 18). The *SVMClassifier* predicts a label for each of the rows in the input *FeaturesTable* and returns the array of predicted labels in a *ClassificationResult* object. Line 21 organizes the aforementioned components into a Matlab's cell array and line 22 builds an *Algorithm* that executes them in a sequence by passing the output of a component as input to its successor component. The *Al-*

gorithm Assessment tool uses the *ClassificationResults* produced by this algorithm to compute different performance metrics, as we discuss in the next subsection.

Listing 4.1: Algorithm to detect and classify soccer goalkeeper training exercises.

```
axisSelector = AxisSelector(1:3);%AX AY AZ
magnitude = Magnitude();

%minPeakHeight=0.8, minPeakDist=100
peakDetector = SimplePeakDetector(0.8,100);

%creates segments in the range: [p-200,p+30]
segmentation = EventSegmentation(200,30);

%loads algorithm shown in the right
featureExtractor = DataLoader.LoadComputer('features.mat');

featureNormalizer = FeatureNormalizer();
featureNormalizer.fit(trainTable);%compute normalization values
featureNormalizer.normalize(trainTable);%normalize training data

%order=1, boxConstraint=1.0
classifier = SVMClassifier(1,1);
classifier.train(trainTable);

components = {axisSelector, magnitude, peakDetector, segmentation,
  featureExtractor, featureNormalizer, classifier};
algorithm = Computer.ComputerWithSequence(components);
```

4.1.4 Performance Assessment

This section describes how we use the *Algorithm Assessment* tool to assess and optimize the performance of the recognition algorithm described in the previous subsection. First, we find suitable values for the *minPeakHeight* and *minPeakDistance* properties of the *SimplePeakDetector*. We know that too low values of these parameters might cause a larger amount of false positive detections and too high values might cause relevant exercises to be missed (false negatives). We test different values for these parameters and decide to configure them as: $minPeakHeight = 0.8$ and $minPeakDistance = 100$. With these parameters, the algorithm detects 93.3% of the relevant exercises but has a false positive rate of 49.1% (i.e. it detects approximately one false positive for every two relevant exercises detected). We use the frame-by-frame analysis to gain an insight into what irrelevant motions are detected by the algorithm, as shown in Figure 4.4. We observe that several motions cause false positive detections, such as the motions performed by players to pick up a ball from the ground or to pass a ball to a coach after performing a catch execise. As discussed in subsection 4.1.2, we addressed this issue by including the motions performed frequently by goalkeepers as classes in the machine learning classifier. Using an *SVMClassifier* with parameters: *order* = 1 and *boxConstraint* = 1.0, the events detected are classified with an accuracy of 81.8%, a precision of 81.4% and a recall of 79.8%.

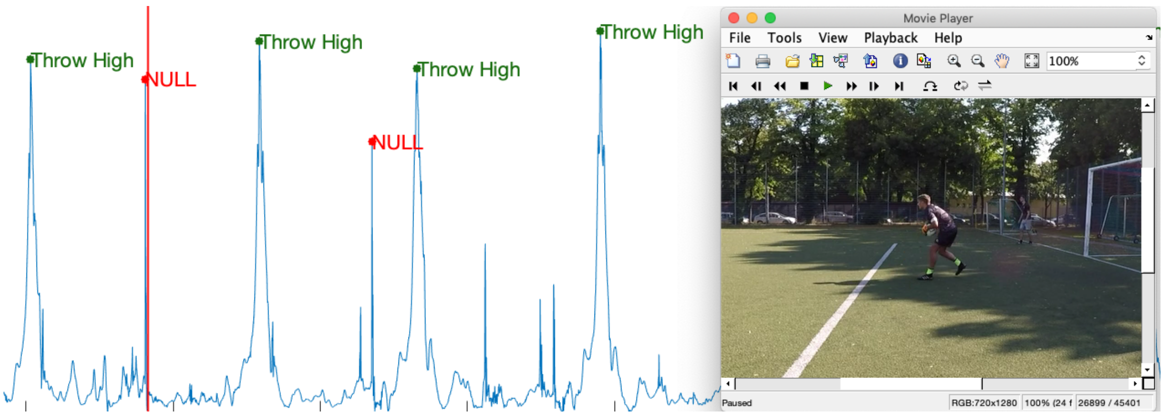


Figure 4.4: The frame-by-frame analysis displays the results of a recognition algorithm on top of the magnitude of acceleration. The algorithm detected four exercises (shown in green) and two irrelevant motions (shown in red). After this goalkeeper performs a throw, the ball is passed back at him with high intensity, which is detected (as a false positive) by the algorithm.

The *Algorithm Assessment* tool provides an overview of the computational performance of different architectures to run this algorithm. If only the segmentation was done on the wearable device, 657.7 KB of data would have to be transferred from the wearable device for an average training session. This is calculated by the WDK as an average of 244 segments per training session with a size of 230x6 values each and using 2 bytes per value. If the feature extraction was also performed on the wearable device, only 42,9 KB of data would be produced on average per training (244 feature vectors with 45 features represented with 4 bytes each). Finally, if the classification was also performed on the wearable device, only 2.1 KB of data would be generated (244 1-byte labels and an 8-byte timestamp). Furthermore, the *Algorithm Assessment* tool estimates a memory cost of 2.7 KB for the event detection, segmentation and feature extraction stages. Most of the memory required by this algorithm corresponds to the *EventSegmentation* component, which allocates a matrix of 230x6 cells of 2 bytes per value to store the produced segment. Based on this information, we decide for an architecture that performs the segmentation and feature extraction on the wearable device and transmits the extracted feature vectors for classification to a mobile device.

4.2 Reference Applications

The WDK supports the development of activity recognition applications in different domains. This section demonstrates how the reusable components in the WDK are instantiated to re-create two existing activity recognition algorithms. The first one recognizes household activities such as sitting, stretching or tooth brushing. The second one recognizes rehabilitation exercises performed by patients after a hip replacement surgery to help caregivers decide on a treatment. We use UML object models to describe how the WDK components are instantiated and the flow of execution in each recognition algorithm.

4.2.1 Reference Application 1: Daily Activity Monitoring

The first algorithm we discuss was developed by Bao and Intille [14] in 2004. It is able to recognize activities of daily living with 84% accuracy using 2-axis accelerometers placed at difference positions in the body. The algorithm computes features on windows with 512 samples of acceleration data with 50% overlapping between consecutive windows. For each window, it computes the following features: mean acceleration, sum of the squared FFT coefficients, entropy of FFT coefficients and correlation between both accelerometer axes. It relies on a C4.5 decision tree classifier to classify feature vectors.

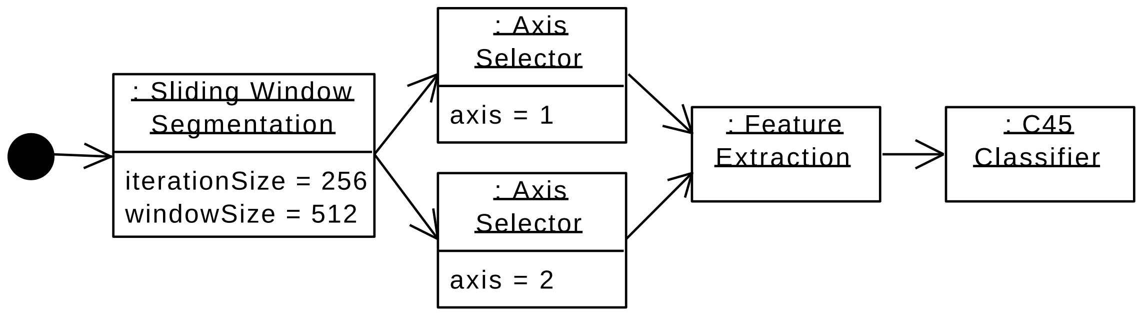


Figure 4.5: An instantiation of the WDK components to reproduce the recognition algorithm introduced in [14].

A concrete instantiation of the WDK components to reproduce this algorithm is shown in Figure 4.5. Our algorithm takes a *Signal* as input with two columns (one for each accelerometer axis) and segments it with a *SlidingWindowSegmentation*. The *SlidingWindowSegmentation* is configured to have a size of 512 samples and 50% overlapping between consecutive windows. Hence, it passes *Segments* of 512x2 samples to the two *Axis Selectors*. The *Axis Selectors* extract a single *Signal* contained within the *Segments*, which they pass to the *Feature Extraction* component. The *Feature Extraction* extracts a total of four features for each *Signal* it receives, as we describe in the next subsection. The features extracted by the *Feature Extraction* are aggregated into a *Features Table*. Finally, the *C45 Classifier* predicts a label for each row in the input *Features Table* and sets the *Features Table's label* property.

Feature Extraction

Figure 4.6 shows the reusable objects we instantiate to extract features in our recognition algorithm. The *Feature Extraction* component is a composite algorithm that contains an array of *Feature Extractor* subclasses. Its *execute* method invokes the *execute* method of each of the *Feature Extractor* subclasses it contains. In particular, it computes the *Mean*, *Spectral Entropy* and *Spectral Energy* of both accelerometer axes and the *Correlation* between the two axes. The *Spectral Entropy* and *Spectral Energy* require the prior computation of the Fourier coefficients, which is done by the *FFT* component. The *Axis Merger* is a convenience component that creates a two-column *Signal* from two single-column *Signal*s. The two-column *Signal* containing both accelerometer axes is passed to the *Correlation* object, which computes the correlation between both axes. The output of each leaf node in the feature extraction algorithm is appended to an array processed by the *Feature Extractor.*

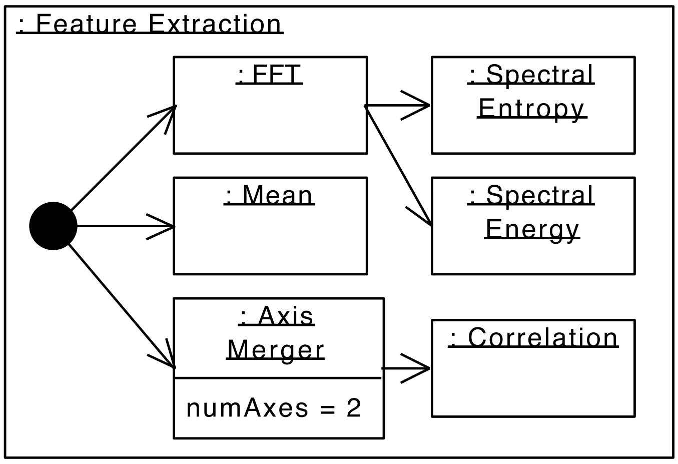


Figure 4.6: Feature extraction algorithm described in [14].

4.2.2 Reference Application 2: HipRApp

The second application we demonstrate is HipRApp (Hip, Rehabilitation App). HipRApp is a wearable strap band to track the progress of the rehabilitation of patients who underwent a hip replacement surgery. Hip replacement is a procedure in which a hip joint is removed and replaced by a prosthetic implant. The rehabilitation after a hip replacement usually starts a day after the surgery and involves walking and performing a set of physical exercises daily. Patients are usually discharged within the first week after the surgery and are advised to continue exercising for months afterwards. Thereafter, they train mostly without supervision and lack of feedback regarding the quality of their exercising and rehabilitation progress. Orthopedists meet patients at irregular time intervals (often every 3 or 4 months) and base their treatment decisions mostly on the observations they make during the visit. HipRApp estimates the rehabilitation progress of a patient of hip replacement surgery by counting the amount of exercise repetitions performed by a patient per day. In this section, we demonstrate how we developed an algorithm to recognize exercise repetitions performed by patients using a motion sensor attached to the patient's ankle.

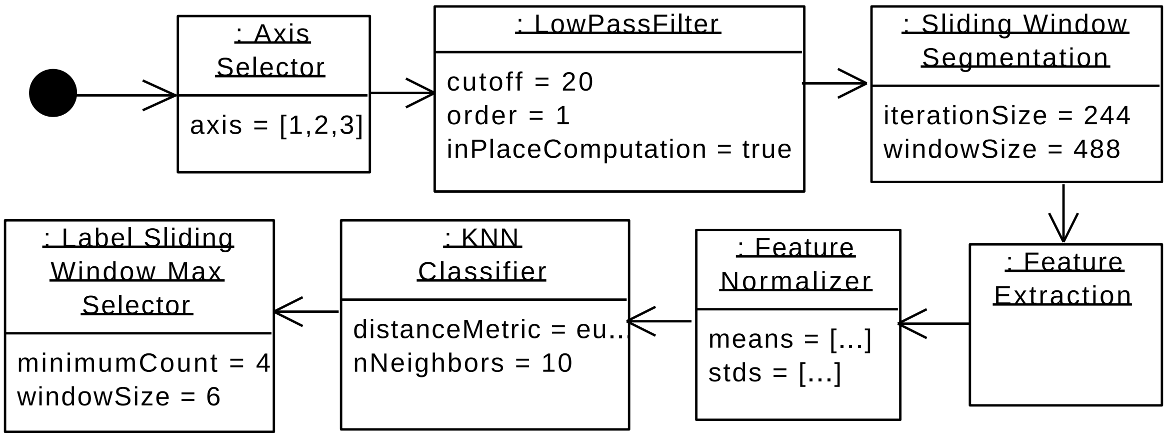


Figure 4.7: Algorithm to classify rehabilitation exercises performed by patients after a hip surgery.

Figure 4.7 illustrates the algorithm we developed to recognize rehabilitation exercise repetitions in a stream of samples produced by a 6-axis inertial sensor (accelerometer and gyroscope) worn by patients at the ankle. First, the *AxisSelector* extracts the three accelerometer signals from the input data into an Nx3 *Signal* object. The *LowPassFilter* applies a Butterworth low-pass filter to each of the *Signal*'s columns to eliminate high-frequency noise in the accelerometer signal. Then, the *SlidingWindowSegmentation* creates *Segments* of 488x6 samples that overlap by 244 samples with consecutive *Segments*. For each *Segment* produced by the *SlidingWindowSegmentation*, the *Min*, *Max*, *Mean*, *Median*, *Variance*, *STD*, *AUC*, *AAV*, *MAD*, *IQR*, *RMS*, *Skewness* and *Kurtosis* are computed. Listing 4.2 shows the code we use to create a *FeatureExtraction* instance that extracts these features on every signal from the accelerometer and gyroscope sensors. The extracted features are aggregated by the *FeatureExtraction* into a *FeaturesTable*.

Listing 4.2: Code to create a feature extraction algorithm for the HipRApp application.

```
%creates a cell array of Feature Extractor objects
featureExtractors = {Min(), Max(), Mean(), Median(), Variance(), STD(),...
   AUC(), AAV(), MAD(), IQR(), RMS(), Skewness(), Kurtosis()};

%creates a FeatureExtraction object that extracts each
%feature in the 'featureExtractors' array for each axs in the range [1,6]
featureExtraction = FeatureExtraction(featureExtractors,1:6);
```

The *FeatureNormalizer* normalizes *FeaturesTables* and passes them to the *KNNClassifier*. The *KNNClassifier* predicts a label for each row in a *FeaturesTable* and returns a *ClassificationResult* containing an array of predicted labels. Finally, the *SlidingWindowMaxLabelSelector* replaces every label at index *labelIndex* in the array of predicted labels with the most frequent label in the range [*labelIndex* − 3, *labelIndex* + 3], or with the NULL-class if no label occurs at least 4 times in the range. This is done to *'favor'* the recognition of a single exercise in a series of repetitions based on the fact that patients usually perform 10 to 20 repetitions of an exercise in a row before stopping or moving on to the next one.

Chapter 5

Conclusions and Future Work

We have presented a development environment for activity recognition applications with wearables. The reusable components and tools available in the WDK derive from our comprehensive analysis of the state of the art in the field of activity recognition with wearable sensors. The applications we chose to demonstrate the toolkit together with the different publications we reprint in Chapter 6 validate the ease of use and generalizability of the WDK to different application domains. Additional details about the WDK are available in our publication [36]. The latest version of the WDK's code can be found in its Git repository[1].

This work opens up several future research directions. First, our work addresses how to recognize activities, but not what to do with the recognized activities. However, most applications use the recognized activities in similar ways. For example, sport applications often compute performance metrics specific to each recognized activity and applications for daily activity monitoring keep track of the activities over time to compute trends, detect deviations from usual activity patterns and provide comparisons across users. A comprehensive literature review could study how the results of a recognition have been used in previous applications and derive abstractions that could be made available within a tool such as the WDK, or as an extension to the WDK.

In the last few years, there has been an increasing interest in adapting neural networks to activity recognition applications. Deep neural networks methods are of great interest because: 1) they have surpassed the manually-crafted recognition algorithms in recognition performance in large public data sets [110] and 2) they don't require developers to go through the process of studying signal processing and feature extraction methods, which is not only tedious but demands a considerable amount of experience. We decided to leave these methods out of the scope of this work because they usually require larger amounts of data than typically available when working with wearable sensors and due to their computational complexity. However, neural networks are likely to become the state of the art in activity recognition with wearable sensors eventually, as it has been the case in other research fields. To facilitate the development of recognition algorithms that rely on neural networks, we foresee a stronger need for tools to support the proper selection and augmentation of training

[1]https://github.com/avenix/WDK

data, the elimination of outlier data, the optimization of hyperparameters and the deployment of a trained model into an embedded device.

Another way to avoid having to manually develop recognition algorithms is by drawing methods from *generative design*. In generative design, different designs are generated automatically, from which the most suitable one is selected based on a user-defined optimization metric and set of constraints. The WDK offers a set of reusable components, each of which might have different properties. A generative design approach would automatically generate different recognition algorithms by combining the different components in the WDK and configuring their properties. The recognition and computational performance of each automatically generated recognition algorithm could be used as optimization metrics. In addition, developers could set constraints to the computational performance of an algorithm to limit the search space and to ensure that the computational requirements of an algorithm remain within the constraints imposed by the wearable device.

Chapter 6

Publications

This chapter contains a copy of the publications this Habilitation is based on. We first summarize the publication and then include a reprint of it. Every activity recognition application presented in this chapter has been used to refine the WDK's abstractions and elicit requirements for its tools.

6.1 The Wearables Development Toolkit: An Integrated Development Environment for Activity Recognition Applications

This publication describes the WDK in detail including its architecture, repository of reusable components, set of tools and its key features. We demonstrate the usage of the WDK with the same three wearable device applications we discussed in the Chapter 4 of this Habilitation. In addition, we present a usability study conducted with five engineers who used the WDK and gave us feedback to improve it.

The author of this Habilitation developed the WDK based on the different applications presented in this Habilitation and wrote the article.

Authors	Haladjian, J.
Journal	Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)
Number of Pages	26
Type	Journal Article
Review	Peer Reviewed (4 Reviewers)
Year	2019
DOI	

The Wearables Development Toolkit: An Integrated Development Environment for Activity Recognition Applications

JUAN HALADJIAN, Technische Universität München

Although the last two decades have seen an increasing number of activity recognition applications with wearable devices, there is still a lack of tools specifically designed to support their development. The development of activity recognition algorithms for wearable devices is particularly challenging because of the several requirements that have to be met simultaneously (e.g., low energy consumption, small and lightweight, accurate recognition). Activity recognition applications are usually developed in a series of iterations to annotate sensor data and to analyze, develop and assess the performance of a recognition algorithm. This paper presents the Wearables Development Toolkit, an Integrated Development Environment designed to lower the entrance barrier to the development of activity recognition applications with wearables. It specifically focuses on activity recognition using on-body inertial sensors. The toolkit offers a repository of high-level reusable components and a set of tools with functionality to annotate data, to analyze and develop activity recognition algorithms and to assess their recognition and computational performance. We demonstrate the versatility of the toolkit with three applications and describe how we developed it incrementally based on two user studies.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

Additional Key Words and Phrases: Human Activity Recognition, Wearables, Toolkit, Flow-based programming, Development Environment, Machine Learning

ACM Reference Format:
Juan Haladjian. 2019. The Wearables Development Toolkit: An Integrated Development Environment for Activity Recognition Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4, Article 134 (December 2019), 26 pages. https://doi.org/10.1145/3369813

1 INTRODUCTION

Over the last two decades, a number of activity recognition applications based on wearable sensors have been introduced, mostly by the research community. Applications areas include *sports* (e.g., table tennis [6], soccer [61], cricket [32]), *health* (e.g., gait analysis of patients of Parkinson's Disease [46], rehabilitation after knee injuries [25]), *daily activity monitoring* (e.g., drinking [52], eating [1], fall detection [8]) and *animal welfare* (e.g., lameness detection in dairy cattle [24], horse jump and gait classification [13]). These applications can help assess, keep track of and improve the physical condition of the wearer unobtrusively, often with minimum setup and independently of the wearer's location.

While the existing applications have already highlighted the potential benefits of activity recognition to different end user groups, developing activity recognition systems that are ultimately accepted by end users remains a challenging task, for several reasons. First, in contrast to other recognition applications (e.g., computer vision, speech recognition), activity recognition applications with wearable devices are bound to additional requirements besides a highly accurate recognition. Common requirements include: low energy consumption

Author's address: Juan Haladjian, juan.haladjian@cs.tum.edu, Technische Universität München, Boltzmanstr. 3, Munich, Germany, 85748.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2474-9567/2019/12-ART134 $15.00
https://doi.org/10.1145/3369813

(i.e., long-lasting battery), small and lightweight device and user comfort (e.g., form-factor, does not heat up) [15]. Second, the design space of a wearable device application is large. Design decisions have to be made regarding the device itself (CPU, memory, sensors, communication and storage modules), the computations that will be run on the device (sensor configurations, signal processing and machine learning methods) and the architecture of the wearable system (e.g., hardware-software mapping involving the wearable, mobile devices, the cloud and the communication between devices). As a consequence, it is often not possible to find a design that meets every requirement, in which case a suitable trade-off between design alternatives has to be made. For example, a particular recognition algorithm might deliver a higher accuracy, but might drain the battery faster than another, less accurate recognition algorithm. Hosting a larger battery could make the device remain functional for a longer period of time, but will usually also increase its size and weight, which might affect user comfort. Furthermore, the entrance barrier to wearable device development remains high, as knowledge in multiple disciplines (e.g., computer science, data science, electrical engineering, human-computer interaction) is often necessary to design wearable systems that meet the user needs.

Due to the aforementioned challenges, developers can rarely make every decision regarding the design of a wearable system upfront. Instead, they usually engage in a series of iterations to assess different design alternatives before they can decide for a suitable one. In particular, they collect and annotate data, they study the collected data and devise, implement and assess different recognition methods. Based on the results of the assessment, they decide whether further iterations are needed. Further iterations might include the collection of new data, or the development, assessment and optimization of recognition methods.

While there exist Integrated Development Environments (IDEs) specifically designed to support the development of other physical devices (e.g., mobile devices), there is up to date no IDE for activity recognition applications with wearable devices. As a consequence, most developers of wearable systems still use general-purpose data analysis tools and programing languages such as Matlab, Python, WEKA and C++. However, as these tools were not designed for activity recognition applications with wearables, they do not directly support the aforementioned tasks and have a high entrance barrier.

In this paper, we present the Wearables Development Toolkit (WDK), a development environment for activity recognition applications with wearable devices. To lower the entrance barrier to the development of activity recognition applications, the WDK offers a set of reusable software components that hide the complexity of algorithms commonly used across activity recognition applications such as signal processing procedures and machine learning classifiers. For developers with less programming experience, the same components are made available within a visual flow-based programming environment. The WDK also facilitates the iterative and incremental design of wearable device systems. In particular, it enables developers to assess the suitability of a particular wearable system (recognition algorithm, hardware and architecture) to the requirements of an application. To this end, it offers four tools to support common development tasks including the annotation of sensor data, the analysis of the data produced by different algorithms, the development of a recognition algorithm and the assessment of the computational performance (CPU usage, memory consumption, amount of data transferred) of a particular wearable system design.

The rest of the paper is structured as follows. Section 2 provides an overview of other toolkits and development environments and discusses how the WDK relates to them. In Section 3, we present the WDK and describe its features, including the goals we based its design on, its architecture and the functionality it offers. Section 4 presents a step-by-step walkthrough describing how the WDK is used to create an activity recognition application. We also demonstrate the versatility of the WDK to re-create two further activity recognition applications in Section 5. In Section 6, we present the results of two user studies we conducted to assess and improve the usability of the WDK.

2 RELATED WORK

Several toolkits have been created that support the development of interactive, ubiquitous and wearable devices and their applications. This section first provides an overview of the toolkits developed so far and then discusses different development methods existing toolkits have relied on to lower the entrance barrier and reduce the time needed to develop applications.

2.1 Toolkits

The toolkits developed so far can be grouped by the kind of applications they support. These include:

- Toolkits that facilitate the 3D-scanning, computer-aided design and 3D printing of objects as well as the integration of electronics into them. These toolkits often offer high-level programming semantics to develop applications that interact with the created objects. Toolkits that fall into this category include: Pineal [34], Retrofab [48], Makers Marks [51], Sauron [50], Modkit [40].
- Toolkits that support the development of applications that sense information from a physical environment (e.g., room temperature) and/or users in the environment (e.g., their posture) and enable joint interactions between them. Some of the toolkits under this category are: EagleSense [58], Physikit [29], Sod-toolkit [54].
- Toolkits that enable the creation of applications distributed across multiple wearable, mobile or ubiquitous devices. These toolkits offer programming semantics that span across multiple devices; hence, they save users from having to program each device as well as the communication protocols between them. Toolkits under this category include: Interactex [20], Panelrama [59], XDStudio [42], Weave [9], WatchConnect [30], iStuff Mobile [3], ToyVision [39] and C4 [33].
- Toolkits that lower the entrance barrier to the development of applications that rely on specific hardware technologies such as: smart textiles, [20], printed circuit boards [56], electrical muscle stimulation devices [47], capacitive sensors [19] and paper-based electronics [49]. These toolkits offer a set of reusable hardware and software components with high-level programming semantics that hide low-level implementation details about the particular technology.

Most of these toolkits were not developed for activity recognition applications with wearables; hence, they target different kinds of applications than the WDK. A more related class of toolkits corresponds to those designed to lower the entrance barrier to the development of applications that react to user gestures. Toolkits under this category include: Exemplar [27], MAGIC [2], GART [38] and (GT^{2}k) [57]. These toolkits are similar to our work in that they facilitate the creation of applications that detect specific patterns in sensor data. However, they focus on gesture recognition and offer predefined recognition methods for this purpose: Exemplar [27] uses Dynamic Time Warping (DTW), the MAGIC toolkit [2] relies on DTW together with a set of predefined features extracted from the input data, the (GT^{2}k) [57] uses a Hidden Markov Model (HMM) configured with a grammar specified by the user and the GART toolkit [38] relies only on a HMM. The WDK makes a broader set of recognition methods available to enable developers to experiment and ultimately design a recognition algorithm that fulfills the requirements of the particular application.

The CRN Toolbox [4] and the more recent Gesture Recognition Toolkit (GRT) developed by Nick Gillian [16] are perhaps the existing toolkits which share most similarity with the WDK. Both toolkits enable the development of recognition algorithms with a set of reusable software components. While these toolkits ease the implementation (i.e., programming) and assessment of an activity recognition algorithm, they do not support the rest of the development lifecycle of a recognition algorithm. For example, these tools don't facilitate the annotation of data, its analysis and don't provide a detailed assessment of the performance of an algorithm besides an aggregated metric (e.g., F1-Score). This is an issue because developers rarely know upfront what algorithm to implement without annotating and studying the data, developing, assessing and optimizing different recognition algorithms.

The WDK consists of different tools integrated within a development environment to support these tasks and facilitate the iterative development of activity recognition algorithms for wearables.

2.2 Programming Semantics

The existing toolkits lower the entrance barrier to users with high-level programming semantics. We have identified four main programming paradigms used by most toolkits. In *programming by demonstration*, the toolkit learns from demonstrations performed by users. Since this technique saves users from having to write code, it has been used in several toolkits from the human-computer interaction community such as: a CAPpella [11], Exemplar [27], Topiary [37], d.tools [28] and PaperPulse [49]. The ease to program an application using this technique often comes at the cost of a lack of flexibility to define custom behaviors and performance limitations. Since this paradigm relies on predefined recognition methods, users can often not optimize the recognition algorithms to their applications.

In *rule-based programming*, the user defines which predefined behaviors should be executed upon the occurrence of predefined events. Depending on the variety of events and behaviors available in the toolkit, relatively complex programs can be created with this technique by *connecting* events to behaviors that might themselves trigger other events. Toolkits that rely on this method include: Phidgets [17], Calder [36], Intuino [55], Amarino [31].

Flow-based programming is a popular visual programming approach where functionality commonly used in a particular domain is modularized in so-called *nodes*. Nodes are visual representations that encapsulate functionality and can be manipulated over a graphical user interface. A flow-based program looks like a directed graph; users draw arrows between nodes to define the order of execution of the nodes as well as the flow of data between them. Several toolkits have taken advantage of this programming technique in the past, including iStuff Mobile [3], the CRN Toolbox [4] and Interactex [20].

In *block-based programming*, different programming constructs (for loops, if-conditions, variables) are represented visually in the form of blocks that can be otherwise used as in conventional programs. While the visual representations of blocks make it easier to understand the syntax of a program (e.g., to understand the scope of a for loop), users still need to be able to create programs using conventional programming constructs. Toolkits that feature *block-based programming* include: Modkit [40] and AppInventor[1].

Previously developed toolkits have also relied on *text-based programming* approaches, including scripting and domain-specific languages. For example, the Weave toolkit [9] provides high-level APIs in Javascript for rapid prototyping wearable device applications and C4 [33] is a script language with APIs to manipulate and animate media objects such as images and movies in mobile device applications.

Since the *programming by demonstration* approach hides the recognition algorithm from developers, it also takes away the opportunity for developers to optimize it, which we considered important due to the limited computational resources available in a wearable device. Hence, we decided against it. We also thought that a *rule-based programming* paradigm would not provide developers enough freedom to create and optimize activity recognition algorithms. Furthermore, we considered that a *block-based programming* paradigm would make sense for relatively simple programs developed for educational purposes, but not for activity recognition applications. Since activity recognition applications often rely on similar functionality (signal processing and feature extraction algorithms), we opted to encapsulate this functionality under a uniform interface and made it available for reuse within a flow-based programming environment. However, we thought that a visual programming approach alone would be prone to scalability issues when developing complex recognition algorithms with several feature extraction methods. Therefore, we decided to offer the same functionality within a text-based programming language.

[1]http://appinventor.mit.edu

3 THE WEARABLES DEVELOPMENT TOOLKIT

The WDK consists of a library of reusable software components and a set of tools built on top of them. This section first discusses the goals we aimed for in the design of the WDK and then describes the reusable components, tools and main features in the WDK. The WDK is implemented in Matlab and is open source[2] under MIT license.

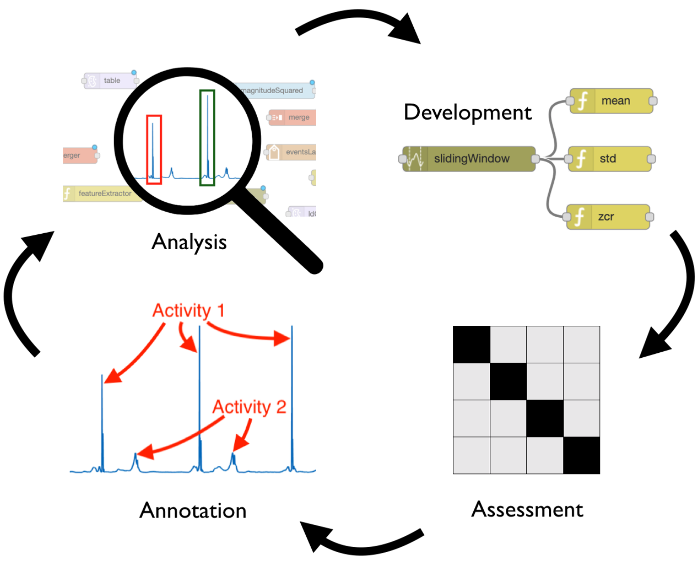


Fig. 1. Typical development lifecycle of an activity recognition algorithm. Developers usually engage in a series of iterations to collect and annotate a data set, study the collected data and then develop one or more recognition algorithms, assess and optimize their performance until the requirements of the application are met.

3.1 Design Goals

We designed the WDK based on the following design goals:

Low entrance barrier. The development of a wearable system requires knowledge from multiple disciplines including data analysis, signal processing, pattern recognition and embedded firmware development. Hence, their entrance barrier is still high. However, many wearable systems rely on similar functionality (e.g., feature extraction methods) and are developed in a similar way, as illustrated by Figure 1. A main goal of the WDK is to provide a simple way for developers to reuse common functionality as well as to ease the development tasks.

Extensibility. Even if the most common functionality used across activity recognition applications was made available for reuse within a toolkit, developers are likely to need new functionality for their particular applications, such as custom feature extraction algorithms. For this reason, one goal in the design of the WDK was to enable its set of available functionality to be extensible by developers with little effort.

Assessment of the computational requirements. Activity recognition applications are usually constrained by the computational capabilities of the wearable device. In order to study the suitability of a recognition algorithm to a particular wearable device, developers need to assess its computational requirements (CPU speed, memory capacity, battery duration). As data analysis tools used to develop wearable device applications not always provide an insight into the computational requirements of an algorithm, these are often estimated once the algorithm is

[2]https://github.com/avenix/WDK

ported to the target device. A goal in the design of the WDK was to aid developers with an early estimation of the computational requirements of a recognition algorithm.

Recognition insight. Most activity recognition applications with wearable sensors extract patterns in a stream of sensor data using machine learning classifiers. While existing tools provide aggregated metrics describing the performance of a classifier (e.g., accuracy, F1-Score) they don't provide further insight to aid developers find issues in the recognition algorithm. A goal we pursued in the design of the WDK was to enable developers to spot issues in a recognition with a frame-by-frame comparison between the ground truth and the recognition results.

Quick assessment. The training and performance assessment of a recognition algorithm is usually a computationally intensive task. As a consequence, the iterative process to develop, optimize and assess the performance of an activity recognition algorithm could be hindered by long algorithm execution times. Therefore, in the design of the WDK we aimed for solutions to quickly execute and assess recognition algorithms.

3.2 Architecture

Figure 2 illustrates the architecture of the WDK. The WDK is based on a repository architectural style. The repository consists of a set of reusable components organized as a layered architecture on top of the Matlab runtime environment. The middle layer of the repository contains the *runtime components*, a set of procedures executed by activity recognition algorithms, whereas the top layer contains functionality to facilitate the development of such algorithms. The different tools in the WDK create, make changes to, simulate and assess the performance of activity recognition algorithms using the abstractions in the repository. Applications running on wearable devices rely on the *runtime components* to execute the activity recognition algorithms created with the WDK.

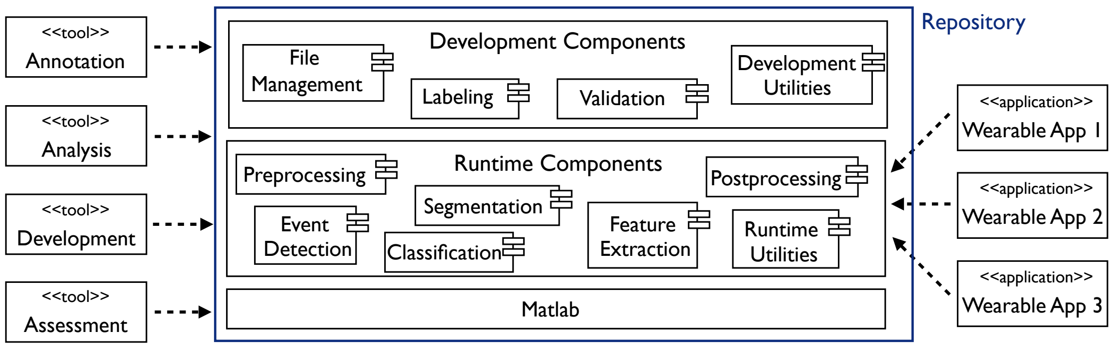


Fig. 2. The WDK is based on a repository architecture. The different tools in the WDK use the repository to create activity recognition algorithms which are executed by applications running on a wearable device.

The main design goal that drove our decision for this architecture was the *extensibility* goal. The repository architecture decouples the reusable components from the tools, enabling the components to be reused independently of the tools and the tools to be extended without changes to the reusable components. It also decouples the different tools from each other, as they interact only indirectly through the repository. This facilitates extending each tool without affecting the other tools. In addition, decoupling the *runtime components* from the rest of the toolkit eases their reuse by the wearable applications.

The decision to base the WDK on Matlab was mainly driven by the *low entrance barrier* goal. Matlab facilitates data analysis tasks with a broad set of functionality and native language semantics to perform arithmetic, statistical and signal processing operations on multi-dimensional arrays of data. This functionality can be used in combination with the set of reusable components in the WDK to manipulate and process data. Another alternative would have been Python in combination with third-party libraries such as *NumPy*, *Matplotlib*, *TensorFlow* and *Keras*.

3.3 Reusable Components

Other toolkits have lowered the entrance barrier to the development of different applications by hiding implementation details behind high-level components. Similarly, the WDK provides a set of high-level reusable components with functionality commonly used across activity recognition applications. To reuse a component, developers don't have to understand its implementation, but only what it does and what data types it requires and produces.

	Component Type	Input	Output	Used to...
Runtime	Preprocessing	*Signal*	*Signal*	transform a *Signal* and prepare it for further processing
	Event Detection	*Signal*	*Events*	detect the occurrence of specific events (e.g., peaks) in a *Signal*
	Segmentation	*Signal* / *Events*	*Segments*	divide a *Signal* into regions of interest
	Feature Extraction	*Segments*	*FeaturesTable*	compute time or frequency-domain features of a *Signal*
	Classification	*FeaturesTable*	*ClassificationResult*	predict a label for each feature vector in a *FeaturesTable*
	Postprocessing	*ClassificationResult*	*ClassificationResult*	add, remove or alter labels in a sequence of predicted labels
	Utilities	multiple	multiple	split, merge or transform data (e.g., extract values from a *Signal*)
Development	File Management	N/A	multiple	load and parse a data file or an annotations file
	Labeling	*Segments*	*Segments*	assign *Labels* to *Events* or *Segments* using an annotations file
	Validation	*FeaturesTable*	*ClassificationResult*	train and evaluate a machine learning classifier
	Utilities	multiple	multiple	different functions used at development time (e.g., feature selection)

Table 1. Summary of the functionality in the WDK's repository. *Signal*s are two-dimensional arrays of floating-point values. *Event*s represent a specific sample in a *Signal* and store an integer timestamp and a floating-point value. *Segment*s represent a range of samples in a *Signal* and contain a two-dimensional array of floating-point values and the start and end indices in the original *Signal*. *FeaturesTable*s are two-dimensional arrays of floating-point features and an 8-bit integer *label* column. *ClassificationResult*s are arrays of 8-bit integer labels predicted by a machine learning classifier.

Table 1 provides a summary of the reusable components in the WDK and the data types they take as input and produce as output. The *runtime components* encapsulate methods for each *stage* of the Activity Recognition Chain [7], including: preprocessing, event detection, segmentation, feature extraction, classification and post-processing. The *development components* offer functionality needed to manipulate the data used by the *runtime components*, label the segments produced by a segmentation algorithm and validate machine learning classifiers. Most of the preprocessing, feature extraction, classification and validation algorithms are standard off-the-shelf methods commonly used in activity recognition and offered by Matlab. In contrast, most of the event detection, segmentation, labeling and postprocessing components are our own implementations of less common algorithms described in different scientific papers [6, 7, 10, 14, 45] or derived from our own previous work. A full list of the components offered by the WDK until the date of submission of this article is available in the Appendix.

The set of reusable components is designed as a modular object-oriented architecture based on a pipes and filter architectural style. The reusable components act as filters: they receive data, process it and pass it over to other components. Developers create recognition algorithms by instantiating components and connecting them together. An algorithm is represented as a directed graph and executed with a stack in a depth-first order. To extend the functionality available in the repository, developers only have to subclass the *Computer* class and implement its *compute* method.

3.4 Tools

The WDK offers four tools to support the main tasks in the development lifecycle of an activity recognition application: the *Annotation* tool is used to annotate a time series data set, the *Analysis* tool provides a way to study the data and segments produced by a segmentation algorithm, the *Development* tool enables the creation of activity recognition algorithms with the set of components and the *Assessment* tool is used to evaluate the runtime performance of a recognition algorithm.

The *Annotation* tool is used to add annotations to a multi-dimensional time series signal. The tool supports two kinds of annotations: *event annotations* and *range annotations*. *Event annotations* correspond to events that occur at specific moments in time (i.e., a single timestamp) and *range annotations* correspond to activities that have a duration in time (i.e., two timestamps indicating start and an end of the activity). Both annotation types can be used simultaneously.

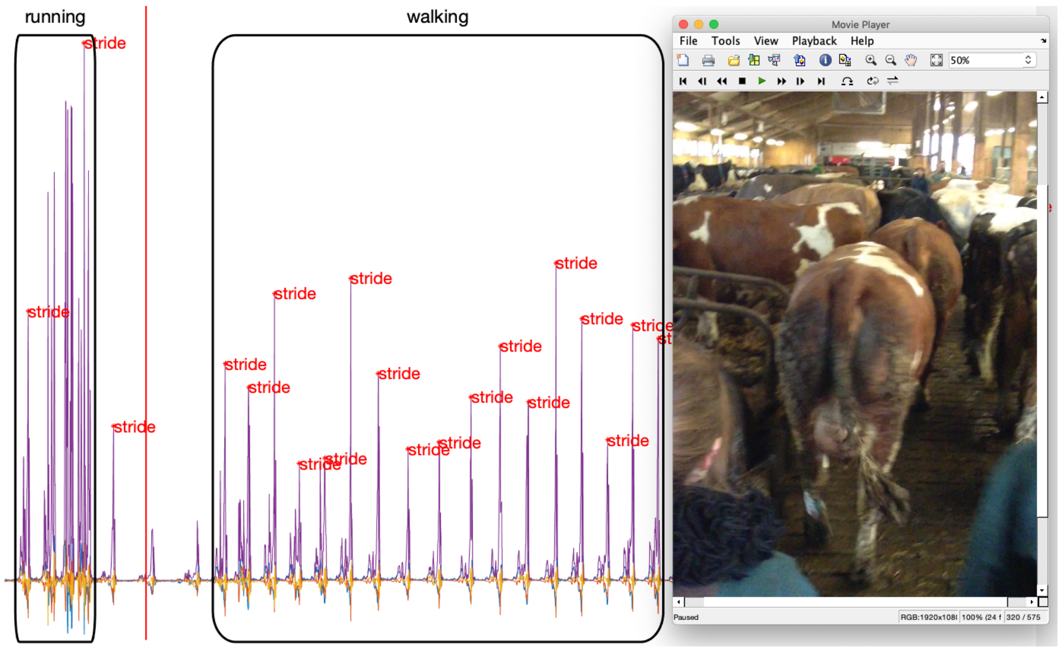


Fig. 3. The *Annotation* tool displays the squared magnitude of the accelerometer signal collected by a motion sensor attached to a cow. The individual strides of the cow have been annotated as *event annotations* (red) and the walking and running activities as *range annotations* (black rectangles).

Video is commonly used as a reference to annotate collected wearable sensor data. The *Annotation* tool displays video and data next to each other and automatically updates the current video frame to the current data selection and vice-versa. Users synchronize video and data once by providing two video frames and two data timestamps which correspond to the same event. In addition, external markers can be displayed on top of the data when annotations are performed in real time (i.e., during the data collection) or using external video annotation software.

The *Analysis* tool provides insight into the behavior of an activity recognition algorithm by displaying the segments produced by it. To this end, developers design an activity recognition algorithm either directly over the user interface of the *Analysis* tool or by importing it from the *Development* tool. The segments produced by the algorithm are then labeled, grouped by activity and shown to the user. A visualization strategy can display the segments next to each other, as shown in Figure 4, or on top of each other to help spot the pattern or signature of a particular activity. A particular kind of recognition algorithm generates segments from the annotations, which can be helpful to review the annotations and to gain insight into the patterns the algorithm should recognize.

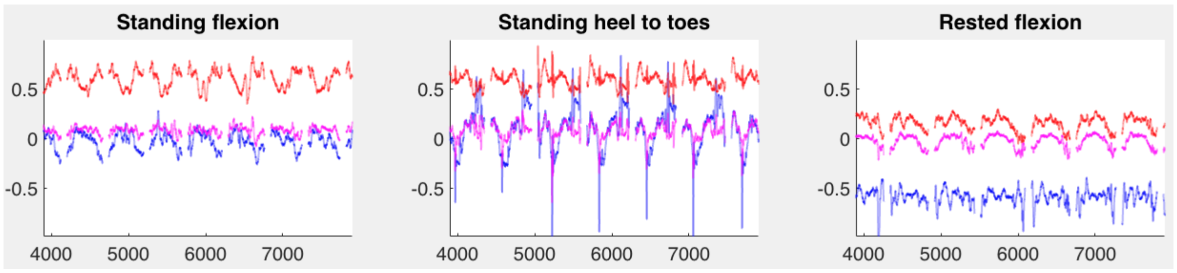


Fig. 4. The *Analysis* tool shows segments produced by a recognition algorithm corresponding to different physical rehabilitation exercises performed by patients after a hip replacement surgery.

The *Development* tool is a visual programming interface to enable less experienced users to create applications by reusing the components in the WDK. To this end, we extended Node-RED, a popular flow-based programming platform with a Javascript implementation of each reusable component. This implementation is available in a separate open source repository[3]. Algorithms created in Node-RED can be imported and executed in the different tools of the WDK. Figure 5 shows a simple activity recognition algorithm developed with the *Development* tool.

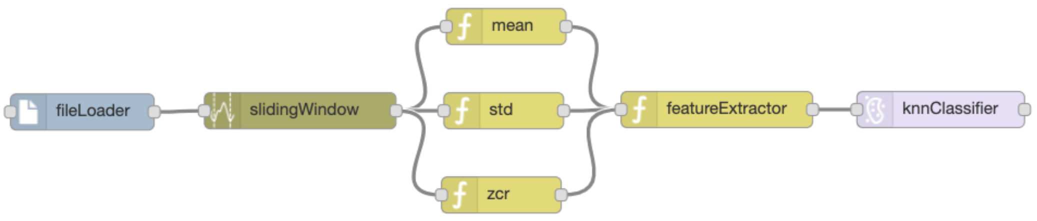


Fig. 5. Activity recognition algorithm developed in the *Development* tool. The algorithm generates consecutive segments of a one-dimensional signal using a *SlidingWindow*. For each segment, it extracts the mean, standard deviation and zero-crossing rate features. The *featureExtractor* groups the three features into a *FeaturesTable*, which is passed as input to a KNN classifier.

The *Assessment* tool enables the assessment of activity recognition algorithms regarding their *recognition* and *computational* performance. To this end, the WDK simulates the execution of an activity recognition algorithm and computes different metrics. The *recognition* performance of an algorithm is quantified by the following metrics: *accuracy*, *precision*, *recall*, *F1-Score* and *confusion matrix*. These metrics are calculated per data file and activity (i.e., class). The *computational* performance is quantified with three cost metrics: *execution*, *memory* and *communication costs*. To enable developers to compare different architectures of their wearable systems, the WDK estimates these metrics for each stage of a recognition algorithm. These metrics are averaged across data files and displayed over the user interface for each algorithm execution. Next subsection describes how the *computational* performance metrics are computed.

[3]https://github.com/avenix/WDK-RED

3.5 Computational Performance Assessment

Every reusable component in the WDK computes three computational performance metrics: *execution*, *memory* and *communication cost*. The *execution cost* is an estimation of the number of floating point operations performed by the recognition algorithm normalized by the amount of data samples provided as input. The *memory cost* is an estimation of the maximum amount of memory required to execute a recognition algorithm. *Execution* and *memory cost*s are calculated at runtime by executing a recognition algorithm. Each reusable component computes its *execution* and *memory cost*s for a provided input based on the values of its properties at runtime. The *execution cost* of an algorithm is then calculated by adding the *execution cost*s returned by each reusable component every time their *compute* method is invoked. The *memory cost* is calculated by adding the *memory cost* returned by each component in a recognition algorithm once. The *communication cost* of an algorithm is computed by adding up the amount of bytes produced by the last component in the algorithm. The *execution*, *memory* and *communication cost*s of each reusable component are listed in Section A in the Appendix.

3.6 Frame-by-frame analysis

Many activity recognition applications are evaluated with respect to time [7]. To provide further insight into the recognition performance of an algorithm with respect to time, the *Assessment* tool displays a frame-by-frame comparison between the ground truth and the classifier's prediction on top of the raw data and reference video. To this end, the WDK stores the list of labels predicted by a classification algorithm, feature vectors extracted by a feature extraction algorithm, segments generated by a segmentation algorithm and signals produced by a preprocessing algorithm. The start and end index of a segment are used to correlate predicted labels to original annotations in the ground truth.

3.7 Cache

To enable the quick assessment of a recognition algorithm, the WDK stores the execution results of a recognition algorithm in a cache under a hash-key that uniquely identifies the algorithm. This key is generated by concatenating a description of each component used in the algorithm in depth-first order. The description of a component is a string containing its name and the value assigned to each of its properties. Before executing a particular algorithm, the WDK loads its execution results, in case these are available in the cache.

4 WALKTHROUGH: GOALIEGLOVE

This section describes step-by-step how to develop an algorithm to recognize the training exercises performed by soccer goalkeepers including dives, catches and throws with an inertial sensor inserted into goalkeepers' gloves. The goal of this application is to give goalkeepers personalized feedback about their training.

4.1 Data collection

This application uses a data set collected from 7 goalkeepers during their training sessions using a sensor device based on the ICM20948 9-axis Inertial Measurement Unit. To capture the full range of motion of exercises that might contain high intensity impacts and rotations, the accelerometer, gyroscope and compass were set to their maximum ranges: ±16 g, ±2000 dps and ±4900 μT respectively, as done in similar IMU-based sports applications [6, 18, 53]. The sensor device collected data at 200 Hz and normalized it to the range $[-1, 1]$. Each training session lasted an average of 33 minutes. The training sessions were recorded on video for annotation purposes. On average, each video and data file in binary format had a size of 2.22 GB and 18.25 MB, respectively. To synchronize the data and video, goalkeepers were asked to applaud three times in front of the camera between exercise sets.

4.2 Data Annotation

This application aims at detecting sporadic events that have a high energy of motion. Previous work has detected similar events by finding peaks on the (squared) magnitude of acceleration or gyroscope signals [6, 18, 23]. In order to be able to assess the performance of a peak detection algorithm later on, we add an *event annotation* to each peak in the magnitude of acceleration that corresponds to an exercise repetition. We also annotate other motions with high accelerations performed often by goalkeepers such as ball passes and bouncing the ball on the ground. Annotating these motions will enable us to train a classifier to filter these motions out in case they are detected. The ground truth contains 4153 annotated motions, out of which 916 correspond to relevant exercise repetitions and 3237 are instances of irrelevant motions.

4.3 Analysis

Most relevant exercises have a high intensity of acceleration. We know that high intensity accelerations can be detected using a peak detector. Therefore, we use the *Analysis* tool to study the signal to determine how to create segments of data around the peaks detected by a peak detector. Figure 6 shows the data around the relevant events we annotated. We observe that the characteristic motion of most exercises starts approximately 200 samples before the peak and that most exercises end shortly after it. Furthermore, we observe that the relevant motion previous to the peak might last longer than a second (200 samples) in some exercises such as the dives. However, extending the segments to more than 200 samples before the peak would cause motion to be included in the segment that is not characteristic of most exercises. Furthermore, longer segments increase the amount of memory required by the device. Based on these observations, we decide to segment the signal according to: $[p - 200, p + 30]$.

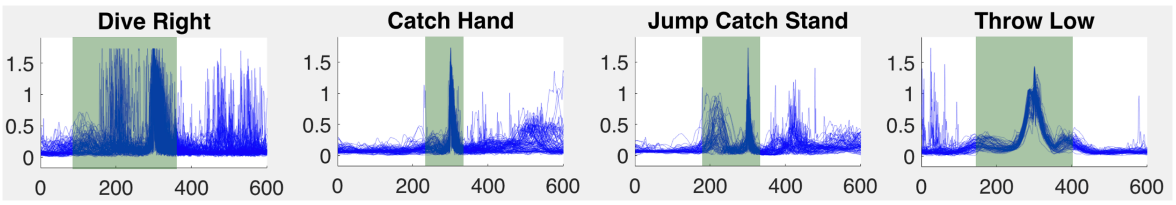


Fig. 6. The *Analysis* tool displays the magnitude of acceleration of segments corresponding to different exercises plotted on top of each other and grouped by their label. We marked the parts of the signal that contain motion characteristic of each exercise with a green overlay. We used this visualization to devise an event detection and segmentation algorithm.

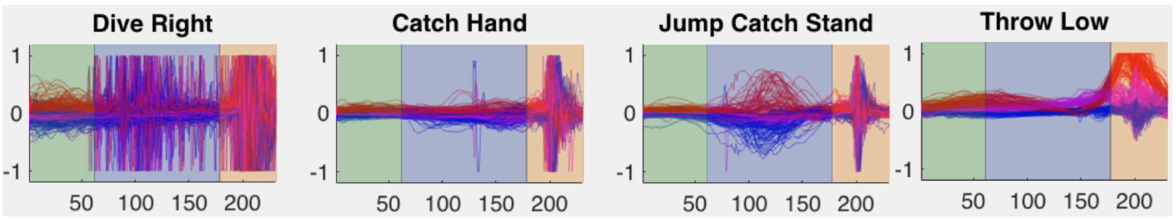


Fig. 7. The exercises performed by soccer goalkeepers can be divided in three sub-segments. The range $[181, 230]$ (orange overlay) corresponds to a ball or ground contact. The range $[61, 180]$ (blue overlay) can provide information to determine whether the exercise is a dive or a jump, as these exercises present more acceleration in this range than the other ones. The range $[1 - 60]$ (green overlay) can be used to recognize dives due to the arm swing goalkeepers perform before a dive.

To decide what features should the algorithm extract for each segment, we study the characteristic motions of the different exercises in the *Analysis* tool, as shown in Figure 7. Most exercises consist of sequences of motions. For example, the *Jump Catch Stand* consists of a jump, a ball catch in the air and a ground contact. Based on this analysis, we decide to divide segments in three sub-segments: $[1, 60]$, $[61, 180]$ and $[181, 230]$ and extract a total of 45 time-domain features including: *Min, Max, Mean, Median, Variance, STD* and *AUC* computed on different axes of the accelerometer and magnetometer signals for each sub-segment.

4.4 Development

Next, we develop the algorithm shown in Listing 1 in a Matlab script. The algorithm first selects all three accelerometer axes in the input *Signal* using the *AxisSelector* and passes the resulting Nx3 *Signal* to the *Magnitude*. The *Magnitude* computes the magnitude of each accelerometer vector in the input *Signal* and passes the computed magnitude in an Nx1 *Signal* to the *SimplePeakDetector*. The *SimplePeakDetector* detects peaks in the magnitude *Signal* and returns the *Events* of the detected peaks. The *EventSegmentation* generates *Segments* around the detected peaks by extracting the 200 samples to the left of the detected peak and 30 samples to its right. The *Segments* are passed to a *FeatureExtractor* (loaded from the *features.mat* file), which extracts the 45 features mentioned in the previous subsection for each *Segment* and outputs a *FeaturesTable*. The *FeatureNormalizer* normalizes the *FeaturesTable* so that each of its feature columns has zero mean and a standard deviation of 1 and passes it to the *SVMClassifier*. The *SVMClassifier* returns the predicted labels in a *ClassificationResult* object.

```
%computes magnitude of acceleration
axisSelector = AxisSelector(1:3);
magnitude = Magnitude();

%minPeakHeight=0.8, minPeakDist=100
peakDetector = SimplePeakDetector(0.8,100);

%segments in the range: [p-200,p+30]
segmentation = EventSegmentation(200,30);

%loads feature extraction algorithm
featureExtractor =
    DataLoader.LoadComputer('features.mat');
```

```
%computes normalization values and normalizes
featureNormalizer = FeatureNormalizer();
featureNormalizer.fit(trainTable);
featureNormalizer.normalize(trainTable);

%order=1, boxConstraint=1.0
classifier = SVMClassifier(1,1);
classifier.train(trainTable);

components = {axisSelector, magnitude, peakDetector,
    segmentation, featureExtractor,
    featureNormalizer, classifier};
algorithm =
    Computer.ComputerWithSequence(components);
```

Listing 1. Algorithm to detect and classify soccer goalkeeper training exercises. The algorithm starts at the left and continues at the right column. The *trainTable* variable in the right column has been generated with a similar sequence of computations, excluding the *featureNormalizer* and *classifier* components and using an *EventSegmentsLabeler* after the segmentation.

4.5 Performance Assessment

After having developed the algorithm, we use the *Assessment* tool to assess and optimize its recognition performance. We test different values for the properties *minPeakHeight* and *minPeakDistance* of the *SimplePeakDetector*. Low values for these parameters might cause more irrelevant motions to be detected whereas high values might cause relevant exercises to be missed. We use the frame-by-frame analysis to understand the effects of different values for these parameters on the data set, as shown in Figure 8. After this analysis, we decide for the values: $minPeakHeight = 0.8$ and $minPeakDistance = 100$. The *SVMClassifier* component configured as: ($order = 1$ and *boxConstraint* = 1.0) achieves the highest performance with an accuracy of 81.8%, a precision of 81.4% and a recall of 79.8%. Adapting the previous script to select subsets of features with the *FeatureSelector* component reveals that up to 5 features can be excluded with a minimal drop in accuracy.

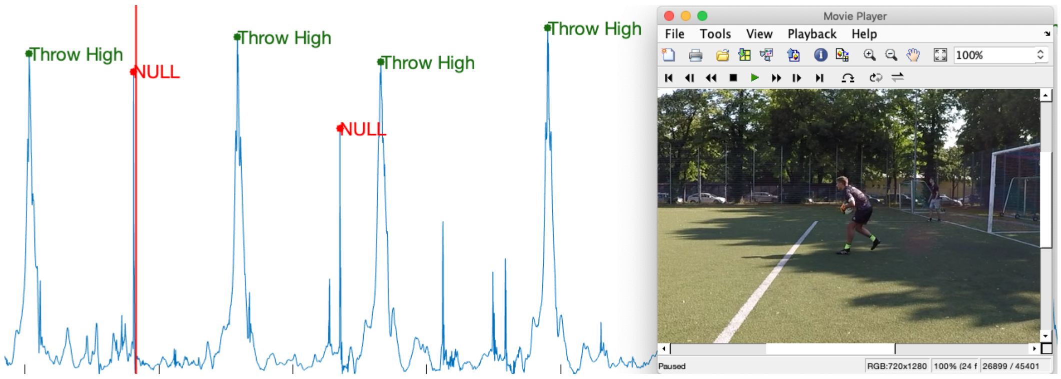


Fig. 8. The frame-by-frame analysis displays the results of a recognition algorithm on top of the magnitude of acceleration. The algorithm detected four exercises (shown in green) and two irrelevant motions (shown in red). After this goalkeeper performs a throw, the ball is passed back at him with high intensity, which is detected (as a false positive) by the algorithm.

The *Assessment* tool provides an overview of the computational performance of different architectures to run this algorithm. If only the segmentation was done on the wearable device, 657.7 KB of data would have to be transferred from the wearable device for an average training session. This is calculated by the WDK as an average of 244 segments per training session with a size of 230x6 values each and using 2 bytes per value. If the feature extraction was also performed on the wearable device, only 38,1 KB of data would be produced on average per training (244 feature vectors with 40 features represented with 4 bytes each). Finally, if the classification was also performed on the wearable device, only 2.1 KB of data would be generated (244 1-byte labels and an 8-byte timestamp). The *Assessment* tool estimates a *memory cost* of 2.7 KB for the event detection, segmentation and feature extraction stages - most of which corresponds to the *EventSegmentation* component which allocates a matrix of 230x6 cells of 2 bytes per value.

5 REFERENCE APPLICATIONS

Ledo et al. [35] proposed four types of ways to evaluate toolkits: demonstration, usage, technical performance and heuristics. Demonstration evaluations show how a toolkit is used to create applications. Usage evaluations investigate the usability of a toolkit, often by means of user studies. Technical performance evaluations assess the non-functional requirements of a toolkit such as the recognition accuracy of a created algorithm. A heuristics evaluation investigates a toolkit's usability with respect to a set of heuristics, such as Nielsen's usability heuristics [43, 44]. The previous section demonstrated the usage of the WDK with a step-by-step walkthrough to create an application. This section demonstrates the WDK's versatility to support different applications.

We created the WDK iteratively by extending and refining its abstractions to replicate different activity recognition applications from the literature and from our previous work. These applications include: an algorithm to classify daily activities presented by Bao and Intille [5], a smart bandage to track the rehabilitation progress of patients after a knee injury [21, 25, 26], a chest belt strap band to recognize basketball defensive training exercises, a lameness detection system for dairy cattle [23, 24], an activity tracker for pigs [22] and a sensor-based horse gait and jump detection system for show jumping applications [12, 13]. Next, we demonstrate how two of these applications are developed using the reusable components in the WDK.

5.1 Daily Activity Monitoring

Listing 2 replicates the activity recognition algorithm presented by Bao and Intille [5]. This algorithm recognizes physical activities (e.g., walking, sitting, eating) using two-axis accelerometers worn on different parts of the body. The algorithm processes a stream of sensor values in *Segments* of 512 samples with 50% overlapping using the *SlidingWindowSegmentation*. The *SlidingWindowSegmentation* passes *Segments* of 512x2 samples to a *FeatureExtractor*. The *FeatureExtractor* computes a feature vector for each segment it receives as input and appends it to a *FeaturesTable*. Each feature vector contains the mean, spectral entropy and spectral energy of both accelerometer axes and the correlation between the two axes. *FeaturesTables* output by the *FeatureExtractor* are passed to the *TreeClassifier*, which returns an array of labels in a *ClassificationResult*.

```
%segmentSize=512, 50% overlapping
slidingWindow =
    SlidingWindowSegmentation(512,256);

%creates feature extraction algorithm
featureExtractor =
    createFeatureExtractor();

%maxNumSplits=30
classifier = TreeClassifier(30);

%creates algorithm
algorithm =
   Computer.ComputerWithSequence({
   slidingWindow, featureExtractor,
   classifier});
```

```
function featureExtractor = createFeatureExtractor()
   fftFeatures = FFT();
   fftFeatures.addNextComputers({SpectralEntropy(),
       SpectralEnergy()});
   featureComputers = {Mean(),fftFeatures};

   %extract features on accelerometer axes x and y
   axis1 = AxisSelector(1);%x-axis
   axis2 = AxisSelector(2);%y-axis
   axis1.addNextComputers(featureComputers);
   axis2.addNextComputers(featureComputers);

   %returns feature extraction algorithm
   featureExtractor = FeatureExtractor({axis1,axis2,
       Correlation()});
end
```

Listing 2. Algorithm to classify daily activities proposed by Bao and Intille [5] reproduced with the WDK's components.

5.2 Hip Rehabilitation App

The Hip Rehabilitation App (HipRApp) is a wearable strap band to track the rehabilitation progress of patients who underwent a hip replacement surgery. It counts the amount of exercise repetitions and walking steps performed by patients during a training session. The algorithm shown in Listing 3 recognizes exercise repetitions in a stream of samples produced by a 6-axis inertial sensor (accelerometer and gyroscope) worn by patients at the ankle. First, the *AxisSelector* extracts the accelerometer axes from the input data into an Nx3 *Signal*. The *LowPassFilter* applies a Butterworth low-pass filter to each of the *Signal*'s columns to eliminate high-frequency noise in the accelerometer signal. The filtered data is processed using a sliding window. For each *Segment* produced by the *SlidingWindowSegmentation*, the *Min*, *Max*, *Mean*, *Median*, *Variance*, *STD*, *AUC*, *AAV*, *MAD*, *IQR*, *RMS*, *Skewness* and *Kurtosis* are computed. These features are extracted on every axis of the accelerometer and gyroscope *Signals* and aggregated by the *FeatureExtractor* into a *FeaturesTable*. The *FeatureNormalizer* normalizes *FeaturesTables* and passes them to the *KNNClassifier*. The *KNNClassifier* predicts a label for each row in a *FeaturesTable* and returns a *ClassificationResult* containing an array of predicted labels. Finally, the *SlidingWindowMaxLabelSelector* post-processing component replaces every label at index *labelIndex* in the array of predicted labels with the most frequent label in the range $[labelIndex - 3, labelIndex + 3]$, or with the NULL-class if no label occurs at least 4 times in the range. This is done to '*favor*' the most frequent label within a 6-label window and avoid the sporadic misclassification of unrelated exercises or instances of the NULL-class. This increases the recognition accuracy due to the fact that patients usually perform 10 to 20 repetitions of an exercise in a row.

```
%select signals 1,2,3 (accelerometer x,y,z)
axisSelector = AxisSelector(1:3);

%order=1, cutoff=20Hz
lowPassFilter = LowPassFilter(1,20);

%segmentSize=488, 50% overlapping
segmentation =
    SlidingWindowSegmentation(488,244);

%max, min, etc. on signals 1,2,3,4,5 and 6
features =
    FeatureExtractor.DefaultFeatures();
featureExtractor =
    FeatureExtractor(features,1:6);
```

```
%computes normalization values
featureNormalizer = FeatureNormalizer();

%k=10, distanceMetric='euclidean'
classifier = KNNClassifier(10,'euclidean');

%windowSize=6, minimumCount=4
postprocessor = LabelSlidingWindowMaxSelector(6,4);

%creates algorithm
components = {axisSelector, lowPassFilter,
    segmentation, featureExtractor,
    featureNormalizer, classifier, postprocessor};
algorithm =
    Computer.ComputerWithSequence(components);
```

Listing 3. Algorithm to classify rehabilitation exercises performed by patients of hip replacement. The algorithm starts in the left column and continues in the right. In a separate script, the *classifier* is trained and the *featureNormalizer* is fit with normalization values.

5.3 Discussion

The incremental development process we used to create the WDK enabled us to assess its coverage of the functionality present in a variety of applications and to refine it accordingly. The applications presented in this section demonstrate the WDK's versatility to different domains and illustrate that complex activity recognition algorithms can be created with a few components in the WDK.

6 USABILITY EVALUATION

To study the usability of the WDK, we conducted a user study with three participants who used the WDK to create different applications, as summarized in Table 2. The participants were students of computer science at the Technical University of Munich who contacted us to write a bachelor's or master's thesis at our department after they read a project description on our department's website. None of them had previous experience in activity recognition or in Matlab. They were instructed to develop an application using the WDK during a period of two to four months. After the development phase, we conducted an semi-structured interview where the participants described their experiences using the WDK and demonstrated to us how they had used it.

Participant	Gender	Application
P1	Male	GoalieGlove
P2	Male	Recognition of basketball defensive training exercises
P3	Female	HipRApp

Table 2. Participants of the first user study and applications they developed using the WDK.

All three participants found the functionality to annotate data while looking at the video useful. P3 said: *"The Annotation tool is very useful because everything is in the same place. I was using DaVinci Resolve for the annotations in the video but that was a lot of back and forth switching"*[4]. The participants also praised the functionality to compare the segments produced by an algorithm in the *Analysis* tool. In particular, they welcomed the functionality to quickly switch between signals to design feature extraction [P2,P3] and event detection algorithms [P1,P2]. P1 said: *"It's good to compare different players: what segments are too small and which ones are too big"*. Furthermore, every participant reported that they found the frame-by-frame comparison in the *Assessment* tool useful in their projects. Notably, P2 mentioned that he had been using wrongly annotated data for months until he observed a contiguous sequence of misclassified exercises in the frame-by-frame comparison. He described the insights he gained as: *"if we go frame-by-frame, then we can see that longer strides have a longer intensity and that the player is lean forward a bit more. That explains that instance A was detected and not instance B"*. Furthermore, P2 and P3 mentioned that they could save time by reusing functionality available in the WDK. P3 said: *"I had to implement a lot of machine learning algorithms in Python. Here you can reuse a lot of functionality"*.

While using the WDK, the participants also mentioned different issues, bugs and feature requests. Two main issues they mentioned were the difficulty to identify the root of an error in a recognition algorithm they had developed and the difficulty to understand some of the reusable components in the WDK. Errors when executing a recognition algorithm were caused when two reusable components were connected to each other, although the data type produced by the predecessor component was not compatible with the input type required by the successor component. P1 said: *"If something fails, you don't know what went wrong"*. Furthermore, when executing an invalid algorithm, Matlab displays an error message containing the execution stack trace. Although the first line in the stack trace contained the name of the reusable component that caused the failure, the participants did not find this information helpful to identify the root of errors. Based on this feedback, we introduced a major change to the reusable components to prevent developers from connecting two incompatible components to each other. To this end, every reusable component now specifies a meta-data describing the type of its input and output parameters. The output type of a component is used to determine whether it can be connected to another component. At runtime, the reusable components print an error message when they receive an incompatible object as input and return an empty object, which causes the execution of an algorithm to stop. Furthermore, we adapted the user interfaces of every tool in the WDK to dynamically adapt the reusable components developers can choose from at each stage of the recognition pipeline based on the components selected at the previous stages.

Participants P1 and P3 also mentioned that they did not understand some of the reusable components available in the WDK, such as the *ManualSegmentation* and the different components to label events and segments. P1 said: *"The Labeling is not clear what it does"* and also pointed out that he did not know what the *LabelMapper* was for. P3 reported that she did not know how to *"get to a segment from an event"*, which can be done with the *EventSegmentation*. To address this issue, we documented every reusable component in the WDK's GitHub website. For each reusable component, the documentation describes the type of input it requires and output it produces.

[4]DaVinci Resolve is a video editing and annotation tool: https://www.blackmagicdesign.com/products/davinciresolve/

The participants also mentioned several minor issues. When using the *Annotation* tool, the participants mentioned the loss of annotations because of closing the window without saving them beforehand [P2], the lack of information regarding what signals were being produced when a preprocessing algorithm was executed [P3], the lack of a legend to indicate how computed signals mapped to colors in the plot [P3] and the difficulty to recognize a data selection due to the similarity of the colors used to plot data and to select a range of data [P1,P3]. Regarding the *Analysis* tool, the participants pointed out that the tool was too 'laggy' when zooming into a plot with more than 400 segments [P1, P2], the lack of feedback to indicate that a time-intensive computation had finished [P1], that there was no way to know which table was editable and which one was not [P1] and that the labels shown above each list box were not consistent, as some of them were numbered and others were not [P3]. P2 also requested a feature to plot different signals without having to reset the zoom level of the plots. In the *Assessment* tool, the participants had difficulties to create a feature extraction algorithm. This was due to a lack of consistency between the user interfaces to reuse components in the different stages: for the preprocessing, segmentation, classification and validation stages, a single reusable component had to be selected from the user interface, whereas feature extraction algorithms had to be created by selecting multiple components and defining on which signal each of them were to be computed. P2 and P3 also noted a lack of consistency in the user interface to select features, which required developers to have executed a recognition algorithm once before a subset of features could be selected, but provided no indication about this restriction over the user interface. In the detail view of the *Assessment* tool, P2 and P3 criticized that the results of the recognition were not always visible depending on the zoom level of the plot that displays the data. We performed several minor changes to improve the usability of our toolkit based on the issues mentioned by the participants.

To assess the usability of the improved version of the WDK, we conducted a second user study with two engineers from the industry. To this end, we contacted two companies located in Munich that had collaborated with our research lab in the past and asked them to participate in our user study. The first participant (P1) was a senior software engineer (33 years old) working at a startup that offers professional coaching to soccer goalkeepers. The second participant (P2) was a recent graduate of computer science (25 years old) working as a data scientist in a startup specialized in wearable electronics. Both participants had previous experience with activity recognition. P1 had used mostly Matlab and had only passing experience in Python and P2 had two years of experience in Python and was familiar with the Node-RED platform but had no experience in Matlab.

We gave the participants specific tasks to solve with the WDK while thinking out-loud using the data from the GoalieGlove application. The tasks included annotating a data set with *event* and *range* annotations, finding outliers in the annotated data set, comparing the different signals (accelerometer, gyroscope and magnetometer) corresponding to two exercises, discussing possible feature extraction algorithms based on the exercise signatures, developing the algorithm we presented in Section 4 and assessing its recognition performance. After solving these tasks, we conducted an unstructured interview with the participants to inquire about their impression using the WDK. Finally, the participants were given a questionnaire with seven 5-point Likert scale questions. Each session lasted approximately 90 minutes. Table 3 shows the questionnaire we asked and the participant's answers.

The ease to understand the reusable components in the WDK was rated 4 by P1 and 3 by P2. While the participants understood how to instantiate components and connect them together in the *Development* tool and in the code, they acknowledged the need to refer to the documentation to understand the functionality behind the different components. P1 said: *"I am not sure what all of these do, but I am sure you will have some documentation"*. P2 had difficulties to understand how to combine the event detection and segmentation components: *"Obviously you need some user manual to know that SimplePeakDetector works with the EventSegmentation"* but had no difficulty reusing the feature extraction and classification components.

We found that both participants were quickly able to understand what *event* and *range annotation*s are and to annotate a data set using the *Annotation* tool. They welcomed the *Annotation* tool and mentioned that they were

#	Question	P1	P2
Q1	Do you find the reusable components in the WDK easy to understand?	4	3
Q2	Do you find the tools in the WDK easy to use?	4	4
Q3	Do you find the WDK useful to annotate your data?	5	5
Q4	Do you find the WDK useful to study your data set?	5	4
Q5	Do you find the WDK useful to develop a recognition algorithm?	4	5
Q6	Do you find the WDK useful to assess the performance of a recognition algorithm?	5	4
Q7	How likely are you to use the WDK within your organization?	5	5

Table 3. Questionnaire and answers of participants of the second study. The scales were: 1 (very difficult) to 5 (very easy) for Q1 and Q2; 1 (useless) to 5 (very useful) for Q3-Q6 and 1 (very unlikely) to 5 (very likely) for Q7.

not aware of other free annotation tools for time series that display video files next to the data. Both participants rated the WDK's usefulness to annotate data with a 5 (very useful). We also observed that the participants could use the *Analysis* tool without issues to display the annotated data. They quickly found outlier motions in the annotations and discussed possible feature extraction algorithms based on the data. Both participants found the tool useful to make sense of their data sets and design feature extraction algorithms. P1 said: *"The Analysis App is the most useful tool because it helps you see what's going on with the data. It helps you choose the features because you can see patterns in the data and on which axes to calculate the feature"*. The participants rated the WDK's usefulness to study their data sets with a 5 (P1) and a 4 (P2).

Both participants rated the WDK's usefulness to assess the performance of a recognition algorithm with a score of 5 (P1) and 4 (P2). In particular, the functionality to display the recognition results on top of the raw data in the frame-by-frame analysis was identified as the most convenient feature. P2 said: *"the part of the assessment can differentiate [the WDK] from other tools. [...] if you see a confusion matrix you see it misclassifies these exercises but you don't have a clue why [...]. It can help a lot to see the video and see that because of this it was not properly predicted and see that together with the data"*.

Both participants praised the WDK and rated how likely they were to use it within their organizations with a 5 (very likely). P2 said: *"there are no tools that are publicly available to developers so they create their own software [...] or they just do it intuitively by using standard parameters trusting they will work for their specific problem. With this tool I can see the data with different parameters and decide"*. On the other hand, both participants pointed out Matlab license fees as an issue and mentioned that their organizations would not be willing to afford the fees.

6.1 Discussion

Based on what we observed, we feel confident that the WDK can significantly lower the entrance barrier to the development of activity recognition applications. The participants of our studies mentioned that they were not aware of similar tools and found the WDK useful to automatize their development tasks. In particular, they praised the ability to reuse a broad set of existing functionality in their own applications. The features perceived to be the most useful by the participants are the functionality to annotate the data together with the video in the *Annotation* tool, to quickly assess and optimize the parameters of different algorithms and the frame-by-frame analysis to correlate the recognition results to the original data and reference video.

Furthermore, the participants of the user studies mentioned the difficulty to understand some reusable components. While the WDK enables the reuse of high-level components without having to understand their implementation details, developers still need to 1) be familiar with the Activity Recognition Chain and 2) understand the function, inputs and output produced by the components in the WDK. However, we believe that understanding and reusing the components in the WDK is significantly less time consuming than implementing a recognition algorithm without them. To facilitate learning the Activity Recognition Chain as well as the abstractions behind the WDK, we recently created a tutorial on activity recognition that relies on the reusable components in the WDK[5]. We found that most developers with no experience in activity recognition are able to finish the tutorial in a few hours and that they have less questions and are more effective at using the WDK afterwards.

In addition, both engineers from the second user study pointed out Matlab's license fees as a main limitation and suggested Python as a free alternative. In the future, the components the WDK offers could be re-implemented in Python or C++, or a combination of both. A C++ implementation of the *runtime components* would avoid differences between the execution of algorithms in the development environment and target device and is likely to lead to better execution performance. The current design of the WDK can be reused in future implementations.

7 CONCLUSIONS

This paper presented a toolkit to facilitate the development of activity recognition applications with wearables. In contrast to previous work, the WDK supports different tasks in the development lifecycle of an activity recognition application, such as the annotation and analysis of data and the development and performance assessment of an algorithm. Supporting these tasks within a single environment facilitates an iterative development process which is often necessary because developers rarely know upfront how to design activity recognition systems but rather develop them iteratively. To ensure the versatility of the toolkit, we developed it incrementally based on a variety of applications from different domains including sports, health, animal welfare and daily activity monitoring. We also collected feedback from different users with varying levels of experience in activity recognition and adapted the WDK to ensure it meets their needs.

One aspect we haven't studied until now is how well the *execution* and *memory cost*s computed by the WDK correlate to the amount of floating point operations and memory an actual algorithm implementation in the target device would require. As these metrics depend on the target device, its architecture and drivers, estimating them accurately at development time can be challenging. However, the costs estimated by the WDK provide a rough estimate that can be used to compare two or more algorithms to each other and make decisions early in the development lifecycle of an activity recognition application. Furthermore, the *execution* and *memory* costs of each reusable component can be adapted to a specific benchmark by modifying two lines of code in each component.

Furthermore, the current version of the WDK is limited to local computations. If the scale of the data exceeds what is physically possible to compute in a reasonable amount of time on a local device, developers might need to use remote computing power. Future work could extend the WDK to enable the simulation and assessment of activity recognition algorithms in parallel. To this end, every stage until the classification stage could be executed in parallel for the different input data files.

Despite the variety of reusable components and functionality already available in the WDK, the toolkit is far from finished. We are still extending its set of reusable components, refactoring its code, improving its usability and documenting it. A particular feature we are working on is the deployment of recognition algorithms into wearable devices. Our vision is to do so by sending an algorithm configuration wirelessly, without recompiling and flashing a firmware. To this end, we are currently porting the WDK's *runtime components* to C++.

We also haven't studied how to support application developers at creating applications that rely on activity recognition algorithms. We believe that many applications handle recognition results in similar ways. For example, they keep track of a training performance over time and compare the training performances among users. Future work could identify patterns of usage of recognized activities within applications and facilitate their development.

While the WDK eases the effort to develop recognition algorithms, these still need to be crafted manually. Different groups are investigating how to avoid this manual effort by adapting artificial neural networks to activity

[5]https://github.com/avenix/ARC-Tutorial

recognition applications with wearables [41, 60]. We are currently studying how to automatize the development of recognition algorithms by means of *generative design*. In generative design, the assembly of activity recognition algorithms is formulated as an optimization problem where the recognition performance (e.g., F1-Score) is used as an optimization metric and the computational requirements (e.g., memory, energy consumption) derive into constraints to the optimization. The reusable components in the WDK and the functionality to assess the performance of an algorithm represent a first step towards the realization of this idea.

ACKNOWLEDGMENTS

This work would not have been possible without the support from Prof. Bernd Brügge and Prof. Dan Siewiorek over the last two years. The author would also like to thank Prof. Oliver Amft and Prof. Antonio Krügger for allowing this work to be presented at their research labs and providing valuable ideas to improve the WDK.

REFERENCES

[1] Oliver Amft. 2010. A wearable earpad sensor for chewing monitoring. In *SENSORS, 2010 IEEE*. IEEE, 222–227.
[2] Daniel Ashbrook and Thad Starner. 2010. MAGIC: a motion gesture design tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2159–2168.
[3] Rafael Ballagas and Faraz Memon. 2007. iStuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. *Proceedings of the SIGCHI conference on Human factors in computing systems* (2007), 1107–1116. https://doi.org/10.1145/1240624.1240793
[4] David Bannach, Oliver Amft, and Paul Lukowicz. 2008. Rapid Prototyping of Activity Recognition Applications. *IEEE Pervasive Computing* 7, 2 (apr 2008), 22–31. https://doi.org/10.1109/MPRV.2008.36
[5] Ling Bao and Stephen S Intille. 2004. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*. Springer, 1–17.
[6] Peter Blank, Julian Hoßbach, Dominik Schuldhaus, and Bjoern M Eskofier. 2015. Sensor-based stroke detection and stroke type classification in table tennis. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 93–100.
[7] Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)* 46, 3 (2014), 33.
[8] Jay Chen, Karric Kwong, Dennis Chang, Jerry Luk, and Ruzena Bajcsy. 2006. Wearable sensors for reliable fall detection. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. IEEE, 3551–3554.
[9] Pei-Yu Peggy Chi and Yang Li. 2015. Weave: Scripting cross-device wearable interaction. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, 3923–3932.
[10] Guglielmo Cola, Marco Avvenuti, Alessio Vecchio, Guang-Zhong Yang, and Benny Lo. 2015. An on-node processing approach for anomaly detection in gait. *IEEE Sensors Journal* 15, 11 (2015), 6640–6649.
[11] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. ACM Press, New York, New York, USA, 33–40. https://doi.org/10.1145/985692.985697
[12] Jessica Echterhoff, Juan Haladjian, and Bernd Brügge. 2018. Gait Analysis in Horse Sports. In *Proceedings of the Fifth International Conference on Animal-Computer Interaction*. ACM, 3.
[13] Jessica Echterhoff, Juan Haladjian, and Bernd Brügge. 2018. Gait and Jump Classification in Modern Equestrian Sports. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, 88–91.
[14] Davide Figo, Pedro C Diniz, Diogo R Ferreira, and João M Cardoso. 2010. Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing* 14, 7 (2010), 645–662.
[15] Francine Gemperle, Chris Kasabach, John Stivoric, Malcolm Bauer, and Richard Martin. 1998. Design for wearability. In *digest of papers. Second international symposium on wearable computers (cat. No. 98EX215)*. IEEE, 116–122.
[16] Nicholas Gillian and Joseph A Paradiso. 2014. The gesture recognition toolkit. *The Journal of Machine Learning Research* 15, 1 (2014), 3483–3487.
[17] Saul Greenberg and Chester Fitchett. 2001. Phidgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology - UIST '01*. ACM Press, New York, New York, USA, 209. https://doi.org/10.1145/502348.502388
[18] Benjamin H Groh, Martin Fleckenstein, Thomas Kautz, and Bjoern M Eskofier. 2017. Classification and visualization of skateboard tricks using wearable sensors. *Pervasive and Mobile Computing* 40 (2017), 42–55.
[19] Tobias Grosse-Puppendahl, Yannick Berghoefer, Andreas Braun, Raphael Wimmer, and Arjan Kuijper. 2013. OpenCapSense: A rapid prototyping toolkit for pervasive interaction using capacitive sensing. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 152–159.

[20] Juan Haladjian, Katharina Bredies, and Bernd Bruegge. 2016. Interactex: An integrated development environment for smart textiles. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 8–15. https://doi.org/10.1145/2971763.2971776
[21] Juan Haladjian, Katharina Bredies, and Bernd Bruegge. 2018. KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries. In *Proceedings of the 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. IEEE, 9–12.
[22] Juan Haladjian, Ayca Ermis, Zardosht Hodaie, and Bernd Brügge. 2017. iPig: Towards Tracking the Behavior of Free-roaming Pigs. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction (ACI2017)*. ACM, New York, NY, USA, 10:1—-10:5. https://doi.org/10.1145/3152130.3152145
[23] Juan Haladjian, Johannes Haug, Stefan Nüske, and Bernd Bruegge. 2018. A Wearable Sensor System for Lameness Detection in Dairy Cattle. *Multimodal Technologies and Interaction* 2, 2 (2018), 27.
[24] Juan Haladjian, Zardosht Hodaie, Stefan Nüske, and Bernd Brügge. 2017. Gait Anomaly Detection in Dairy Cattle. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction (ACI2017)*. ACM, New York, NY, USA, 8:1—-8:8. https://doi.org/10.1145/3152130.3152135
[25] Juan Haladjian, Zardosht Hodaie, Han Xu, Mertcan Yigin, Bernd Bruegge, Markus Fink, and Juergen Hoeher. 2015. KneeHapp: A Bandage for Rehabilitation of Knee Injuries. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 181–184.
[26] Juan Haladjian, Constantin Scheuermann, Katharina Bredies, and Bernd Bruegge. 2017. A Smart Textile Sleeve for Rehabilitation of Knee Injuries. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers (UbiComp '17)*. ACM, New York, NY, USA, 49–52. https://doi.org/10.1145/3123024.3123151
[27] Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07)* (2007), 145–154. https://doi.org/10.1145/1240624.1240646
[28] Björn Hartmann, Scott R Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. ACM, 299–308.
[29] Steven Houben, Connie Golsteijn, Sarah Gallacher, Rose Johnson, Saskia Bakker, Nicolai Marquardt, Licia Capra, and Yvonne Rogers. 2016. Physikit: Data engagement through physical ambient visualizations in the home. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 1608–1619.
[30] Steven Houben and Nicolai Marquardt. 2015. Watchconnect: A toolkit for prototyping smartwatch-centric cross-device applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1247–1256.
[31] Bonifaz Kaufmann and Leah Buechley. 2010. Amarino: A Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '10)*. ACM, New York, NY, USA, 291–298. https://doi.org/10.1145/1851600.1851652
[32] Aftab Khan, James Nicholson, and Thomas Plötz. 2017. Activity Recognition for Quality Assessment of Batting Shots in Cricket using a Hierarchical Representation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 62.
[33] Travis Kirton, Sebastien Boring, Dominikus Baur, Lindsay MacDonald, and Sheelagh Carpendale. 2013. C4: a creative-coding API for media, interaction and animation. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 279–286.
[34] David Ledo, Fraser Anderson, Ryan Schmidt, Lora Oehlberg, Saul Greenberg, and Tovi Grossman. 2017. Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2583–2593.
[35] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 36.
[36] Johnny C. Lee, Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz, and Darren Leigh. 2004. The calder toolkit. In *Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04*. ACM Press, New York, New York, USA, 167–175. https://doi.org/10.1145/1013115.1013139
[37] Yang Li, Jason I Hong, and James A Landay. 2004. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*. ACM, 217–226.
[38] Kent Lyons, Helene Brashear, Tracy Westeyn, Jung Soo Kim, and Thad Starner. 2007. Gart: The gesture and activity recognition toolkit. In *International Conference on Human-Computer Interaction*. Springer, 718–727.
[39] Javier Marco, Eva Cerezo, and Sandra Baldassarri. 2012. ToyVision: a toolkit for prototyping tabletop tangible games. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 71–80.
[40] Amon Millner and Edward Baafi. 2011. Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In *Proceedings of the 10th International Conference on Interaction Design and Children*. ACM, 250–253.

[41] Vishvak S Murahari and Thomas Plötz. 2018. On attention models for human activity recognition. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, 100–103.

[42] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. 2014. Interactive development of cross-device user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2793–2802.

[43] J Nielsen. 1994. *Usability Engineering*. Academic Press Inc.

[44] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 249–256.

[45] Girish Palshikar and Others. 2009. Simple algorithms for peak detection in time-series. In *Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence*, Vol. 122.

[46] Shyamal Patel, Delsey Sherrill, Richard Hughes, Todd Hester, Theresa Lie-Nemeth, Paolo Bonato, David Standaert, and Nancy Huggins. 2006. Analysis of the Severity of Dyskinesia in Patients with Parkinson's Disease via Wearable Sensors. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06)*. IEEE, 123–126. https://doi.org/10.1109/BSN.2006.10

[47] Max Pfeiffer, Tim Duente, and Michael Rohs. 2016. Let your body move: a prototyping toolkit for wearable force feedback with electrical muscle stimulation. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, 418–427.

[48] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 409–419.

[49] Raf Ramakers, Kashyap Todi, and Kris Luyten. 2015. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15* (2015), 2457–2466. https://doi.org/10.1145/2702123.2702487

[50] Valkyrie Savage, Colin Chang, and Björn Hartmann. 2013. Sauron: embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 447–456.

[51] Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. 2015. Makers' Marks: Physical markup for designing and fabricating functional objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 103–108.

[52] Giovanni Schiboni and Oliver Amft. 2018. Automatic dietary monitoring using wearable accessories. In *Seamless Healthcare Monitoring*. Springer, 369–412.

[53] Dominik Schuldhaus, Carolin Jakob, Constantin Zwick, Harald Koerger, and Bjoern M Eskofier. 2016. Your personal movie producer: generating highlight videos in soccer using wearables. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 80–83.

[54] Teddy Seyed, Alaa Azazi, Edwin Chan, Yuxi Wang, and Frank Maurer. 2015. Sod-toolkit: A toolkit for interactively prototyping and developing multi-sensor, multi-device environments. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*. ACM, 171–180.

[55] Akira Wakita and Yuki Anezaki. 2010. Intuino: an authoring tool for supporting the prototyping of organic interfaces. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. ACM, 179–188.

[56] Chiuan Wang, Hsuan-Ming Yeh, Bryan Wang, Te-Yen Wu, Hsin-Ruey Tsai, Rong-Hao Liang, Yi-Ping Hung, and Mike Y Chen. 2016. CircuitStack: supporting rapid prototyping and evolution of electronic circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 687–695.

[57] Tracy Westeyn, Helene Brashear, Amin Atrash, and Thad Starner. 2003. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *Proceedings of the 5th international conference on Multimodal interfaces*. ACM, 85–92.

[58] Chi-Jui Wu, Steven Houben, and Nicolai Marquardt. 2017. Eaglesense: Tracking people and devices in interactive spaces using real-time top-view depth-sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 3929–3942.

[59] Jishuo Yang and Daniel Wigdor. 2014. Panelrama: enabling easy specification of cross-device web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2783–2792.

[60] Ming Zeng, Haoxiang Gao, Tong Yu, Ole J Mengshoel, Helge Langseth, Ian Lane, and Xiaobing Liu. 2018. Understanding and improving recurrent networks for human activity recognition by continuous attention. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, 56–63.

[61] Bo Zhou, Harald Koerger, Markus Wirth, Constantin Zwick, Christine Martindale, Heber Cruz, Bjoern Eskofier, and Paul Lukowicz. 2016. Smart soccer shoe: monitoring foot-ball interaction with shoe integrated textile pressure sensor matrix. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 64–71.

A APPENDIX

This section lists the reusable components in the WDK until the date of submission of this article. The first and second columns of the tables provide the name and a description of each component. The *execution*, *memory* and *communication cost*s are abbreviated as *Exec*, *Mem* and *Comm* and described with respect to an input of size n.

Preprocessing Components Runtime		Exec	Mem
HighPassFilter	Butterworth High-pass filter	$13 * o * n$	*
LowPassFilter	Butterworth Low-pass filter	$31 * o * n$	*
Magnitude	$\sqrt{a_x(x_i)^2 + a_y(x_i)^2 + a_z(x_i)^2}$	$4 * n$	*
SquaredMagnitude	$a_x(x_i)^2 + a_y(x_i)^2 + a_z(x_i)^2)$	$2 * n$	*
Norm	$\lvert a_x(x_i)\rvert + \lvert a_y(x_i)\rvert + \lvert a_z(x_i)\rvert)$	$2 * n$	*
Derivative	$D'_i(x) = (x_i - x_{i+1})/\delta$ and $D''_i(x) = (x_{i-1} - x_i + x_{i+1})/\delta^2$	$40 * n$	*
S1	$\frac{max(x_i - x_{i-1}, ..., x_i - x_{i-k}) + max(x_i - x_{i+1}, ..., x_i - x_{i+k})}{2}$	$40 * k * n$	n
S2	$\frac{max(x_i - x_{i-1}, ..., x_i - x_{i-k}) + max(x_i - x_{i+1}, ..., x_i - x_{i+k})}{2k}$	$203 * k * n$	n

Table 4. The preprocessing components produce n 32-bit floating-point values. The o variables in the *HighPassFilter* and *LowPassFilter* refer to these components' *order* property. The algorithms with a (*) in the memory field require O(1) memory when their *computationInPlace* property is set to *true* or $O(n)$ additional memory otherwise.

Event Detection Components Runtime		Exec	Mem
SimplePeakDetector	Threshold-based peak detector	$11 * n$	1
MatlabPeakDetector	Matlab's peak detector	$1787 * n$	n

Table 5. The event detection components produce either none or one 32-bit floating-point value.

Segmentation Components Runtime		Exec	Mem
SlidingWindow	Extracts *Segments* of *windowSize* from the input *Signal*	$(n - s)/it$	s
EventSegmentation	Creates *Segment*s around the input *Events*	$11 * n$	$l + r$
ManualSegmentation	Converts *event* and *range annotation*s to *Segments*	-	-

Table 6. The segmentation components produce s or $l + r$ values. The s, l, it and r variables in the *Exec* and *Mem* columns refer to these components' *segmentSize*, *iterationSize*, *segmentSizeLeft* and *segmentSizeRight* properties, respectively.

Time-domain Feature Extraction Components Runtime		Exec	Mem
Min	Minimum value in the input *Signal*	n	1
Max	Maximum value in the input *Signal*	n	1
Mean	Average of every value in the input *Signal*	n	1
Median	Median of the values in the input *Signal*	$15 * n$	1
Variance	Variance of the input *Signal*	$2 * n$	1
STD	Standard Deviation of the values in the input *Signal*	$2 * n$	1
ZCR	Zero Crossing Rate of the input *Signal*	$5 * n$	1
Skewness	Skewness of the input *Signal*: $\sum_{i=1}^{n} \left(\frac{x_i - \bar{x})}{\sigma} \right)^3$	$6 * n$	1
Kurtosis	kurtosis of the input *Signal*: $\sum_{i=1}^{n} \left(\frac{x_i - \bar{x})}{\sigma} \right)^4$	$6 * n$	1
IQR	Interquartile Range of the values in the input *Signal*	$57 * n$	n
AUC	Area under the curve (trapezoid rule) of the input *Signal*: $\sum_{i=1}^{n-1} \frac{x_i + x_{i+1}}{n}$	$8 * n$	1
AAV	Average Absolute Variation of the input *Signal*: $\sum_{i=1}^{n-1} \frac{\lvert x_i - x_{i+1} \rvert}{n}$	$5 * n$	1
Correlation	Pearson correlation coefficient of the two input *Signal*s	$3 * n$	n
Energy	Sum of squared values of the input *Signal*	$2 * n$	1
Entropy	Entropy of the input signal: $\sum_{i=1}^{n} p_i \log(p_i)$ where p_i are the probability distribution values of the input *Signal*	n^2	n
MAD	Mean Absolute Deviation of the input *Signal*: $\sum_{i=1}^{n} \frac{\lvert x_i - \bar{x} \rvert}{n}$	$5 * n$	1
MaxCrossCorr	Maximum of the cross correlation coefficients of two input *Signals*	$161 * n$	n
Octants	Octant of each sample in the three input *Signal*s	$7 * n$	1
P2P	Difference between max. and min. values of the input *Signal*	$3 * n$	1
Quantile	q cutpoints that separate the distribution of values in the input *Signal*	$3 * n * log(n)$	q
RMS	Root Mean Squared of the input *Signal*: $\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$	$2 * n$	1
SMV	Signal Vector Magnitude of a two-dimensional input *Signal*: $\frac{1}{n} \sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2)}$	$4 * n$	1
SMA	Sum of absolute values of a one or two-dimensional input *Signal*: $\sum_{i=1}^{n} \sum_{j=1}^{n} \lvert x_{ij} \rvert$	$m * n$	1

Table 7. The time-domain feature extraction algorithms produce a single value except for the *Quantile* component, which produces *numQuantileParts* values. The octant is defined as: $Octant = 1$ if $x_1, x_2, x_3 > 0$ and $Octant = 7$ if $x_1, x_2, x_3 < 0$.

Frequency-domain Feature Extraction Components Runtime		Exec	Mem
FFT	FFT of the input *Signal*	$n * \log(n)$	n
FFTDC	DC component of FFT coefficients	1	1
MaxFrequency	Largest Fourier coefficient	n	1
PowerSpectrum	Power spectrum of FFT coefficients	$4 * n$	n
SpectralCentroid	Centroid of FFT coefficients: $\frac{\sum_{i=1}^{n-1} \bar{y}_i y_i}{\sum_{i=1}^{n-1} y_i}$	$10 * n$	1
SpectralEnergy	Squared sum of FFT coefficients: $\sum_{i=1}^{n} \bar{y_i}^2$	$2 * n$	1
SpectralEntropy	Entropy of the FFT coefficients: $-\sum_{i=1}^{n} y_i \log 2(y_i)$	$21 * n$	1
SpectralFlatness	Flatness of the distribution of FFT coefficients: $\frac{\sqrt[n]{\prod_{i=1}^{n} x_i}}{\frac{1}{n} \sum_{i=1}^{n} x(n)}$	$68 * n$	1
SpectralSpread	Variance of the distribution of FFT coefficients	$11 * n$	1

Table 8. The frequency-domain feature extraction components output a single value except for the *FFT* and *PowerSpectrum* which produce $n/2$ and n values respectively. Every frequency-domain feature extraction component receives the *Signal* with FFT coefficients produced by the *FFT* component as input.

Classification Components Runtime	
LDClassifier	Linear Discriminant classifier
TreeClassifier	Decision tree classifier with properties: *maxNumSplits*
KNNClassifier	K-NN classifier with properties: *nNeighbors*, *distanceMetric*
EnsembleClassifier	Ensemble classifier with properties: *nLearners*
SVMClassifier	Support Vector Machine classifier with properties: *order*, *boxConstraint*

Table 9. The classification components produce 9 bytes (a 1-byte label and an 8-byte timestamp). Their computational performance depend strongly on their implementation.

Postprocessing Components Runtime		Exec	Mem
LabelMapper	Transforms the array of labels in a *ClassificationResult* by mapping labels in the *sourceLabeling* property to labels in the *targetLabeling* property	n	n
LabelSlidingWindowMaxSelector	Replaces every label at index *labelIndex* in a *ClassificationResult* with the most frequent label in the range [*labelIndex* − *windowSize*, *labelIndex* + *windowSize*], or with the NULL-class if no label occurs at least *minimumCount* times in the range	1	1

Table 10. The postprocessing components produce 9 bytes (a 1-byte label and an 8-byte timestamp).

Runtime Utility Components Runtime		Exec	Mem	Comm
FeatureNormalizer	Normalizes a *FeaturesTable* by subtracting each row from the *means* property and dividing it by the *stds* property	$2 * n$	$2 * n$	n
ConstantMultiplier	Multiplies an input *Signal* by the *constant* property	n	n	n
Substraction	Subtracts the second column from the first column of a two-dimensional input *Signal*	$2 * n$	n	n
AxisMerger	Merges *m* *Signals* of size *n* into an *nxm* *Signal*	$3 * n$	$m * n$	$m * n$
AxisSelector	Selects the *axes* columns of the provided input *Signal*	-	$m * n$	$m * n$
RangeSelector	Outputs a new *Signal* with the values in the range $[rs...re]$ of the input *Signal*	$2 * n$	$re - rs$	$re - rs$

Table 11. Utility components available in the *runtime components* layer.

File Management Components Development	
FilesLoader	Loads and parses a data file (.csv or .mat) formats.
AnnotationsLoader	Loads ans parses an annotations file (.txt format)

Table 12. The *FilesLoader* and *AnnotationsLoader* are convenience components used during development.

Labeling Components Development	
EventsLabeler	Labels *Events* as the closest *event annotation* under a specified *tolerance*
EventSegmentsLabeler	Labels *Segments* extracted around a detected *Event*
RangeSegmentsLabeler	Labels *Segments* based on *range annotations*

Table 13. The labeling components are methods to label the *Event*s and *Segment*s produced by a recognition algorithm.

Validation Components Development	
HoldoutValidator	Trains a classifier using the *trainData* and tests its with the *testData*
LeaveOneOutCrossValidator	Applies the leave-one-subject-out cross-validation technique

Table 14. The validation components receive a set of *FeaturesTable*s as input and produce a *ClassificationResult*.

Development Utility Components Development	
FeatureExtractor	Generates a *FeaturesTable* from an array of *Segments*
FeatureSelector	Identifies the *nFeatures* most relevant features of a *FeaturesTable*
NoOp	Outputs the input object without modification
PropertyGetter	Outputs the *property* property of the input object
PropertySetter	Sets the *property* property of the object in the *node* property to the input value
SegmentsGrouper	Outputs the input *Segments* grouped by their class
TableRowSelector	Removes every row of the input *FeaturesTable* with a label column not contained in the *selectedLabels* property.

Table 15. Utility components available in development *components layer* of the repository.

6.2 Sensor-based Detection and Classification of Soccer Goalkeeper Training Exercises

This publication presents a wearable device application that detects and recognizes the training exercises performed by soccer goalkeepers (e.g. catches, throws, dives) using a wearable motion sensor attached to the glove. We provide a comprehensive literature review on similar activity recognition applications with wearable devices, describe each computation performed to accurately detect training exercises, filter out irrelevant motions detected (i.e. false positives) and determine the actual type of exercise using a machine learning classifier.

The author of this Habilitation developed the activity recognition algorithm including the collection of data during goalkeeper trainings, the development and assessment of different recognition methods and wrote the article.

Authors	Haladjian, J., Schlabbers, D., Taheri, S., Tharr, M., & Bruegge, B.
Journal	Transactions on Internet of Things (TIoT)
Number of Pages	14
Type	Journal Article
Review	Peer Reviewed (4 Reviewers)
Year	2019
DOI	

Sensor-based Detection and Classification of Soccer Goalkeeper Training Exercises

JUAN HALADJIAN, Technical University Munich
DANIEL SCHLABBERS, Technical University Munich
SAJJAD TAHERI, Technical University Munich
MAX THARR, Technical University Munich
BERND BRUEGGE, Technical University Munich

Many goalkeeper trainees cannot afford a personal human coach. Hence, they could benefit from a virtual coach that provides personalized feedback about the execution of their training exercises. As a first step towards this goal, we developed an algorithm to detect and classify goalkeeper training exercises using a wearable inertial sensor attached to a goalkeeper glove. We collected data from 14 goalkeeper trainees while performing a series of training exercises (e.g., dives, catches, throws). Our approach first detects the exercises using an event detection algorithm based on a high-pass filter, a peak detector, and Dynamic Time Warping to detect and eliminate irrelevant motion instances. Then, it extracts a set of statistical and heuristic features to describe the different exercises and train a machine learning classifier. Our exercise detection approach retrieves 93.8% of the relevant exercises with 90.6% precision and classifies the detected exercises with an accuracy of 96.5%. The exercises recognized by our algorithm can be used to compute further qualitative metrics about individual exercise executions to provide goalkeepers with relevant feedback about their training.

CCS Concepts: • **Human-centered computing** → *Personal digital assistants*; • **Hardware** → *Sensor devices and platforms*;

Additional Key Words and Phrases: Soccer, goalkeeping, event detection, wearable sensor, activity recognition, signal processing, machine learning, dynamic time warping

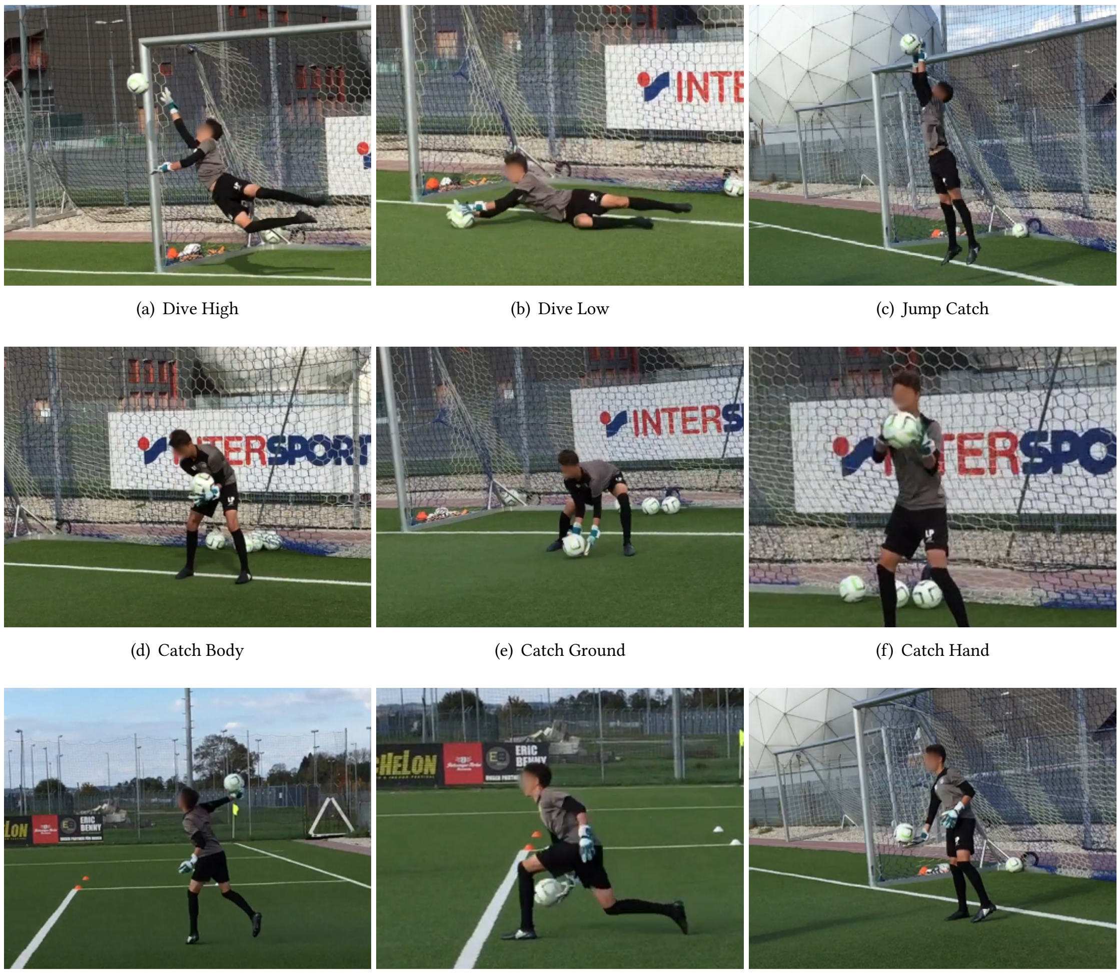
ACM Reference Format:
Juan Haladjian, Daniel Schlabbers, Sajjad Taheri, Max Tharr, and Bernd Bruegge. 2018. Sensor-based Detection and Classification of Soccer Goalkeeper Training Exercises. *ACM Trans. Internet Things* 1, 1, Article 1 (November 2018), 20 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Soccer is one of the most popular sports in the world with numerous professional and an even larger number of non-professional practitioners. A soccer team consists of ten field players and a goalkeeper. Due to their unique role within the team, goalkeepers undergo a different training than the rest of the players in the team. Goalkeepers train a specific set of well-defined motions to consolidate them into muscle memory and lower their reaction time during

Authors' addresses: Juan Haladjian, Technical University Munich, juan.haladjian@tum.de; Daniel Schlabbers, Technical University Munich, daniel.schlabbers@tum.de; Sajjad Taheri, Technical University Munich, sajjad.taheri@tum.de; Max Tharr, Technical University Munich, max.tharr@tum.de; Bernd Bruegge, Technical University Munich, bruegge@tum.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
Manuscript submitted to ACM



(a) Dive High (b) Dive Low (c) Jump Catch

(d) Catch Body (e) Catch Ground (f) Catch Hand

(g) Throw High (h) Throw Low (i) Pass

Fig. 1. Goalkeeper training exercise variations.

a game. The correct execution of the exercises have a steep learning curve and can only be assessed by experienced trainers.

Despite the importance of personalized feedback from a professional trainer, only older, more experienced goalkeepers enrolled in a soccer club have access to a dedicated trainer. In contrast, most young goalkeeper trainees cannot afford a personalized coach. The ultimate goal of our work is to realize a system based on an unobtrusive wearable sensor able to provide personalized and objective feedback to goalkeepers automatically after or during a training session to help them improve their skills.

A goalkeeper training session typically includes exercises to stop incoming balls (e.g., dives, catches, jumps) and others to pass the ball at field players. Goalkeepers execute different variations of these exercises to learn the right movements—mostly involving the legs, torso and arms—to handle each incoming ball. For example, they execute a different motion to catch balls thrown at their chest, belly and legs. They catch balls thrown at the chest with their hands, wrap balls thrown at their belly between the hands and their chest and control lower balls on the ground using their entire body. Figure 1 shows different variations of goalkeeper training exercises.

In this article, we present a smart glove and algorithm to automatically detect and keep track of goalkeeper training exercises using a single inertial sensor. To the best of our knowledge, this application has not been studied previously. The work we present enables goalkeepers to keep track of the training exercises they perform. Goalkeepers could get an overview of the exercises performed by goalkeepers during the last days, weeks or months, which they could use to plan future training sessions accordingly. More importantly, the ability to detect and classify training exercises is a first step towards a virtual coach that extracts performance metrics from the recognized exercises and gives personalized feedback to goalkeepers.

The high variability in the training exercises and individual execution of each exercise make this application extremely challenging. A particular challenge we address is on filtering out irrelevant motions that originate from the high degree of freedom in the movement of a goalkeeper's hands. The development process and methods we describe in this article can be reused in similar activity recognition applications.

This paper is structured as follows. Section 2 lists similar activity recognition applications and discusses the state of the art in activity recognition with wearable sensors. In Section 3, we present our algorithm to detect and classify goalkeeper training exercises using the signal produced by an inertial sensor. Section 4 presents the results of our evaluation to quantify the performance of our detection and classification algorithms. In Section 5, we discuss the results we obtained and in Section 6 we summarize our contribution and present future research directions towards the realization of a virtual coach for soccer goalkeepers.

2 RELATED WORK

Activity recognition using wearable sensors has been a field of study for over two decades. In this period, a variety of activity recognition applications have been developed that extract information about the user or her context using sensor signals. This is done for several possible reasons: 1) to assess the performance and correctness of a physical exercise of a patient or athlete, 2) to monitor a physiological parameter or activity over time, 3) to provide feedback to a user about her actions and 4) to adapt a user interface to the user's context. This section first provides a list of similar activity recognition applications and then discusses the state of the art in recognition methods.

2.1 Activity Recognition Applications

Activity recognition applications developed so far can be grouped under different fields:

- **Daily activity monitoring**. One of the first activity recognition applications proposed by the research community used inertial sensors to recognize daily activities (e.g. walking, jogging, standing, sitting) [Bao and Intille 2004; Tapia et al. 2004]. Since then, several applications have been studied including the automatic detection of falls [Abbate et al. 2012; Chen et al. 2006] and the recognition and monitoring of dietary activities such as drinking [Amft et al. 2005; Schiboni and Amft 2018] and chewing [Amft 2010; Amft et al. 2005].
- **Medicine**. Several studies have developed methods to segment strides and extract information from human gait [Barth et al. 2015]. In particular, the gait of patients of Parkinson's disease has been widely investigated by different research groups [Mariani et al. 2013; Patel et al. 2009, 2006]. Activity recognition applications have also

been developed to extract objective performance metrics to support clinical assessment during rehabilitation and physical therapy. Performance metrics include the angle of flexion of a leg and the stability while performing a squat during rehabilitation after a knee surgery [Haladjian et al. 2018a, 2015].

- **Sports**. Wearable activity recognition applications in sports have been developed to detect and classify strokes (i.e. serves) in table tennis [Blank et al. 2015] and in tennis [Yang et al. 2017], determine the part of the shoe used to kick a ball in soccer [Zhou et al. 2016], classify tricks in skateboarding [Groh et al. 2017a], compute the velocity and the length of a jump in ski jumping [Groh et al. 2017b], calculate performance parameters in swimming such as the amount of strokes and time needed per lane [Bächlin et al. 2009] and recognize batting shots in cricket [Khan et al. 2017].
- **Animal welfare and sports**. Although the field is often referred to as *Human Activity Recognition* (HAR) [Bulling et al. 2014], several activity recognition applications have been developed for non-human users. These include wearable sensors to: detect deviations in the usual gait of cows in order to warn veterinarians about possible lameness-related diseases [Haladjian et al. 2018b, 2017], assess the performance during horse dressage riding [Thompson et al. 2015], recognize gaits and compute the duration of jumps in equestrian show jumping [Echterhoff et al. 2018a,b] and track the activities (i.e. eating, walking, resting) of sheep [Walton et al. 2018].

Few research works have studied the use of wearable sensors in soccer applications. Zhou et al. [Zhou et al. 2016] attached a pressure sensitive smart textile matrix to a soccer shoe and studied how to compute the angle at which a soccer ball is kicked. Similarly, Weizman et al. [Weizman and Fuss 2015] developed a smart shoe system with a matrix of pressure sensors and a user interface that displays the force and center of pressure of soccer kicks. Hossain et al. [Hossain et al. 2017] studied the use of wrist-worn sensors to classify motion performed by soccer field players such as passes, kicks, sprints, runs and dribblings. Schuldhaus et al. [Schuldhaus et al. 2016] used a wearable motion sensor in the insole of a soccer shoe to detect specific motions such as dribbling and kicking, which they use to generate video highlights automatically.

Other studies have used video cameras to automatically extract metrics from soccer games, such as the positions of the field players and ball trajectories [Figueroa et al. 2006; Müller Junior and Anido 2004]. Our work focuses on tracking soccer goalkeeper training exercises. We do this with an unobtrusive wearable sensor strapped around the glove at the goalkeeper's main hand that does not require calibration or setup.

2.2 Activity Recognition Methods

Most activity recognition applications developed so far are hand-crafted chains of computations, known as the activity recognition chain [Bulling et al. 2014] to extract information from sensor signals. Multiple methods to segment sensor signals, extract features and to select and prioritize features have been studied in the past. Methods used to recognize patterns in sensor signals proposed so far include template-based methods such as string matching [Stiefmeier et al. 2007] and Dynamic Time Warping (DTW) [Barth et al. 2015; Muscillo et al. 2007; Plouffe and Cretu 2015; Seto et al. 2015], probabilistic methods such as the Hidden Markov Model (HMM) [Li et al. 2015; Martindale et al. 2017; Schiboni and Amft 2018] and machine learning classifiers such as Support Vector Machines (SVMs) [Haladjian et al. 2018b; Reyes-Ortiz et al. 2016]. Our exercise detection algorithm is based on a memory-efficient peak detection algorithm and a two-step recognition method. In a first step, our recognition method filters out most irrelevant motion patterns that correspond to passes (i.e. when goalkeepers pass the ball back to their trainer) by matching the signal to a pre-computed template using Dynamic Time Warping. Second, it uses a machine learning classifier to determine the actual exercise.

In the last few years, the community has started investigating the so-called end-to-end methods for activity recognition. These methods extract information directly from the raw data, thus, relieving developers from the tedious work to study data and manually implement and assess an activity recognition algorithm. Different Convolutional Neural Network (CNN) architectures have already surpassed hand-crafted methods in recognition performance [Ha and Choi 2016; Jiang and Yin 2015; Yang et al. 2015; Zeng et al. 2014]. Most notably, Recurrent Neural Networks (RNN) - mostly Long-Short-Term-Memory (LSTM) - have been adapted to exploit the time-dependency of the sensor signals, attaining the highest recognition results so far on large public benchmark datasets [Li et al. 2018; Murahari and Plötz 2018; Neverova et al. 2016; Ordóñez and Roggen 2016; Zeng et al. 2018]. We use a lightweight unobtrusive device that can be worn at the wrist by goalkeepers during a training session for up to 3 hours, or ideally longer. To satisfy these requirements, we designed our system based on a lightweight embedded device that performs a low-cost hand-crafted recognition algorithm locally.

3 METHODS

Figure 2 shows an overview of our recognition system. In this section, we describe the hardware device we used, how we collected the data and the computations we performed to detect and classify goalkeeper training exercises.

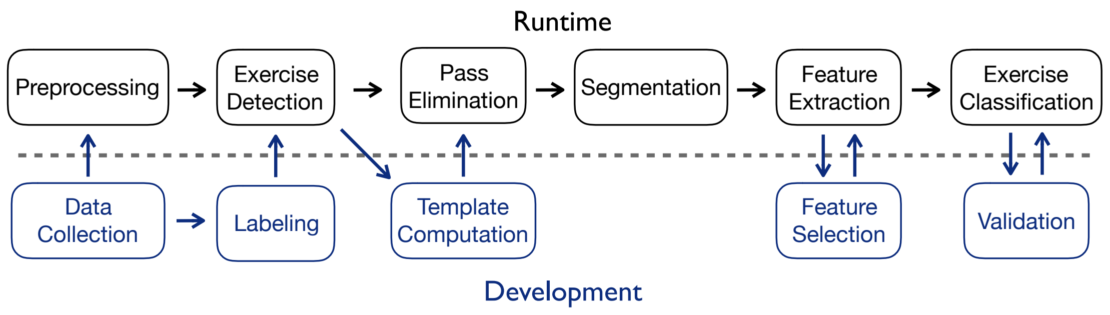
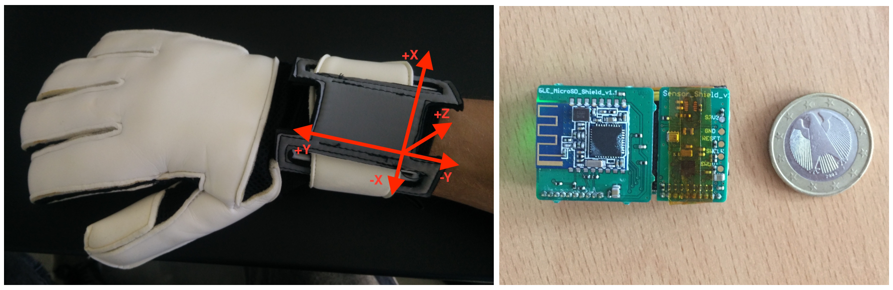


Fig. 2. Overview of the computations we perform to detect and classify goalkeeper training exercises. We separate between computations done at development and runtime.

3.1 Hardware

We collected data using a sensor device called *MicroHub* developed by the company InteractiveWear [1]. The MicroHub is a modular sensing device developed for wearable applications with an ARM Cortex-M0 microcontroller, Invensense's ICM20602 6-axis (accelerometer and gyroscope) Inertial Measurement Unit (IMU), a Bluetooth Low Energy (BLE) module and an SD card. The accelerometer measures acceleration forces in units of gravity *g*. Such forces may be static (e.g. the gravity) or dynamic (e.g. motion). The gyroscope measures angular velocity (i.e. the speed of rotation) in degrees per second (*dps*). The accelerometer and gyroscope were set to their maximum ranges: ±16 g and ±2000 dps, respectively. The data produced by both sensors was recorded with a 16 bit resolution and stored in an SD card mini with a capacity of 2 GB. This made it possible for us to store several hours of training at a sampling rate of 100 Hz without data loss or disconnections, as might have been the case with a wireless communication technology.

[1]http://www.interactive-wear.com/



(a) Glove (b) MicroHub

Fig. 3. Smart Glove components and orientation of the accelerometer axes. The x-axis represents side movements, the y-axis forward and backward movements and the z-axis up and down movements.

The Microhub was powered with a 200 mAh Li-Po battery. With this battery, the device remains functional for at least 6 hours while recording raw data continuously and storing it on the SD card. The device's dimensions (including the battery) are: 3.8 x 1.1 x 2.4 cm. To fit the device into a goalkeeper's glove, we designed a mount that is strapped tightly using the glove's strap above the wrist. Since our mount does not require modifications to the glove, it can be strapped around most goalkeeper gloves in the market. The MicroHub is oriented such that the x-axis represents side movements (i.e. to the left and right of the user's hand), the y-axis forward and backward movements (i.e. towards or against the fingers) and the z-axis up and down movements (assuming the palm of the hand is facing the ground). The glove, mount and sensor coordinate system are shown in Figure 3.

Table 1. Data collection plan followed with each goalkeeper. Dive Stand is a variation of a dive where the goalkeeper had to stand up quickly after the dive.

#	Exercise Type	Repetitions
1	Dive	x5 low left, x5 low right, x5 high left, x5 high right
2	Catch	x10 hand catch, x10 body catch and x10 ground catch
3	Dive Stand	x5 low left, x5 low right, x5 high left, x5 high right
4	Throw	x10 high, x10 low
5	Jump Catch	x8

3.2 Data Collection

We collected data from 14 goalkeeper trainees during their training with a professional goalkeeper trainer with over 30 years of experience. During the training, goalkeepers performed different variations of dives, throws and catches. The trainer threw balls at the goalkeepers from different angles and at different intensities. After performing the motion, goalkeepers passed the ball back at the trainer to signal that they were ready for the next repetition. Figure 1 shows goalkeepers executing these exercises during their regular training. The goalkeeper trainees we recorded were male, right handed, 10 to 17 years old, had 1 to 7 years of experience training as goalkeepers and were 1.5 to 2.01 meters tall. Goalkeepers first warmed up, stretched and then started the exercise execution. Table 1 summarizes the exercises performed by goalkeepers and the approximate amount of repetitions they performed. Each training session lasted on average 33 minutes, which led to a file size of approximately 3 MB per recording session.

3.3 Labeling

Each training session lasted approximately one hour. We video-recorded the entire training and annotated every repetition of the training exercises (e.g. dives, catches, throws) as well as other motions goalkeepers performed during the training, such as passes (i.e., when the goalkeeper passed the ball back to the trainer), sprinting, clapping with the hands and bouncing the ball. To synchronize the sensor signal and video recording, we asked goalkeepers to clap three times in the beginning, middle and at the end of the training session. The annotations were done by five individuals and were reviewed by one of the authors to ensure consistency. In total, we labeled 2562 motion instances (1518 training exercises and 1044 instances of other motion, such as passes and hand claps).

3.4 Exercise Detection

The *Exercise Detection* identifies possible exercises in an incoming stream of motion data. An overview of the different computations our detection algorithm performs is shown in Figure 4. We start by computing the squared magnitude of acceleration along the x, y and z-axes according to:

$$Magnitude^2 = a_x^2 + a_y^2 + a_z^2 \tag{1}$$

Most exercises cause a peak in the squared magnitude signal due to an impact with the ball or ground. These peaks occur suddenly (i.e., have a duration of 50 ms or less) and hence contain mostly motion in high frequency bands. Therefore, we filter out low-frequency motion using a first order Butterworth high-pass filter with a cutoff frequency $f_c = 25\ Hz$.

In the next step, we detect peaks in the filtered squared magnitude signal with a peak detection algorithm that executes for each new value. A new value becomes the peak candidate if it is above a *minimum peak threshold* η and if it is larger than the current peak candidate (in case the peak candidate is already set). Peak candidates become a peak after a *minimum amount of samples* δ elapse. We selected the parameter values $\eta = 8.2 \times 10^6\ g^2$ and $\delta = 97\ samples$ because they yielded the best detection performance, as we discuss in the Results section.

3.5 Pass Elimination

The peak detection method described in the previous subsection detects movements with a high intensity. A particular movement with high intensity performed frequently by goalkeepers is the pass, as they pass the ball back to the trainers after most exercise repetitions. Since passes are not an exercise that is of interest to goalkeepers, they need to be filtered out. While passes could be filtered out during the classification stage (i.e. by including them as a class in the machine learning classifier), doing so would require extracting features and performing a classification for each detected pass. The goal of the *Pass Elimination* procedure is to eliminate pass instances with a simple heuristic to avoid the more computationally expensive feature extraction and classification computations.

Most goalkeepers perform passes in a similar way: they sequentially swing the arm back to gain momentum and then forward and release the ball. This motion usually lasts approximately one second. Based on this observation, we designed a method to detect and filter out passes that does not require extracting features or running a prediction for

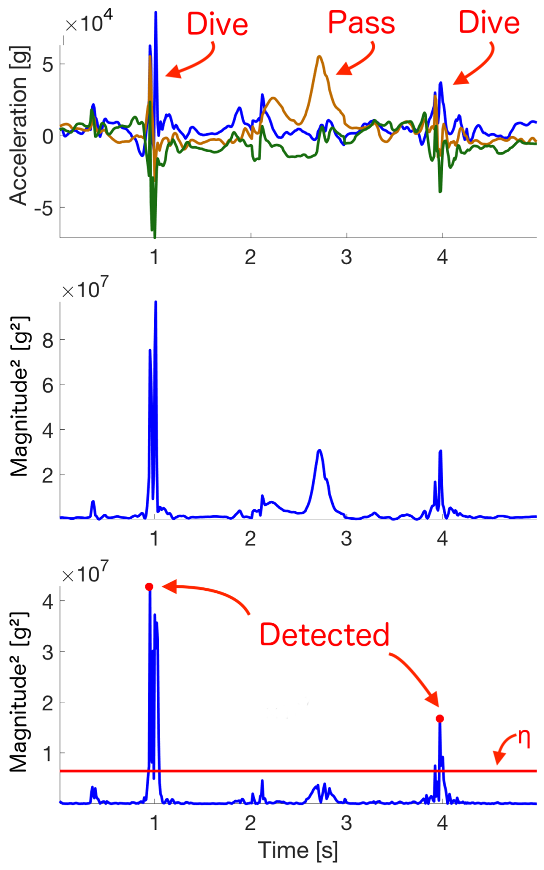


Fig. 4. Sequence of computations performed by our approach to detect exercises. Top: raw accelerometer signal for two dives and a pass. Middle: squared magnitude of the accelerometer signal. Bottom: high-pass filtered squared magnitude of accelerometer signal. Both dives are detected using a peak detector with a minimum peak height $\eta = 8.2 \times 10^6$.

each detected pass. It should be noted that we still include the pass in the different classifiers we studied under the *Other* class, as described in the Classification subsection.

Comparing two pass segments by accumulating the distances of each of their samples sequentially would not account for the fact that passes can vary in speed. As a consequence, this naive comparison procedure would compute a high distance for two similar passes whenever one of the passes was performed at a different speed. Dynamic Time Warping (DTW) is an algorithm that relies in dynamic programming to measure the similarity of two time series that might vary in speed. In particular, it finds the matching between samples in a segment to samples in another segment that lead to a minimal distance between segments.

Previous work in human activity recognition has successfully used DTW to identify human strides by first detecting strides with a peak detector and then comparing them to a stride template [Derawi et al. 2010] or by directly comparing consecutive segments to a stride template [Barth et al. 2015]. Similarly, our method compares the signal around a detected peak to a pass template and discards the detected peaks if the segment around them is similar enough to the template. Our procedure first creates a segment around the detected peak with index p according to:

$$sk_{pass} = [p - a', ..., p + b'] \tag{2}$$

We determined the offsets $a' = 70$ and $b' = 25$ empirically based on our observation that most peaks corresponding to passes occur approximately 70 samples after the beginning of a pass and that the motion corresponding to a pass lasts for approximately 95 samples. These segments contain only the acceleration along the y-axis, as we found this axis to represent best the swing performed during a pass. The DTW algorithm computes the distance between a pass segment and the pass template based on a cost metric to compare two samples. As a cost metric, we used the absolute difference in y-acceleration between the two samples.

Our procedure discards events if the segments around them have a DTW-distance to the pass template smaller than a threshold τ. We compared the performance of different values of τ (see section *Results*) and set $\tau = 39000$. The next subsection describes how we computed the pass template.

3.6 Template Computation

An ideal pass template S_T has a small distance to as many pass segments S_i and a large distance to as many non-pass segments as possible. Our template computation procedure selects as the template the pass segment with the minimal distance to the greatest number of other pass segments. We compute the template in three steps. The first step computes the distance between every pair of pass segments. To this end, we fill the matrix $M(i, j) \in R^{nxn}$ with the DTW-distances of each pass segment S_i to each other pass segment S_j according to:

$$M(i,j) = \begin{cases} DTW(S_i, S_j), i, j \in [1, ..., n], & \text{if } i \neq j \\ \infty, & \text{if } i = j \end{cases} \tag{3}$$

where n is the number of pass segments detected by the *Exercise Detection* stage. The distance between a segment to itself is set to ∞ because the second step counts how many *other* segments a segment is closest in distance to. Thus, setting $M(i, i) = \infty$ avoids counting a segment as having the closest distance to itself.

In the second step, we determine which segments are closest in distance to each other segment. To this end, we define the matrix $B \in R^{nxn}$ that contains a 1 at each position $B(i, j)$ if the segment S_i is closest in distance to the segment S_j, or a 0 otherwise. In other words, we set a 1 at $B(i, j)$ if $M(i, j)$ is the minimum of the j column in M:

$$B(i,j) = \begin{cases} 1, & \text{if } M(i,j) = min(M(k,j)), \forall k \in [1, ..., n] \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The third step counts how many segments a specific segment has minimal distance to. This is done by accumulating the values in each row of B into a vector C:

$$C(i) = \sum_{j=1}^{n} B(i,j), \forall i \in [1, ..., n] \tag{5}$$

Finally, we perform a voting method to decide which pass segment becomes the template. A naive method to select a pass template would let every segment *vote* for another pass segment (e.g. the pass segment with shortest distance) and select the pass segment with largest number of votes. However, this voting method might select an outlier pass segment as a template which is closest in distance to only a few other pass segments. In particular, this might happen if the pass segments are spread such that each of them are closest to different pass segment candidates without a clear *winner*.

Instead, we use the *Instant-runoff* voting method. In computational social choice theory, Instant-runoff is a method to select a single winner in an election with more than two candidates. The method lets voters rank every candidate in order of preference. If a candidate obtains the majority of the votes, it is elected as the winner. Otherwise, the candidate with least amount of votes is eliminated and the election is repeated with the remaining candidates. In our adapted version of the method, every pass segment is a voter and a candidate to become the template at the same time and pass templates are ranked based on the number of other segments they are closest to in DTW-distance. In each iteration, our algorithm eliminates the segment S_m with smallest distance to the *least* other segments: $\{m \mid C_m = \min(C_i), \forall i = 1...n)\}$ from the list of candidates. This procedure is repeated $n-2$ times. In each iteration, the matrix B and vector C are recomputed. After $n-2$ iterations, only one segment is left, which is selected as template. Figure 5 shows the template selected with this procedure on top of every other pass segment in our data set.

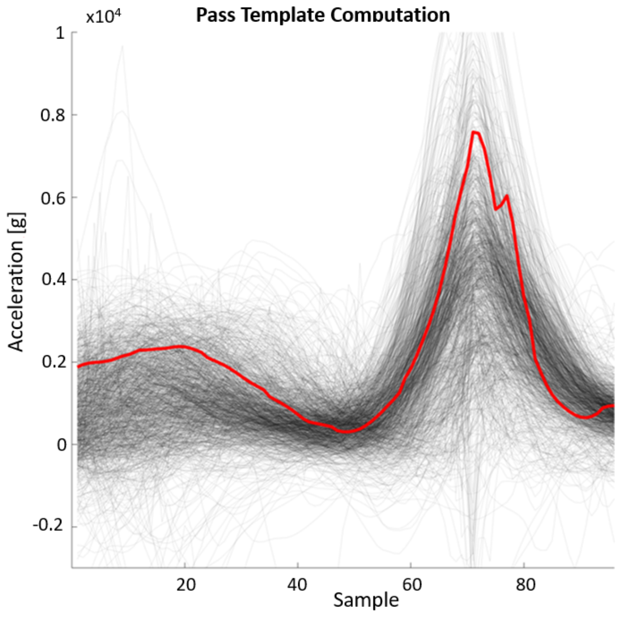


Fig. 5. Acceleration along the y-axis of every pass segment (black) and the template (red).

3.7 Segmentation

For each event that was not eliminated in the *Pass Elimination* procedure, we create a segment range k around the peak location x_p according to:

$$k = [x_{p-a}, ..., x_{p+b}] \tag{6}$$

We set $a = 130$ and $b = 90$ empirically based on a visual comparison of the motion produced by the different exercise instances, as shown in Figure 6. The *Segmentation* stage produces a segment $S(k)$ containing the acceleration, angular

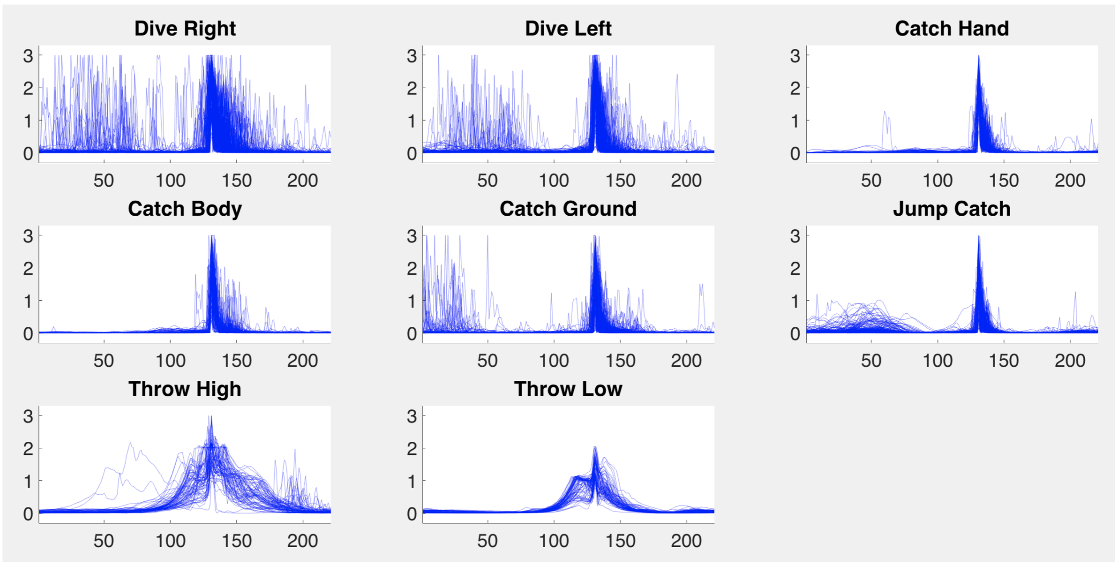


Fig. 6. Squared magnitude of acceleration of every segment produced by the segmentation algorithm plotted on top of each other and grouped by exercise.

velocity and squared magnitude of each sample in the range k. Figure 6 displays the segments produced by our algorithm after this step, which are used for feature extraction, as described in the next section.

3.8 Feature Extraction

We studied a set of statistical and heuristic features. The statistical features we chose have been previously used in similar wearable activity recognition applications [Blank et al. 2015; Bulling et al. 2014; Haladjian et al. 2017]. The list of features we studied is summarized in Table 2. The statistical features are computed on all three axes of acceleration and angular velocity. Heuristic features are computed on all three axes of the acceleration vector, all three axes of angular velocity and on the squared magnitude of acceleration computed with Equation 1.

We designed heuristic features to highlight the differences in motion across the different exercise variations. The peaks detected in the *Exercise Detection* are located at the moment of maximum intensity of acceleration (e.g., contact with the ground, contact with the ball, release of a ball). The different exercises have a different acceleration before the peak and deceleration after the peak along each axis. To capture the differences in acceleration and deceleration, we split the segment in three parts based on the position of the peak p as: $k_{left} = [1, .., p - c]$, $k_{center} = [p - c, ..., p + c]$ and $k_{right} = [p + c, ..., n]$ where n is the number of samples in the segment and c is a constant. We set $c = 20$ by observing the signals grouped by exercise, as shown in Figure 6. Figure 7 shows the mean acceleration along the z-axis of all three parts computed from every right and left dive in our data set.

Some classifiers calculate the distance between feature vectors. Features with a large range of values might contribute more to the distance than features with smaller ranges of values. Therefore, we normalize every feature f_i in the segment to have zero mean and a standard deviation in the range $[-1, 1]$ with:

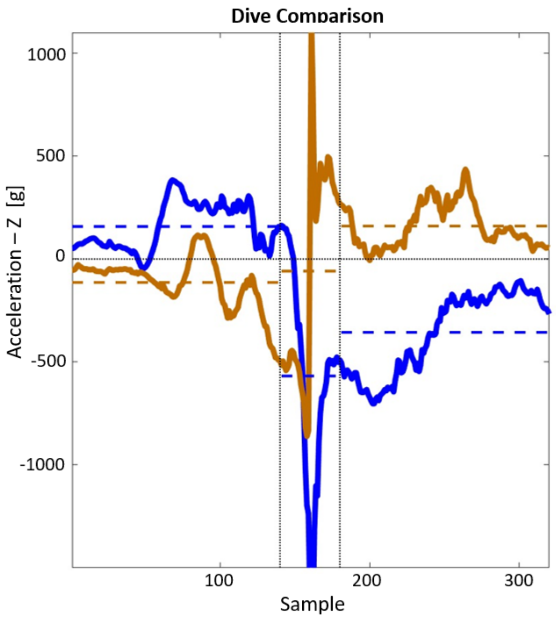


Fig. 7. Dive right (blue) and left (orange) computed by averaging the acceleration along the z-axis of every dive segment in our data set. Most right dives have a positive acceleration mean before the peak and negative during and after the peak. Most left dives have the opposite acceleration.

Table 2. List of features we studied. Statistical features are computed on the entire segment. Heuristic features are computed on each segment part: k_{left}, k_{middle} and k_{right}.

Type	Features
Statistical	mean, median, standard deviation (std), accumulated squared magnitude of acceleration, skewness, kurtosis, zero crossings (zcr), peak to peak amplitude (p2p), root mean square (rms), minimum value (min), maximum value (max), correlation between acceleration axes (corr)
Heuristic	mean, standard deviation (std), zero crossing, quantile, peaks

$$\bar{f_i} = \frac{f_i - \mu(f_i)}{\sigma(f_i)} \tag{7}$$

where $\mu(f_i)$ and $\sigma(f_i)$ are the mean and standard deviation of the feature f_i calculated based on every segment in our data set.

3.9 Feature Selection

Extracting and classifying a larger number of features leads to a larger number of computations that have to done on the wearable device and make machine learning classifiers more prone to overfitting. For this reason, we studied how to reduce the number of features with a feature selection algorithm called *minimum Redundancy Maximum Relevance* (mRMR) [Peng et al. 2005]. The 20 most relevant features selected by mRMR and the signals they are computed on are listed in Table 3.

Table 3. Features selected for classification using mRMR.

Feature	Computed on
Quantile	Az_3, Az_2, Az_4, Ay_2
Amount of Peaks	Squared Magnitude on k_{right}
Cross-correlation	Ay - Az
Mean (Segment)	Ay, Gz
Maximum	Gx, Ay
Root Mean Square	Gz, Gx
Standard Deviation	Ay, Az
Median Absolute Deviation	Ax, Gx
Median	Az
Mean	Az on k_{right}, Ay on k_{right}, Az on k_{middle}

3.10 Exercise Classification

We compared the performance of different classifiers: Naive Bayes, Decision Tree, Random Forest, Support Vector Machine (SVM) with linear and Radial Based Function (RBF) kernels, k-nearest neighbors (kNN) and a Neural Network. For each classifier, we optimized the performance of the following parameters:

- **SVM (RBF and linear kernel):** cost parameter $c \in \{1, 2, 3, 5, 8, 13, 21, 34, 55\}$, kernel coefficient $gamma \in \{0.0, 0.1, 0.2, ..., 10.0\}$, tolerance for stopping criterion $tol \in \{0.001, 0.002, 0.003, ..., 0.3\}$.
- **Decision Tree:** maximum depth $max\_depth \in \{1, 2, 3, ..., 30\}$.
- **Random Forest:** maximum depth $max\_depth \in \{1, 2, 3, ..., 30\}$, number of features to consider when looking for the best split $max\_features \in \{1, 2, 3, ..., 11\}$, minimum number of samples required to reach a leaf node $min\_samples\_leaf \in \{1, 2, 3, ..., 11\}$, minimum number of samples required to split an internal node $min\_samples\_split \in \{2, 3, 4, ..., 11\}$.
- **kNN:** $k \in \{1, 2, 3, ..., 15\}$.
- **Neural Network:** number of hidden layers $n \in \{1, 2\}$, size of hidden layers $hidden\_layer\_sizes \in \{10, 20, ..., 3000\}$, solver for weight optimization $solver \in \{lbfgs, adam\}$, tolerance for the optimization $tol \in \{10^{-2}, 10^{-3}, ..., 10^{-6}\}$.

3.11 Validation

To avoid evaluating the performance of the different methods we presented with the same data used to optimize the parameters η (minimum peak height), δ (minimum amount of samples between peaks), τ (DTW distance threshold) and the hyperparameters of our machine learning classifiers, we used a nested cross-validation procedure. In an outer loop, we tested different values for the parameters and the inner loop performed a leave-one-subject-out cross-validation. First, we found the optimal values for the parameters η and δ with a grid search in the ranges $[3 \times 10^6, 18 \times 10^6]$ and $[10, 200]$ at intervals of 10^5 and 1, respectively. Then, we optimized τ using a linear search in the range $[1 \times 10^2 - 2 \times 10^5]$ at intervals of 5×10^3. Finally, we optimized the hyperparameters of the classifiers with a randomized search followed by a grid search. The randomized search gave us optimal value ranges for each parameter and the grid search found the optimal values within each range. In each iteration of the inner loop, a different goalkeeper data file was excluded from the training set and used for assessing the performance of our methods. We selected the parameters that lead to the best average performance across player files.

4 RESULTS

Our algorithm first detects exercise instances from a stream of sensor values, and then classifies those detected events. Applications that rely on our algorithm would only use the final classifications. However, to provide insight about the different stages of our recognition pipeline, in this section we first report on the performance of the exercise detection method, and then discuss the classification method.

4.1 Exercise Detection

This subsection reports on the performance results for the computations we presented in the *Preprocessing*, *Exercise Detection* and *Pass Elimination* subsections. The detection of exercises can be understood as a binary classification (i.e. a segment is either classified as relevant or not relevant). To validate our exercise detection procedure, we compared the exercise labels to the detected segments as follows:

True Positive (TP). A detected segment that included a relevant exercise label.
False Positive (FP). A detected segment that did not include any relevant exercise label.
False Negative (FN). An exercise labeled as relevant that was not included in any of the detected segments.

Figure 8 shows a performance comparison of η and δ and Figure 9 presents the F-Score for different values of τ. Our detection method achieves an F-Score of 92.2%. In total, 1424 of the 1518 exercise repetitions were detected by our detection method (Recall = 93.8 %) whereas 148 instances of irrelevant movements were also detected as exercises (Precision = 90.6%).

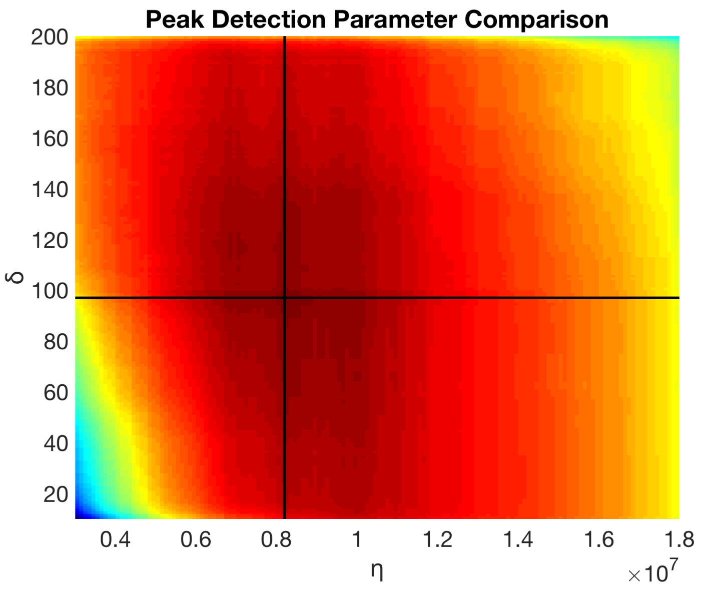


Fig. 8. Comparison of the F-Score for different values of η and δ. Red values indicate higher F-Scores. Line markers were placed at the values leading to the highest F-Score ($\eta = 8.2 \times 10^6$ and $\delta = 97$).

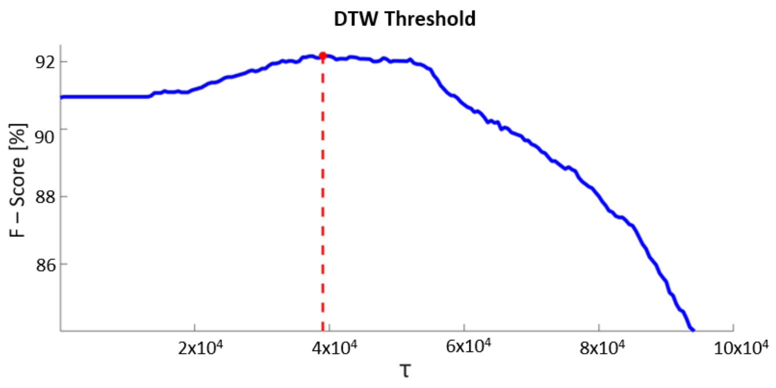


Fig. 9. F-Score of the exercise detection for different values of τ. $\tau = 39000$ achieves the highest F-Score: 92.2% (Precision 90.6%, Recall 93.8%) with the peak detection parameters: $\eta = 8.2 \times 10^6$ and $\delta = 97$.

The performance of our detection algorithm at detecting the different exercises is shown in Table 4.

Table 4. Amount of instances of each motion type detected by our approach and annotated in our data set.

	Exercise	Detected	Annotated	%
Relevant	Dive Right	285	286	99.7%
Relevant	Dive Left	248	251	98.8%
Relevant	Catch Hand	144	146	98.6%
Relevant	Catch Body	212	217	97.7%
Relevant	Catch Ground	145	158	91.8%
Relevant	Jump Catch	112	150	74.7%
Relevant	Throw High	141	145	97.2%
Relevant	Throw Low	137	165	83.0%
	Other	148	1044	14.2%

4.2 Exercise Classification

The segments detected as relevant by our exercise detection method are classified into the different exercises. Table 5 shows the accuracy of the different classification algorithms we tested. The classifier that achieved the highest accuracy is the SVM with an RBF Kernel and a box constraint $c = 7$ (accuracy: 96.5%). The confusion matrix and classification performance per exercise computed with this classifier are shown in Tables 6 and 7. The kNN classifier was tested with $k = 8$, which led to the maximum accuracy among the different variations of the kNN algorithm of 92.2%.

Table 5. Performance of different classifiers at classifying goalkeeper training exercises. SVM performed best with an RBF kernel and parameters: $c = 7$, $gamma = 0.0095$. $decision\_function\_shape = ovr$ and $tol = 0.001$

	Accuracy	Precision	Recall
Naive Bayes	74.4%	76.1%	76.8%
Decision Tree	77.9%	78.8%	79.1%
Random Forest	87.2%	88.8%	88.3%
SVM (Linear)	93.0%	94.6%	93.6%
SVM (RBF)	**96.5%**	**96.9%**	**96.7%**
K-Nearest Neighbors	92.1%	92.5%	93.8%
Neural Network	93.8%	93.9%	93.5%

Table 6. Confusion matrix computed using an SVM classifier with an RBF kernel and cost parameter $c = 7$.

		Annotated							
		DR	DL	CH	CB	CG	JC	TH	TL
Predicted	DR	271	12	2	0	0	0	0	0
	DL	11	236	0	0	0	1	0	0
	CH	2	0	141	0	0	1	0	0
	CB	0	0	1	207	3	1	0	0
	CG	1	1	0	8	135	0	0	0
	JC	0	1	2	0	0	108	1	0
	TH	0	0	0	0	0	0	140	1
	TL	0	0	0	0	0	0	1	136

Table 7. Accuracy for each training exercise computed using the SVM classifier with an RBF kernel and cost parameter $c = 7$.

	Precision	Recall	F-Score
Dive Right	95.1%	95.1%	95.1%
Dive Left	94.4%	95.2%	94.8%
Catch Hand	96.6%	97.9%	97.2%
Catch Body	96.3%	97.6%	97.0%
Catch Ground	97.8%	93.1%	95.4%
Jump Catch	97.3%	96.4%	96.9%
Throw High	98.6%	99.3%	98.9%
Throw Low	99.3%	99.3%	99.3%

5 DISCUSSION

The results we presented indicate that it is possible to detect and classify goalkeeper training exercises accurately. Our detection approach detects 93.8% of the relevant exercise instances with a precision of 90.6% and the SVM classifier with an RBF kernel achieves a classification accuracy of 96.5%.

We focused our analysis on the most common variations of the goalkeeper training exercises performed during a training session. In a free-training (i.e. soccer match), the amount of relevant exercises might be considerably sparser and goalkeepers might perform other exercises such as sprints, kicking the ball with the leg or warm up exercises which we did not consider in this study. Therefore, we expect a lower precision in a free-training than the one we presented in this study. In the future, the wider variability of a free-training should be studied to identify new classes and train a classifier accordingly.

Our detection approach detected over 97% of the instances of most exercise types and filtered out most of the irrelevant motions. Only 148 instances of irrelevant motion were wrongly detected as relevant exercises, which is a small fraction (14.2%) of the total amount of instances of irrelevant motion we annotated (1044) and an even smaller fraction of every irrelevant motion goalkeepers performed during the training session which we did not annotate (e.g., walking, running, picking up a balls from the ground, receiving balls passed at them without any specific technique, etc.). On the other hand, only 74.2% of the jump catches and 83.0% of the low throws were detected. The reason is that these exercises contain low frequency motion that is attenuated by the high-pass filter, which causes the peak detector to miss the peaks produced by some of these exercise instances. These exercises could be detected by lowering the peak threshold η, but this would also increase the false positive detection rate. The SVM classifier with an RBF kernel produced the highest accuracy of 96.5%.

Figure 9 shows that without our *Pass Elimination* procedure ($\tau = 0$), the detection procedure achieves an F-Score of 90.9% and that our *Pass Elimination* procedure improves this performance to 92.2% when $\tau = 39000$. Besides the improvement in recognition performance, an important benefit of this procedure is that every time a pass is eliminated, no features have to be extracted, stored in flash memory (or transmitted wirelessly) and classified, resulting in a lower energy consumption. The procedure can detect and eliminate more pass instances by increasing the template distance threshold τ. However, increasing τ over a value of 39000 leads to the elimination of more relevant exercise instances than irrelevant ones. The main reason for this is that low throws have a similar signature to passes, causing low throws executed by players with a low intensity to be incorrectly matched to the pass template. Nevertheless, the procedure we described can be used as-is in other applications if a particular class has a consistent signature that is different enough from other classes. Furthermore, future work could study different template selection strategies. In particular, we have selected our template based on its similarity to other instances of the same class. In the future, the distances to other classes could also be used as part of the optimization metric to select a template. Furthermore, additional signals (besides only the y-axis of the accelerometer vector) could be used to compare segments to the template and different amounts of templates could be selected per class. Finally, future work could study the recognition and computational performance when using the distances to different templates as features within a machine learning classifier.

A possible threat to validity to our evaluation procedure is that we haven't used a test split to assess the performance of the *Pass Elimination* method. As we assessed the performance of the *Pass Elimination* method with the same data we used to select the pass template, the performance we presented might be slightly higher than on unseen data. In the future, the collection of an additional data set will make it possible to select a pass template and assess the performance of a *Pass Elimination* method with different data.

In order to enable further data analysis and to improve our recognition algorithms by means of online machine learning, it would be convenient to have access to the feature vectors produced by our algorithm outside of the wearable device. Since streaming feature vectors would require a mobile phone nearby during a training session, we decided to store the feature vectors on the wearable device and transmit them to a mobile device after the training over Bluetooth Low Energy. Since we set a minimal distance between peaks of 97 samples (0,97 seconds), at most 3711 motion instances (relevant or irrelevant) can be detected by our approach during an hour of training. The features and a timestamp for every detected motion instance could be stored in less than 304 kB memory (3711 instances x 21 features and timestamp x 4 bytes per feature and timestamp). Furthermore, our detection algorithm detected a total of 1572 motion instances in our data set (computed by aggregating the values in the *Detected* column in Table 4). As our data set consists of 14 training sessions, our algorithm would detect an average of 112 motion instances per training session, which could be stored in approximately 9,6 kB memory (112 instances x 21 features and timestamp represented with 4 bytes each).

6 CONCLUSIONS

We have described a method to detect and classify goalkeeper training exercises. Detecting and classifying these exercises represents a first step towards a virtual coach that gives goalkeepers relevant and objective feedback about their exercise executions. A major challenge we addressed in this paper was on detecting the relevant training exercises while avoiding other movements goalkeepers perform during a training session. We noticed that specially younger goalkeepers tend to perform hectic movements that lead to a larger amount of false positive detections. We showed that a careful preprocessing of the data using a high-pass filter in combination with Dynamic Time Warping can be used to reduce the number of false positive detections. Furthermore, we achieved an exercise detection recall of 93.8% (precision: 90.6%) and a classification accuracy of 96.5%. Our segmentation approach and the methods we used to extract heuristic features and automatically select the most relevant ones can be reused in other applications that require detecting or classifying specific events such as strokes or shots in ball-based sports or strides in gait analysis applications.

Our work is by no means finished. An immediate next step is to investigate what performance metrics are relevant to goalkeepers and which of those can be extracted accurately with our smart glove. Based on our discussions with coaches and experienced goalkeepers, relevant performance metrics include: '*the area of the goal where most balls are missed or caught*', '*the time needed to stand up on the feet after a dive*' and '*the maximum height and length achieved during a dive*'. Future work should study how to compute different performance metrics with the segments detected and classified by our algorithm.

Before goalkeepers are able to benefit from the system we propose, their needs as well as possible interaction modalities for a user interface will have to be investigated. Some open questions are: What information are goalkeepers interested in and how should this information be provided to them? Should the system provide goalkeepers live feedback while in the field or should the feedback be given offline (e.g. while in the locker room or at home)? To keep the device lightweight while exploiting the fact that most users nowadays have a smartphone, similar systems offer users an interface over a smartphone. However, goalkeepers don't always take their smartphone to the field and even if they did they would have to take off their gloves to operate it via touch. An alternative would be a voice-based interaction, but a soccer pitch is loud so goalkeepers would have to get close to the phone. Another question that raises is: where could a smartphone be placed (e.g. behind the goal) such that it does not disturb the goalkeeper and cannot be hit by a ball?

Furthermore, a larger data set is likely to improve the accuracy and reveal corner cases. As we collect more data, we might discover new irrelevant motions performed frequently by players (e.g. as we realized previously that some players bounce the ball before a throw). These motions need to be considered by our algorithm (e.g. included in the classifier). Furthermore, future work should add support for left-handed goalkeepers, which we did not consider in this study. It might also be interesting to investigate how our recognition method could adapt itself based on data generated after its deployment to the market (e.g. with online machine learning or with a user-dependent recognition).

We are currently investigating the suitability of a computer vision approach to recognize training exercises and extract performance metrics from them. So far, we successfully used a pre-trained pose estimation algorithm to obtain the body joints (wrists, hips, knees) of the goalkeeper from an image frame. These joints could be combined with a machine learning algorithm to extract different exercise execution metrics. We believe that a computer vision approach has a higher potential to accurately recognize exercise performance metrics relevant to goalkeepers. At the same time, the approach would pose other challenges such as occlusion (e.g. a coach walking in front of the camera), confusion when multiple players are detected in an image frame and would impose a higher degree of involvement from goalkeepers to setup the camera in the field.

REFERENCES

Stefano Abbate, Marco Avvenuti, Francesco Bonatesta, Guglielmo Cola, Paolo Corsini, and Alessio Vecchio. 2012. A smartphone-based fall detection system. *Pervasive and Mobile Computing* 8, 6 (2012), 883–899.

Oliver Amft. 2010. A wearable earpad sensor for chewing monitoring. In *SENSORS, 2010 IEEE*. IEEE, 222–227.

Oliver Amft, Holger Junker, and Gerhard Troster. 2005. Detection of eating and drinking arm gestures using inertial body-worn sensors. In *Ninth IEEE International Symposium on Wearable Computers (ISWC'05)*. IEEE, 160–163.

Marc Bächlin, Kilian Förster, and Gerhard Tröster. 2009. SwimMaster: a wearable assistant for swimmer. In *Proceedings of the 11th international conference on Ubiquitous computing*. ACM, 215–224.

Ling Bao and Stephen S Intille. 2004. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*. Springer, 1–17.

Jens Barth, Cäcilia Oberndorfer, Cristian Pasluosta, Samuel Schülein, Heiko Gassner, Samuel Reinfelder, Patrick Kugler, Dominik Schuldhaus, Jürgen Winkler, Jochen Klucken, and Others. 2015. Stride segmentation during free walk movements using multi-dimensional subsequence dynamic time warping on inertial sensor data. *Sensors* 15, 3 (2015), 6419–6440.

Peter Blank, Julian Hoßbach, Dominik Schuldhaus, and Bjoern M Eskofier. 2015. Sensor-based stroke detection and stroke type classification in table tennis. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 93–100.

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

Jay Chen, Karric Kwong, Dennis Chang, Jerry Luk, and Ruzena Bajcsy. 2006. Wearable sensors for reliable fall detection. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. IEEE, 3551–3554.

Mohammad O Derawi, Patrick Bours, and Kjetil Holien. 2010. Improved cycle detection for accelerometer based gait authentication. In *2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE, 312–317.

Jessica Echterhoff, Juan Haladjian, and Bernd Brügge. 2018a. Gait Analysis in Horse Sports. In *Proceedings of the Fifth International Conference on Animal-Computer Interaction*. ACM, 3.

Jessica Echterhoff, Juan Haladjian, and Bernd Brügge. 2018b. Gait and Jump Classification in Modern Equestrian Sports. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, 88–91.

Pascual J Figueroa, Neucimar J Leite, and Ricardo M L Barros. 2006. Tracking soccer players aiming their kinematical motion analysis. *Computer Vision and Image Understanding* 101, 2 (2006), 122–135.

Benjamin H Groh, Martin Fleckenstein, Thomas Kautz, and Bjoern M Eskofier. 2017a. Classification and visualization of skateboard tricks using wearable sensors. *Pervasive and Mobile Computing* 40 (2017), 42–55.

Benjamin H Groh, Frank Warschun, Martin Deininger, Thomas Kautz, Christine Martindale, and Bjoern M Eskofier. 2017b. Automated Ski Velocity and Jump Length Determination in Ski Jumping Based on Unobtrusive and Wearable Sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 53.

Sojeong Ha and Seungjin Choi. 2016. Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 381–388.

Juan Haladjian, Katharina Bredies, and Bernd Bruegge. 2018a. KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries. In *Proceedings of the 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. IEEE, 9–12.

Juan Haladjian, Johannes Haug, Stefan Nüske, and Bernd Bruegge. 2018b. A Wearable Sensor System for Lameness Detection in Dairy Cattle. *Multimodal Technologies and Interaction* 2, 2 (2018), 27.

Juan Haladjian, Zardosht Hodaie, Stefan Nüske, and Bernd Brügge. 2017. Gait Anomaly Detection in Dairy Cattle. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction (ACI2017)*. ACM, New York, NY, USA, 8:1—-8:8. https://doi.org/10.1145/3152130.3152135

Juan Haladjian, Zardosht Hodaie, Han Xu, Mertcan Yigin, Bernd Bruegge, Markus Fink, and Juergen Hoeher. 2015. KneeHapp: A Bandage for Rehabilitation of Knee Injuries. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 181–184.

H M Sajjad Hossain, Md Abdullah Al Hafiz Khan, and Nirmalya Roy. 2017. SoccerMate: A personal soccer attribute profiler using wearables. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*. IEEE, 164–169.

Wenchao Jiang and Zhaozheng Yin. 2015. Human activity recognition using wearable sensors by deep convolutional neural networks. In *Proceedings of the 23rd ACM international conference on Multimedia*. Acm, 1307–1310.

Aftab Khan, James Nicholson, and Thomas Plötz. 2017. Activity Recognition for Quality Assessment of Batting Shots in Cricket using a Hierarchical Representation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 62.

Frédéric Li, Kimiaki Shirahama, Muhammad Nisar, Lukas Köping, and Marcin Grzegorzek. 2018. Comparison of feature learning methods for human activity recognition using wearable sensors. *Sensors* 18, 2 (2018), 679.

Zhen Li, Zhiqiang Wei, Yaofeng Yue, Hao Wang, Wenyan Jia, Lora E Burke, Thomas Baranowski, and Mingui Sun. 2015. An adaptive hidden markov model for activity recognition based on a wearable multi-sensor device. *Journal of medical systems* 39, 5 (2015), 57.

Benoit Mariani, Mayté Castro Jiménez, François J G Vingerhoets, and Kamiar Aminian. 2013. On-shoe wearable sensors for gait and turning assessment of patients with Parkinson's disease. *IEEE transactions on biomedical engineering* 60, 1 (2013), 155–158.

C F Martindale, M Strauss, H Gaßner, J List, M Müller, J Klucken, Z Kohl, and B M Eskofier. 2017. Segmentation of gait sequences using inertial sensor data in hereditary spastic paraplegia. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).* 1266–1269. https://doi.org/10.1109/EMBC.2017.8037062

Bruno Müller Junior and Ricardo de Oliveira Anido. 2004. Distributed real-time soccer tracking. In *Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks.* ACM, 97–103.

Vishvak S Murahari and Thomas Plötz. 2018. On attention models for human activity recognition. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers.* ACM, 100–103.

Rossana Muscillo, Silvia Conforto, Maurizio Schmid, Paolo Caselli, and Tommaso D'Alessio. 2007. Classification of motor activities through derivative dynamic time warping applied on accelerometer data. In *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* IEEE, 4930–4933.

Natalia Neverova, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham Taylor. 2016. Learning human identity from motion patterns. *IEEE Access* 4 (2016), 1810–1820.

Francisco Ordóñez and Daniel Roggen. 2016. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors* 16, 1 (2016), 115.

S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, D. Standaert, M. Akay, J. Dy, M. Welsh, and P. Bonato. 2009. Monitoring Motor Fluctuations in Patients With Parkinson #x0027;s Disease Using Wearable Sensors. *IEEE Transactions on Information Technology in Biomedicine* 13, 6 (nov 2009), 864–873. https://doi.org/10.1109/TITB.2009.2033471

Shyamal Patel, Delsey Sherrill, Richard Hughes, Todd Hester, Theresa Lie-Nemeth, Paolo Bonato, David Standaert, and Nancy Huggins. 2006. Analysis of the Severity of Dyskinesia in Patients with Parkinson's Disease via Wearable Sensors. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06).* IEEE, 123–126. https://doi.org/10.1109/BSN.2006.10

Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence* 27, 8 (2005), 1226–1238.

Guillaume Plouffe and Ana-Maria Cretu. 2015. Static and dynamic hand gesture recognition in depth data using dynamic time warping. *IEEE transactions on instrumentation and measurement* 65, 2 (2015), 305–316.

Jorge-L Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. 2016. Transition-aware human activity recognition using smartphones. *Neurocomputing* 171 (2016), 754–767.

Giovanni Schiboni and Oliver Amft. 2018. Sparse natural gesture spotting in free living to monitor drinking with wrist-worn inertial sensors. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers.* ACM, 140–147.

Dominik Schuldhaus, Carolin Jakob, Constantin Zwick, Harald Koerger, and Bjoern M Eskofier. 2016. Your personal movie producer: generating highlight videos in soccer using wearables. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers.* ACM, 80–83.

S Seto, W Zhang, and Y Zhou. 2015. Multivariate Time Series Classification Using Dynamic Time Warping Template Selection for Human Activity Recognition. In *2015 IEEE Symposium Series on Computational Intelligence.* 1399–1406. https://doi.org/10.1109/SSCI.2015.199

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

Emmanuel Munguia Tapia, Stephen S Intille, and Kent Larson. 2004. Activity recognition in the home using simple and ubiquitous sensors. In *International conference on pervasive computing.* Springer, 158–175.

Robin Thompson, Ilias Kyriazakis, Amey Holden, Patrick Olivier, and Thomas Plötz. 2015. Dancing with horses: automated quality feedback for dressage riders. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing.* ACM, 325–336.

Emily Walton, Christy Casey, Jurgen Mitsch, Jorge A Vázquez-Diosdado, Juan Yan, Tania Dottorini, Keith A Ellis, Anthony Winterlich, and Jasmeet Kaler. 2018. Evaluation of sampling frequency, window size and sensor position for classification of sheep behaviour. *Royal Society open science* 5, 2 (2018), 171442.

Yehuda Weizman and Franz Konstantin Fuss. 2015. Sensor Array Design and Development of Smart Sensing System for Kick Force Visualization in Soccer. *Procedia Technology* 20 (2015), 138–143.

Disheng Yang, Jian Tang, Yang Huang, Chao Xu, Jinyang Li, Liang Hu, Guobin Shen, Chieh-Jan Mike Liang, and Hengchang Liu. 2017. TennisMaster: an IMU-based online serve performance evaluation system. In *Proceedings of the 8th Augmented Human International Conference.* ACM, 17.

Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. 2015. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-Fourth International Joint Conference on Artificial Intelligence.*

Ming Zeng, Haoxiang Gao, Tong Yu, Ole J Mengshoel, Helge Langseth, Ian Lane, and Xiaobing Liu. 2018. Understanding and improving recurrent networks for human activity recognition by continuous attention. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers.* ACM, 56–63.

Ming Zeng, Le T Nguyen, Bo Yu, Ole J Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. 2014. Convolutional neural networks for human activity recognition using mobile sensors. In *6th International Conference on Mobile Computing, Applications and Services.* IEEE, 197–205.

Bo Zhou, Harald Koerger, Markus Wirth, Constantin Zwick, Christine Martindale, Heber Cruz, Bjoern Eskofier, and Paul Lukowicz. 2016. Smart soccer shoe: monitoring foot-ball interaction with shoe integrated textile pressure sensor matrix. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers.* ACM, 64–71.

6.3 Teaching wearable device development with the wearables development toolkit

This publication describes the formative process we followed to develop the WDK and how the WDK has been used for teaching purposes in a university context.

The author of this Habilitation developed the WDK, studied its usage by students during the development of activity recognition applications they worked on as part of their theses and wrote the paper.

Authors	Haladjian, J., & Bruegge, B.
Workshop	CEUR Workshop Proceedings
Number of Pages	2
Type	Short Paper
Review	Peer Reviewed (4 Reviewers)
Year	2019
Link	http://ceur-ws.org/Vol-2308/

Teaching Wearable Device Development with the Wearables Development Toolkit

1st Juan Haladjian
Chair for Applied Software Engineering
Technical University of Munich)
Munich, Germany
haladjia@in.tum.de

2nd Bernd Bruegge
Chair for Applied Software Engineering
Technical University of Munich)
Munich, Germany
bruegge@in.tum.de

CEUR-WS.org/Vol-2308/isee2019paper06.pdf

***Abstract*—This paper introduces the Wearables Development Toolkit (WDK), a set of tools to support the development of wearable device applications. It lowers the entrance barrier to wearable device development. We discuss our experiences in leveraging the WDK to teach wearable device development to students of computer science.**

***Index Terms*—software engineering, wearable computing, wearables development toolkit**

I. Introduction

The growing number of students of computer science calls for new teaching methodologies that are able to cope with larger number of students while maintaining the teaching quality. Digital technology has made it possible to transmit content to large numbers of students. However, actively engaging students in the learning process remains a challenge. Courses where students actively work on a project are usually associated with high supervision costs and therefore scale less well to larger number of students. A particular challenge for instructors is combining their teaching and research responsibilities.

Our research focus is on wearable computing. We develop wearable device applications, which make use of sensor data to extract relevant information from the user or her context. For example, we developed a wearable device that uses a motion sensor to detect lameness in dairy cattle [1], [2]. The development of wearable device applications requires multidisciplinary highly-specialized knowledge (e.g. electrical engineering, computer and data science).

The WDK is a collection of wearable sensors (see Figure 1) and tools to facilitate the development process of wearable device applications. The toolkit is meant to guide students during the development process and to enable them to study possible design solutions while saving time in implementation details. In this paper, we discuss our experience in teaching wearable application development to students of computer science using the WDK.

II. Wearables Development Toolkit

Figure 2 shows the different activities in the development process of a wearable device application. Most wearable device applications extract information from sensor data. The development process usually starts with the collection and annotation of data. After this, developers usually develop and evaluate different signal processing and machine learning methods. Finally, the application is deployed into the actual device. The WDK consists of four components: the *Wearable Sensors Platform*, the *Data Annotation Tool*, the *Visualization tool* and the *Evaluation tool*.

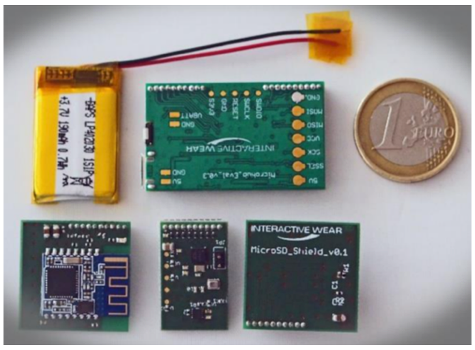


Fig. 1. Wearable sensor kit developed by InteractiveWear. Source: http://www.interactive-wear.com/

The *Wearable Sensors Platform* is a collection of wearable sensors which can be plugged into a sensor hub and configured over an iPhone App. This enables users to collect data without having to design or assemble a new sensor or to develop a firmware that stores data. Figure 1 shows the sensor hub and different hardware components. The *Data Annotation Tool* is used to automatize the data annotation process by synchronizing and displaying the sensor data together with reference markers enabling the user to annotate events in the time series signal. Figure 3 shows the *Data Annotation Tool*. The *Visualization tool* enables users to understand the sensor data as well as the effects different signal processing methods have on the data, which is critical for most activity recognition application. The *Evaluation tool* lets users quickly configure a chain of computations in order to assess its performance. The WDK is open source[1].

[1] https://github.com/avenix/WDK

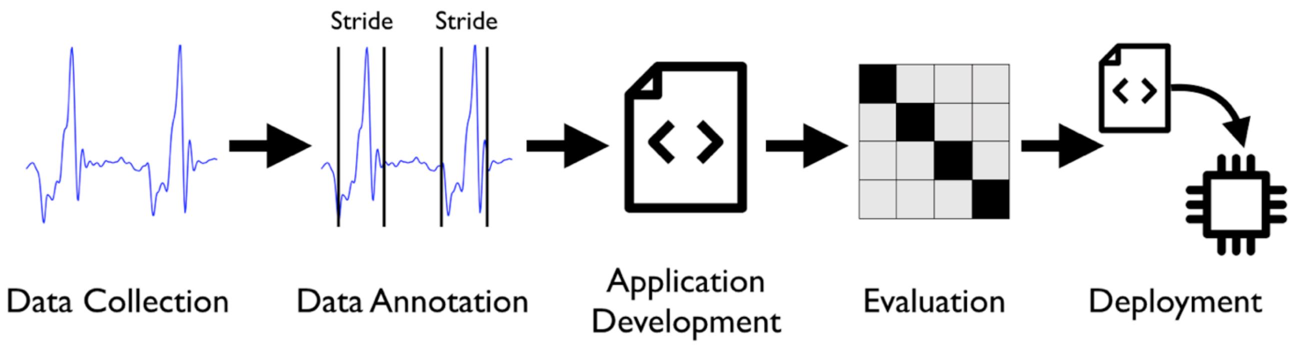


Fig. 2. Wearable device development activities.

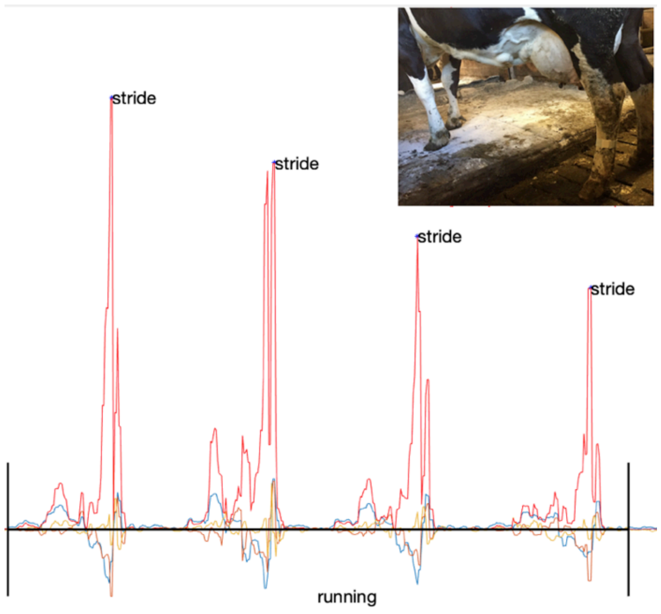


Fig. 3. Data Annotation Tool displaying the acceleration collected by a Inertial Measurement Unit attached to a limb of a cow. The strides performed by the cow have been annotated.

III. Teaching Method

Since 2011, we have supervised over 30 Bachelor's and Master's theses in computer science at the Technical University of Munich. Most of these theses comprise the development of a wearable device application or a feature of the WDK itself. In this section, we describe how we leverage the number of students to contribute to our research in wearable device development.

In the beginning of each semester, we provide students a tutorial on activity recognition with wearable devices[2]. Students are usually able to finish this tutorial within a day. Afterwards, the students start working on a particular

[2]Tuorial on activity recognition with wearable devices: https://github.com/avenix/ARC-Tutorial/

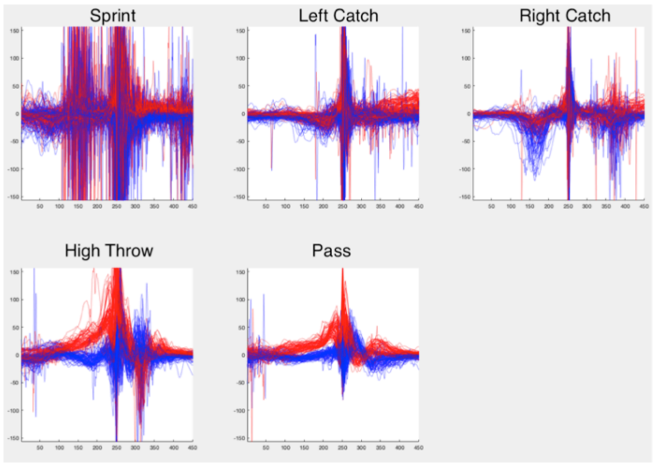


Fig. 4. Data Visualization Tool displaying the acceleration of a lacrosse goalkeeper while performing several training exercises.

application. As students start engaging in activities for which the WDK can spare them time, their instructor demonstrates the relevant tools within the toolkit. As students use the WDK, new requirements for the toolkit are identified, which are usually analyzed by an instructor and implemented by other students in the next term.

IV. Conclusions

The WDK enables students to reuse functionality and focus on the novel aspects of their projects. The different tools and documentation guide students through the development process, thus relieving instructors.

References

[1] J. Haladjian, J. Haug, S. Nüske, and B. Bruegge, "A Wearable Sensor System for Lameness Detection in Dairy Cattle," *Multimodal Technologies and Interaction*, vol. 2, no. 2, p. 27, 2018.

[2] J. Haladjian, Z. Hodaie, S. Nüske, and B. Brügge, "Gait Anomaly Detection in Dairy Cattle," in *Proceedings of the Fourth International Conference on Animal-Computer Interaction*, ser. ACI2017. New York, NY, USA: ACM, 2017, pp. 8:1—-8:8. [Online]. Available: http://doi.acm.org/10.1145/3152130.3152135

6.4 Gait Analysis in Horse Sports

This publication describes an activity recognition application to characterize the gait of horses during showjumping sports. This work builds on top of our work described in Section 6.5, where we presented an approach to recognize gait jumps and gait types. In this publication, we describe an algorithm to detect the length of a stride and jump and present a user interface for riders.

The author of this Habilitation supervised Jessica Echterhoff throughout the development of the algorithm presented in the paper including the data collection, the development of the recognition algorithm and writing the paper.

Authors	Echterhoff, J., Haladjian, J., & Bruegge, B.
Conference	Fifth International Conference on Animal-Computer Interaction
Number of Pages	6
Type	Short Paper
Review	Peer Reviewed (4 Reviewers)
Year	2018
DOI	https://doi.org/10.1145/3295598.3295601

Gait Analysis in Horse Sports

Jessica Maria Echterhoff
Technical University Munich
Munich, Germany
contact@jessicaechterhoff.com

Juan Haladjian
Technical University Munich
Munich, Germany
haladjia@in.tum.de

Bernd Brügge
Technical University Munich
Munich, Germany
bruegge@in.tum.de

ABSTRACT

In modern showjumping and cross-country riding, horse-rider-pairs have to jump a series of obstacles in a given time. A jump is considered successful (penalty-free) if a horse can comfortably jump the fence without elements of the fence falling down. If any of the elements of the fence falls down or the horse refuses to jump, the rider obtains penalty points or can be disqualified from the competition. An unsuccessful jump can lead to injury and loss in trust of the rider. The success of a jump is determined by the number, length and harmony of strides a horse performs before a fence. We propose a system for tracking horse strides and jumps using a smartphone attached to the horse's saddle.

Our system detects and segments individual strides and computes the length of a stride using signal processing and machine learning methods. We collected data from 9 horses who performed several jumps. Our results indicate that our system can detect horse strides with a precision of 96.3%, a recall of 95.7% and a pearson correlation of 0.73 with respect to our ground truth data set. We further describe a method to characterise the canter gait of the horse. Our system is intended to be used by riders to adapt their training and competition strategies to the physical limitations of the horse. The rider can thus prevent accidents due to an overtaxing of the horse or miscalculation of canter strides by the rider.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: Miscellaneous

Author Keywords
Wearable Sensing; Showjumping; Cross-Country; Motion Characteristics; Activity Recognition

INTRODUCTION

Horses served humans for the entirety of modern history. They helped humans to carry weights, farm fields and transport people to different locations. In the modern western culture, most heavy work is done by cars, trucks and agricultural tractors. Humans and horses can thus enjoy athletic and leisure time together, which contributes to a positive life quality of the two companions. Showjumping and cross-country riding are sports based on a deep confidence between the rider and the horse and strengthen the bonding between them.

Showjumping as well cross-country riding require a rider and a horse to jump over a course of fences in a specific order within a given time. The order of fences, course length and optimal speed is given by the course designer and known before riding the course. Inside the course, fences are placed separately or in close distance to each other. The horse has to make a specific amount of strides between fences in order to perform a successful jump.

We propose a system to keep track of gait characteristics and present statistics of the ridden course to the rider with its details for specific canter strides made before each jump. We study the horse's motion signal collected with a sensor on the horse's body to obtain canter characteristics for showjumping and cross-country riding. Our system assesses the stride length to measure the equality of canter strides between the jumps. The system provides the average stride length of each horse to the rider to enable planning a strategy for the course and measures the horse's ability to reduce or extend the stride length. Reducing the stride length means that the horse shortens the stride - it collects itself to cover less ground in the same rhythm. Extending the stride length means that the horse's rhythm stays the same but the horse stretches itself to cover more ground. The variability in stride length is an indicator of the physical condition and training status of the horse. For example, if a sequence of jumps is designed to have the horse put two strides between them, but the horse's stride is too short, it will add a small extra stride. The horse will subsequently take off too close to the second element of the jump combination. Because the takeoff spot is too close to the jump, the horse risks knocking the poles, or worse, tripping over the jump and possibly even falling. Therefore, knowing the correct number of strides between jump elements is not just a performance issue, but a safety issue. Our system supports riders in decisions regarding the right amount of strides to make in order to perform more successful jumps and decrease the probability of severe injury of the horse.

BACKGROUND

An average sport horse is able to move in three different gaits - walk, trot and canter, which differ in the intensity of speed, frequency, and vertical swing. We define the frequency of the horses gait as the amount of steps made in a specific amount of time and the vertical swing as the up-and-down movement of the horse's back. Walk is four-beat movement, trot a two-beat movement with diagonal pairs simultaneously on the ground

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACI18, December 4–6, 2018, Atlanta, GA, USA
© 2018 ACM. ISBN 978-1-4503-6219-1/18/12...$15.00
DOI: https://doi.org/10.1145/3295598.3295601

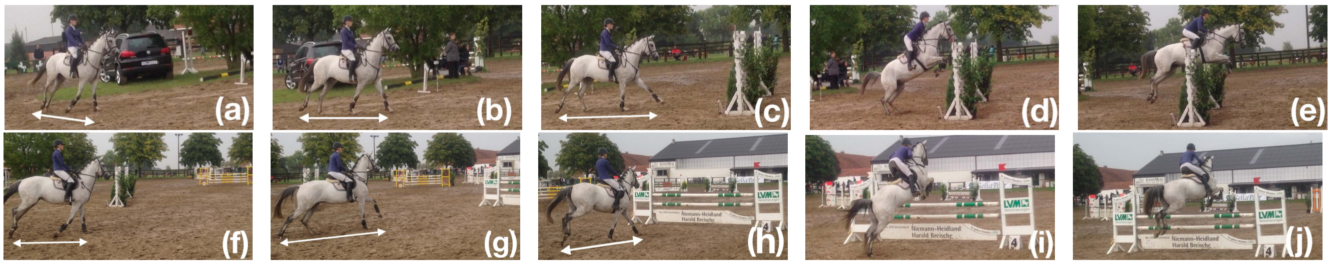


Figure 1. Horse performing three equal canter strides before a jump. Canter strides (a), (b), (c) are of equal length and lead to a successful takeoff point (d) and jump (e). Same horse performing three canter strides before a different jump. Canter strides (g), (h) are significantly longer than (f) and lead to a takeoff point to far from the fence (i) and a jump, where the horse has to distort its body to jump the fence (j).

and canter is a three-beat movement. One canter stride is a full sequence of a hind foot strike, diagonal pair and a front foot strike in the 3-beat rhythm. Horses typically jump all fences in a course running towards them in canter. While it is possible to jump fences when trotting, it is not common in competition and training scenarios.

Riders usually walk the course beforehand to measure distances between the fences and determine the amount of strides to make between them. To successfully find a strategy for riding the course, it is essential to know the horse's stride length and how far the horse is able to reduce or extend its stride length. Depending on its training status and physical conditions, a horse's stride length and variability is different and not always visually predictable. Riders need to prepare the distribution of canter strides in front of a fence equally so that the horse can jump a fence comfortably. This includes bringing the horse towards the fence in an approximately 90 degree angle, so that it can jump the fence straight and close to its centre. If fences are placed in short distance to each other, it is required to make a specific amount of strides between the fences. The rider needs to decide beforehand how many strides to make with the horse. Inside a jumping course, two fences can have no canter strides between them (so called in-and-out), 1-2 strides between them (so called combination), or 3-7 strides between them (so called distances). A well prepared distance reduces the amount of interference between the rider and the horse.

Throughout the course, the rider needs to keep harmony and equality in the frequency, speed and length of canter strides. If the horse increases or decreases the stride length right before the jump more than a certain degree, the probability of an unsuccessful jump increases. Figure 1 compares the strides a rider determined and the horse performed before a proper and a poor jump. In picture (i) and (j), we observe a tilted body position of the horse to the left, performing a jump unorthogonal to the fence jumping to the right side. The horse is scrambling over the jump which can be seen because its head is up and its neck is inverted (not round) to compensate for taking off too far away from the jump. This position results from the rider not determining the last canter strides before the jump correctly ((f)-(h)). The takeoff point before the jump is too far away from the fence and the horse is forced to jump in a different flight curve than physically comfortable. In contrast, picture (a)-(c) show a correct determination of canter strides, leading to good takeoff point a natural flight curve of the horse over the fence ((d)-(e)). We observe the horse to have a round and balanced neck over the jump.

RELATED WORK

Previous research studies in the field of Animal-Computer Interaction (ACI) have investigated the use of wearable technologies for different animal species. Research was done with dogs and the respective dog-human interaction using wearable sensors. Ladha et al. [6] focused on activity recognition for dogs, and classified jumping and walking for measuring the dogs' well being. Similarly, Valentin et al. [14] studied how to classify different dog gestures for the dog's communication with humans using wearable sensors. Thompson et al. [12] classified and quantified postures and posture transitions for enhancing welfare and productivity of pigs. Haladjian et al. and Pastell et al. [4, 9] developed an approach to detect anomalies in the gait of cows that might be caused by a lameness-related disease.

Previous research on horses includes using wearable sensors both for the kinematic analysis of horses as well as for tracking the rider's posture. Low et al. [8] studied how to prevent lamenesses in racehorses using a wireless sensor system attached to the horse's limbs, whereas Uhlir et al. [13] used a wearable motion sensor in the horse's neck to detect lamenesses in horses. Thompson et al. [11] developed a system for automated feedback for dressage riders. They used several sensors on the horse's limbs to classify different gaits and specific exercises in the domain of dressage. Green et al. [5] provided a system for tracking positions, velocity and physiological data of horses using wearable sensors in equestrian training. Barrey and Galloux studied the kinematics of a horse's gait and found that the penalty rate increased if the horse was ridden in a low stride frequency and suddenly reduced its stride frequency at takeoff [1, 3]. Other studies track the rider's behaviour and posture. Li et al. [7] investigated how to correct an equestrians posture using a wearable sensor system. A similar study quantifies the horse-rider-expertise using inertial sensors in show jumping to classify their performance level [10].

In comparison to other approaches, our system focuses on tracking and gathering information on the canter pattern of the horse relevant for the respective rider in the context of horse sports.

METHODS

Our system computes specific characteristics for the canter gait of a horse. We perform computations on the horse motion signal acquired by a smartphone. All further processing and evaluation methods are based on the collected motion signal.

Study Design

We collected motion data from 9 different horses in their regular showjumping or cross-country training. All horses completed one or two full jumping courses, including 4 cross-country courses and 5 showjumping courses. Our recordings include gait motion such as walking and trotting before and after the actual execution of the course. The horse-rider-pairs jumped different fences such as verticals (see Figure 1), oxers (two verticals in close distance to each other, to make the jump wider) or tree trunks in different heights, widths and shapes depending on the experience of the pair. These different fences require different takeoff points to accomodate the width of the respective jump. The horse-rider-pairs we selected ranged in their levels of expertise. Some horse-rider-pairs were able to jump 100 cm (± 5 cm) fences and others jumped 125 cm (± 5 cm) fences. We chose horses of different heights to represent a broad variety of stride lengths. All horses and their information is shown in Table 1. We recorded the signals produced by the accelerometer, gyroscope and magnetometer available in an iPhone 7 with a sampling rate of 100 Hz.

We placed the smartphone in a bag attached to the right side of the saddle pad, with its display facing the horse's body. Figure 2 shows how the iPhone was attached and illustrates the sensor coordinate system. We mounted the bag on the horse's torso to capture its vertical and horizontal body movement. If the rider had a stable and balanced posture inside the saddle, contact to the smartphone with the rider's leg was not possible. We constructed the bag to fit the iPhone tightly to avoid noise in the signal caused by shaking. We gathered the movement of all 9 horses in four different riding arenas. Our data set contains movements from two indoor arenas with a size of 20x30m and 20x60m as well as two outdoor arenas with a size of 50x60m and 200x400m. Outdoor arenas enable horses to gain more speed, which potentially leads to an increased stride length and frequency compared to smaller indoor arenas. We video recorded all horse-rider pairs during the data collection and annotated start and end points for all canter strides according to the video protocol.

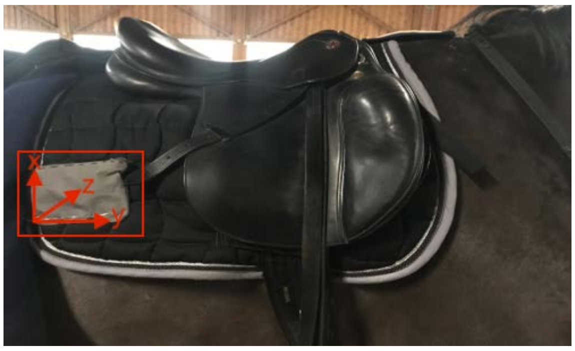


Figure 2. Placement of the smartphone on the horse's body for measuring body movements during jump training.

#	*Fence height*	*Horse Height*	*Discipline*
1	110	164	CC
2	110	145	CC
3	110	145	CC
4	110	164	CC
5	120	170	SJ
6	100	175	SJ
7	100	165	SJ
8	115	167	SJ
9	125	170	SJ

Table 1. Overview of the participants including parameters such as fence height of the course (± 5 cm). Different horse heights were chosen to represent different stride lengths according to physical conditions of the horse. Performed disciplines were Showjumping (SJ) and Cross-Country (CC).

Extraction of Canter Strides

Signal Preprocessing

To evaluate specific canter characteristics, we attenuated walk and trot segments in the recorded data using the approach described by Echterhoff et al. [2]. This approach detects canter strides and jumps with a precision of 94.6% and classifies them with an accuracy of up to 95.4%. We use this method to detect the respective gait and jump segments for further detailed processing.

To calculate the length of a canter stride, we determine the start and end of each stride. For this purpose, we filter the data with a first order Butterworth low-pass filter at 20 Hz to eliminate noise from the signal. The raw signal $S1_{raw}$ is transformed to the filtered signal $S1_{filt}$ within the signal length m:

$$S1_{filt}(x_i) = Butter(S1_{raw}(x_i)) \quad 1 \leq i \leq m \tag{1}$$

Peak Detection

The canter gait pattern is determined by two local minima representing the start and end point as well as one local maximum. To obtain the local maximum for each stride, we run a peak detection on $S1_{filt}$. To get the start and end of each canter stride, we run another peak detection on the inverse filtered acceleration x-axis signal $S1_{filtinv}$, which is defined as:

$$S1_{filtinv}(x_i) = S1_{filt}(x_i) * (-1) \quad 1 \leq i \leq m \tag{2}$$

The threshold h and minimal peak distance r for the peak detection were chosen by visual observation of the raw signal and set to 0.5 and 40. We detect peaks $P_j(x_i)$ for $j \in \{filtinv, filt\}$ for canter strides as follows:

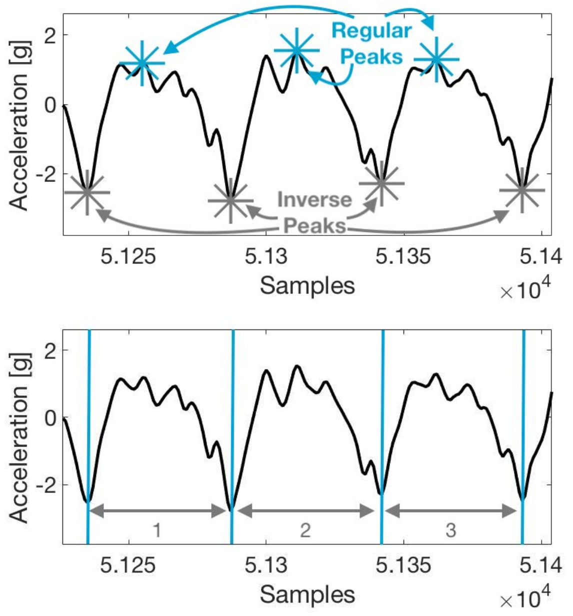


Figure 3. Stride segmentation performed on the filtered x-axis acceleration signal by detecting peaks (blue) and inverse peaks (grey) (top). Final canter stride segments 1,2,3 based on the peak detection (bottom).

$$P_j(x_i) = \begin{cases} 1, & \text{if } s \geq h.\forall s \in \{S1_{filtinv}(x_i), S1_{filt}(x_i)\} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Stride Segmentation

To map each detected canter stride $P_{filt}(x_i)$ to one detected start $P_{start}(x_i)$ and one detected end peak $P_{end}(x_i)$, the two peaks around $P_{filt}(x_i)$ were chosen:

$$\begin{aligned} P_{start}(x_i) = P_{filtinv}(y_i). \\ \forall P_{filtinv}(y_i) < P_{filt}(x_i) \wedge P_{filtinv}(y_i) > P_{filt}(x_{i-1}) \end{aligned} \quad (4)$$

$$\begin{aligned} P_{end}(x_i) = P_{filtinv}(y_i). \\ \forall P_{filtinv}(y_i) > P_{filt}(x_i) \wedge P_{filtinv}(y_i) < P_{filt}(x_{i+1}) \end{aligned} \quad (5)$$

If there were more than one $P_{filtinv}(x_i)$ detected between two $P_{filt}(x_i)$ and $P_{filt}(x_{i+1})$, we chose the one closest to the respective $P_{filt}(x_i)$.

The canter stride segments k, detected with our peak detection, are defined by:

$$k = [P_{start}(x_i), P_{end}(x_i)] \quad (6)$$

Figure 3 shows three canter strides of the filtered signal $S1_{filt}$ and explains how the peaks were detected (top) and segmented (bottom).

Stride Length Evaluation

The segments are characterised by the duration D in samples:

$$D(x_i) = P_{end}(x_i) - P_{start}(x_i) \quad (7)$$

The average stride duration d in samples and seconds of each horse is defined by

$$d_{samples} = \frac{\sum_{i=1}^{n} D(x_i)}{n} \quad (8)$$

$$d_{sec} = \frac{d_{samples}}{f} \quad (9)$$

for the overall amount of canter strides n in seconds with sampling frequency $f = 100$.

For each horse, the average canter stride length $d_{samples}$ is linearly compared with the horse's height h using a pearson correlation ρ to measure the strength of relationship between the two characteristics.

$$\rho = \frac{\text{cov}(h,d)}{\sigma_h \sigma_d} \quad (10)$$

with standard deviation σ_h, σ_d and covariance $\text{cov}(h,d)$ of the horse height and canter stride length. A pearson correlation coefficient of 1 indicates a strong positive relationship, -1 indicates a strong negative relationship. A coefficient of 0 describes the absence of a linear relation.

Finding the Number of Canter Strides before Fences

Based on our stride extraction method, we detect the last canter strides before a jump. To find the relevant strides, we first detect jumps inside the horse motion signal using [2]. We subsequently process the preceding 6 seconds before a jump occurred. If this preceding time interval contains another jump we trim the interval to fit between the respective fences. If the time interval does not contain another jump, the interval is trimmed to evaluate the last 7 canter strides before the jump. We further process the interval with our stride segmentation algorithm. The number and distribution of the detected canter strides between two fences is then displayed inside the user interface as an overview of the critical segments inside a jumping course.

USER INTERFACE

All gait characteristics are evaluated in real time inside the system, so that riders can see their training statistics for each jump. Inside the user interface, we show three different components. To evaluate the ridden path, an overview of the course is shown. The rider can see the path to evaluate the route to the respective fences (Figure 4 (a)). The user interface shows if the curves to the fences were ridden correctly and if the fences were reached in the right angle. Based on this course overview, the rider can zoom into specific sections of the course. The system shows the specific amount of strides made between fences for distances and combinations as well as the last canter strides before every single fence. The rider is thus able to evaluate the canter strides before a fence and see if the last strides were equally distributed. Figure 4 (b) shows this distribution of canter strides between fences, which helps the rider to post-evaluate problems of the jumping course. Riders can furthermore observe canter characteristics of their

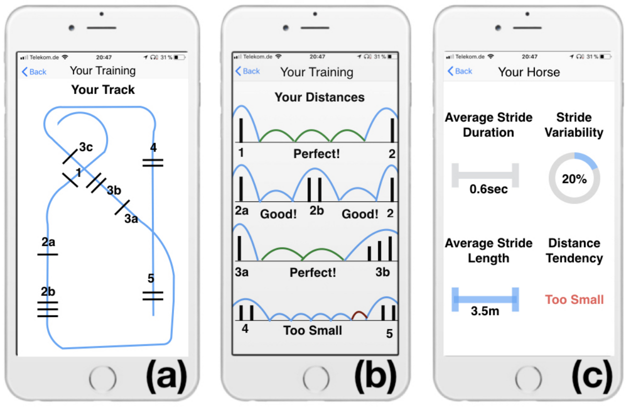


Figure 4. Bird's eye view of the course jumped (left) and obtained horse characteristics (right). Distribution of canter strides between fences. Green color indicates an equal distribution of canter strides before the fence, red color indicates that the canter strides before the jump were not equally distributed (middle).

horse inside the application. We enable the rider to consider the horse's training status by its gait characteristics. The displayed characteristics are a basis for for recommendations regarding possible training and competition strategies. For instance, if the rider gets information on the horse's small stride length from our system, she can consider making one canter stride more between the fences in the future. By knowing the amounts of strides to be performed before specific fences, riders can lead horses to find better takeoff points and gain trust in the rider. The system supports the rider by offering a post-evaluation of the ridden course. The rider can review the horse's performance and development over time and conclude further training steps. The user interface was designed and assembled by an experienced rider and enriched by the study participants suggestions.

RESULTS

Extraction of Canter Strides

Due to the rolling up and down movement of the horse in each canter stride we can detect the beginning and end of each stride. To evaluate the canter stride detection method, all beginnings and endings of canter strides in our data set were previously labeled according to a collected video protocol. In total, 3349 canter strides were labeled and used as our ground truth. A canter stride was detected correctly and marked as a True Positive (TP) if the maximum difference d of the detected to the labeled inverse peak $P_{filtinvstart}(x_i)$ and $P_{filtinvend}(x_i)$ is $d \leq 0.1$ s and the segment k contained a labeled peak $P_{filt}(x_i)$. The maximum difference d was chosen narrowly to ensure a precise calculation of the stride length. Detected strides or jumps were marked as False Positives (FP), if no label was found within range $d > 10$ samples. Labeled, but undetected strides were marked as False Negatives (FN). Correctly undetected instances are referred to as True Negatives (TN). An overview and the respective share to the ground truth is shown in Table 2.

#TP	*%*	*#FP*	*%*	*#FN*	*%*
3204	95.7	122	3.6	145	4.3

Table 2. True Positives, False Positives, False Negatives and their respective share of the overall data set.

#Instances	*Precision*	*Recall*	*F-Measure*
3349	96.3	95.7	96.0

Table 3. Performance of our canter stride detection approach in %.

Figure 5 shows a comparison of every ground truth canter stride length and the respective canter stride length detected by our system. This figure shows that every horse has a tendency towards a certain stride length. To explain the difference in canter stride lengths of different horses, Table 4 shows these two characteristics of the observed horses. We further compared the extracted stride length to the horse's height using the pearson correlation, since a smaller horse typically has a smaller stride length. The horse's stride length and the horse's height are positively correlated with a pearson correlation coefficient of 0.73.

CONCLUSION

This paper provides an evaluation of canter stride characteristics for showjumping and cross-country riding. Our stride detection algorithm extracts strides with a precision of 96.3% and a recall of 95.7%. Our peak detection method is thus

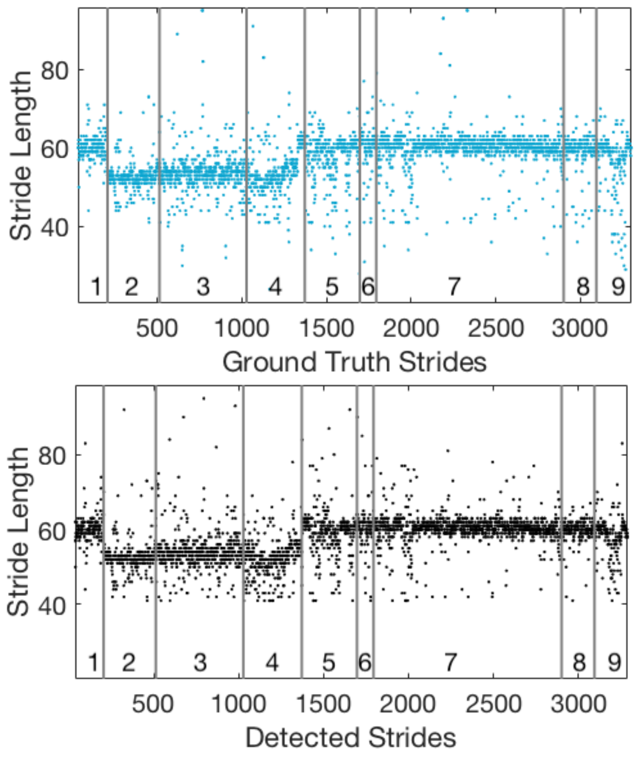


Figure 5. Comparison of the real and detected canter stride length. The vertical sections indicate the canter strides made by each different horse.

#	*Stride Length*	*Horse Height*
1	60.4	164
2	52.7	145
3	54.4	145
4	52.6	164
5	59.5	170
6	60.8	175
7	60.3	165
8	59.8	167
9	58.4	170

Table 4. Average stride length in samples per horse and horse height in cm.

suitable to provide feedback to the rider in real life. The misdetection rate of 3.6% and 4.3% for undetected instances of our system indicates that the system's prediction of the average stride length will not differ significantly when tracking the gait for an entire training session. We evaluate the ridden strides by their length, which can help the rider to get an overview of the training status and physical condition of a horse. This can be used as a reference to plan the amount of strides to make between fences for a successful course strategy. The pearson correlation between the horse's height and the stride length is positively correlated with a correlation coefficient of 0.73. The horse's height is thus an indicator of its average stride length.

In the future, we would like to investigate which other parameters have an impact on the horse's average stride length. Furthermore, we would like to study the importance of a correct takeoff point before a fence and its significance for a successful jump. Some horses tend to get too close to the fence before taking-off, and others takeoff to early before the fence. This can lead to the horse touching or even falling into the fence. Typically, these mistakes can be corrected by proper training. We would like to study how to evaluate the jumping tendency of the horse and provide this information as a training suggestion to the rider.

To enhance the rider's experience with the system, we would like to evaluate the user interface in a study to improve the impact of the user interface on the rider's decision making and consequently the horse-rider-pair's performance.

REFERENCES

1. E. Barrey and P. Galloux. 1997. Analysis of the equine jumping technique by accelerometry. *Equine Veterinary Journal* 29, 23 (1997), 45–49.
2. J. Echterhoff, J. Haladjian, and B. Brügge. 2018. Gait and jump classification in modern equestrian sports. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, To appear.
3. P. Galloux and E. Barrey. 1997. Components of the total kinetic moment in jumping horses. *Equine Veterinary Journal* 29, 23 (1997), 41–44.
4. J. Haladjian, B. Brügge, and S. Nüske. 2017. An approach for early lameness detection in dairy cattle. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*. ACM, 53–56.
5. A. Hill, A. Slamka, Y. Morton, M. Miller, and J. Campbell. 2007. A real-time position, velocity, and physiological monitoring and tracking device for equestian and race training. In *Proceedings of the ION GNSS*.
6. C. Ladha, N. Hammerla, E. Hughes, P. Olivier, and T. Plötz. 2013. Dog's life: Wearable activity recognition for dogs. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 415–418.
7. J. Li, Z. Wang, J. Wang, H. Zhao, S. Qiu, and M. Guo. 2017. Study on the attitude of equestrian sport based on body sensor network. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 3653–3658.
8. J. Low, V. Mak, B. Forbes, F. Sepulveda, and C. Yeow. 2016. Development of a wearable gait detection system for racehorses. In *ISBS-Conference Proceedings Archive*, Vol. 33.
9. M. Pastell, M. Kujala, A. Aisla, M. Hautala, V. Poikalainen, J. Praks, I. Veermäe, and J. Ahokas. 2008. Detecting cow's lameness using force sensors. *Computers and Electronics in Agriculture* 64, 1 (2008), 34–38.
10. M. Patterson, J. Doyle, E. Cahill, B. Caulfield, and U. Persson. 2010. Quantifying show jumping horse rider expertise using IMUs. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*. IEEE, 684–687.
11. R. Thompson, I. Kyriazakis, A. Holden, P. Olivier, and T. Plötz. 2015. Dancing with horses: Automated quality feedback for dressage riders. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 325–336.
12. R. Thompson, S. Matheson, T. Plötz, S. Edwards, and I. Kyriazakis. 2016. Porcine lie detectors: Automatic quantification of posture state and transitions in sows using inertial sensors. *Computers and Electronics in Agriculture* 127 (2016), 521–530.
13. C. Uhlir, T. Licka, P. Kübber, C. Peham, M. Scheidl, and D. Girtler. 1997. Compensatory movements of horses with a stance phase lameness. *Equine Veterinary Journal* 29, 23 (1997), 102–105.
14. G. Valentin, J. Alcaidinho, A. Howard, M. Jackson, and T. Starner. 2016. Creating collar-sensed motion gestures for dog-human communication in service applications. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 100–107.

6.5 Gait and Jump Classification in Modern Equestrian Sports

This publication presents a wearable device application that tracks the gait of horses (e.g. walking, trotting, running) and detects horse jumps during showjumping sports. The paper details how the data was collected using a smartphone, processed using a discrete wavelet transform and classified using different machine learning methods.

The author of this Habilitation supervised Jessica Echterhoff throughout the entire development process presented in the paper including the data collection, the implementation of the recognition algorithm in Matlab and writing the paper.

Authors	Echterhoff, J., Haladjian, J., & Bruegge, B.
Conference	International Symposium on Wearable Computers (ISWC)
Number of Pages	4
Type	Short Paper
Review	Peer Reviewed (4 Reviewers)
Year	2018
DOI	https://doi.org/10.1145/3267242.3267267

Gait and Jump Classification in Modern Equestrian Sports

Jessica Maria Echterhoff
Technical University Munich
Munich, Germany
contact@jessicaechterhoff.com

Juan Haladjian
Technical University Munich
Munich, Germany
haladjia@in.tum.de

Bernd Brügge
Technical University Munich
Munich, Germany
bruegge@in.tum.de

ABSTRACT
In modern showjumping and cross-country riding, the success of the horse-rider-pair is measured by the ability to finish a given course of obstacles without penalties within a given time. A horse performs a successful (penalty-free) jump, if no element of the fence falls during the jump. The success of each jump is determined by the correct take-off point of the horse in front of the fence and the amount of strides a horse does between fences. This paper proposes a solution for tracking gaits and jumps using a smartphone attached to the horse's saddle. We propose an event detection algorithm based on Discrete Wavelet Transform and a peak detection to detect jumps and canter strides between fences. We segment the signal to find gait and jump sections, evaluate statistical and heuristic features and classify the segments using different machine learning algorithms. We show that horse jumps and canter strides are detected with a precision of 94.6% and 89.8% recall. All gaits and jumps are further classified with an accuracy of up to 95.4% and a Kappa coefficient (KC) of up to 93%.

ACM Classification Keywords
I.5.4 Pattern Recognition: Applications: Signal Processing

Author Keywords
Activity Recognition; Wearable Sensing; Showjumping

INTRODUCTION
Horse riding was officially included in the 1900 Olympic games in Paris, and since then equestrian activities gained popularity in the private and professional sector. Showjumping as well cross-country riding are sports that require a rider and a horse to jump over a course of fences in a specific order within a given time. Depending on the course and type of competition, horses have to jump different kinds of fences: upright fences (e.g. verticals), high and wide fences (e.g. oxers, triple bars, tree trunks), wide and flat fences (e.g. moats). Inside the course, fences are placed separately or in close distance to each other. The horse has to perform a specific amount of strides between fences in order to execute a successful jump.

A typical sport horse is able to move in three different gaits - walk, trot and canter, which differ in the intensity of speed, frequency and vertical swing. Walk, trot and transitions between the two gaits are ridden for warming up the horse, keeping it warm and cooling it down before and after the jumping course. A balanced warm-up is essential for a responsive horse inside the jumping course and the foundation of every successful jump. A sport horse typically jumps all fences in a course running towards them in canter. It is possible to jump fences when trotting, but not common in competition scenarios. The amount of strides between two jumps have to be chosen by the rider and depend on the individual stride length and training experience of the horse. Riders usually walk around the course before a competition to measure distances between the fences and calculate how many strides to make with their horse. The ability to determine the optimal amount of strides to perform before each fence is critical for a successful jump. However, the stride length and canter pattern of every horse are different and not always visually predictable, and depend on the rider's experience. Failing to jump a wooden fence can cause injury to the horse and rider. Currently, human trainers provide feedback to riders for each jump in training sessions. However, not every rider can afford a human trainer. Video feedback is used for short jump sequences, but it might be impractical to replay an entire training or competition. We propose a method to classify horse gait types and detect canter strides and jumps performed by a horse-rider-pair using a smartphone attached to the horse's saddle. Our method aims at generating a training report that can be used by riders to analyse and keep track of their training performance by generating a list of jumps and strides made before each jump. This solution supports the

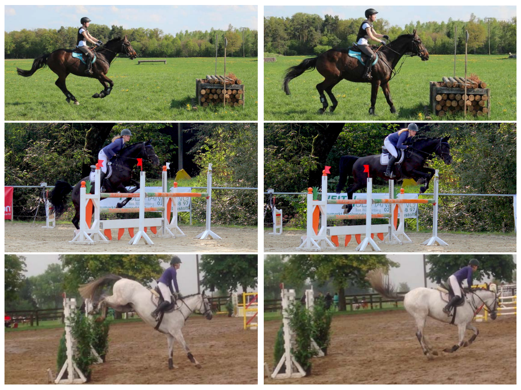


Figure 1. Three horses performing jump sequences over different fence types such as a a tree trunk, an oxer and a vertical (from top to bottom).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISWC '18, October 8–12, 2018, Singapore, Singapore
© 2018 ACM. ISBN 978-1-4503-5967-2/18/10...$15.00
DOI: https://doi.org/10.1145/3267242.3267267

rider in making strategic decisions on how to ride a course and can prevent accidents due to a miscalculation of canter strides in between fences.

Related Work

Wearable sensors have been used in a variety of sport applications. In the field of wearable technologies for animals, research was done with dogs and the respective dog-human interaction using wearable sensors [5]. Ladha et al. [6] focused on activity recognition for dogs, and classified jumping and walking for measuring the dogs' well being. Haladjian et al. [2] developed an approach to detect anomalies in the gait of cows that might be caused by a lameness-related disease. Previous work on equestrian applications includes using wearable sensors both for the kinematic analysis of horses as well as for tracking the rider's posture. Low et al. [8] studied how to prevent lamenesses in racehorses using a wireless sensor system attached to the horse's limbs. Thompson et al. [10] developed a system for automated feedback for dressage riders using sensors on the horse's limbs to classify different gaits and specific exercises in the domain of dressage. Green et al. [4] provided a system for tracking positions, velocity and physiological data of horses using wearable sensors in equestrian training. Barrey and Galloux studied the kinematics of a horse's gait and found that the penalty rate increased if the horse was ridden in a low stride frequency and suddenly reduced its stride frequency at take off [1]. Other studies track the rider's behaviour and posture. Li et al. [7] investigated on how to correct an equestrians posture using a wearable sensor system. A similar study quantifies the horse-rider-expertise using inertial sensors in show jumping to classify their performance level [9].

METHODS

Our data set was collected using the accelerometer and gyroscope of an Apple iPhone 7 with a sampling rate of 100 Hz. We placed the smartphone in a tightly fitted bag attached to the right side of the saddle pad, its display facing the horse's body. We mounted the bag on the horse's torso to capture its vertical and horizontal body movement. We gathered the movement of 9 horses in 4 different riding arenas, containing movements from 2 indoor and 2 outdoor arenas. Outdoor arenas enable horses to gain more speed and lead to an increased stride length. We monitored all horse-rider-pairs during their usual jumping or cross-country training. Our recordings include gait motion such as walking and trotting before and after the actual execution of the course. The horse-rider-pairs jumped different fences in different heights, widths and shapes depending on the experience of the pair (Figure 1). All pairs jumped in their levels of expertise, some being able to jump 100 cm (± 5 cm) fences and others jumped 125 cm (± 5 cm) fences during our study. We chose horses of different heights to include different stride lengths to our data set (Table 2). We video recorded the horse-rider-pairs and annotated the beginning and ending of each walk and trot section as well as canter stride and jump peaks according to the video protocol. In total, 49.24 minutes of data were recorded, including 25.38 minutes of canter, 12.48 minutes of trot, 9.61 minutes of walk and 1.77 minutes of jumps.

FH[cm]	*RA[y]*	*HH[cm]*	*FH[cm]*	*RA[y]*	*HH[cm]*
110	> 35	145-165	100	> 35	145-165
110	26-35	145-165	100	> 35	> 165
110	< 15	< 145	115	15-25	> 165
110	15-25	< 145	120	15-25	> 165
			125	> 35	> 165

Table 1. Overview of the participants including horse height (HH), fence height (FH) of the course (± 5 cm) and rider age (RA) in years indicating experience of cross-country (left) and showjumping (right) riders.

Stride and Jump Detection

To find and further classify gait and jump segments, we use two methods: we first perform an event detection to detect canter strides and jumps. If no event is detected, we determine walk and trot segments, where we do not detect single steps. Detecting canter strides is particularly challenging because most trot strides have a comparable acceleration. We decided to detect canter strides and jumps on the acceleration along the x-axis, based on our observation that most walk strides and some trot strides have less acceleration than canter strides along this axis (Figure 2 (a)).

We designed a filter based on Maximal Overlap Discrete Wavelet Transform (MODWT) to reduce noise and attenuate the frequencies dominant in most trot strides. We found the seventh level MODWT using the 'Sym2' symlet mother wavelet to produce the desired effect based on a visual comparison of the raw and filtered signal for the different gait types. The raw signal $S1_{raw}$ is transformed to wavelet components $M_1(x_i)$ within the signal length m $(1 \leq i \leq m)$.

$$M_1(x_i) = MODWT(S1_{raw}(x_i)) \tag{1}$$

By retaining levels six and seven for the signal reconstruction $S1_{MODWT}(x_i)$, we discard narrower frequencies of the trot sections, while keeping the wider frequencies from canter strides/jumps (Figure 2 (b)).

$$S1_{MODWT}(x_i) = IMODWT_{6:7}(M_1(x_i)) \tag{2}$$

We run a peak detector on $S1_{MODWT}(x_i)$ using a minimal peak height h and a minimal peak distance r (Figure 2). The threshold h and minimal peak distance r were evaluated experimentally and set to 1.25 and 30. We detect peaks for canter strides and jumps as follows:

$$P_j(x_i) = \begin{cases} 1, & \text{if } S_1(x_i) \geq h \wedge P_{j-1}(x_{i-r}) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

All peaks detected by our peak detector are further used for signal segmentation and feature extraction.

Gait, Stride and Jump Segmentation

We segment the signal using two different segment sizes depending on the output of the peak detection: we define peak-centered segments k_p, aiming at describing canter strides and jumps, to ensure that one exact canter stride/jump is evaluated and further classified at a time. We process gaps between peaks using duration-dependent segments k_d, intending to describe a walk or trot interval rather than the single strides made in this interval (Figure 3). We determine segments k_p for every

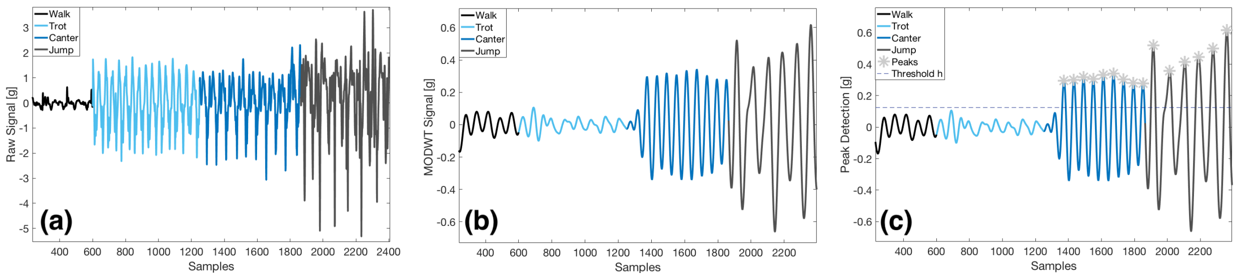


Figure 2. Raw data stream of the x-axis accelerometer data for different movements walk, trot, canter and jumps (a). Processed data stream using MODWT (b). Peak detection based on threshold h on the processed data (c).

detected peak $P_j(x_i) = 1$ using a fixed frame located around the peaks. We set the segment size for detected peaks to 100 ms based on the observation that most jumps last around 100 ms. We use the duration-dependent method if there is no peak detected within the last 150 ms. Based on our observation that jumps and canter strides have a wider length than walk or trot, we define a smaller window size of 50 ms to process duration-dependent segments. We define our peak-based segments k_p and duration-based segments $k_d = [x_{i-a}, x_{i+b}]$ with:

$$a,b = \begin{cases} 50, & \text{if } P_j(x_i) = 1 \\ 25, & \text{otherwise} \end{cases} \tag{4}$$

Feature Extraction and Selection

We calculate 268 different features for each determined segment for all axes of the accelerometer and gyroscope as well as for their general magnitude and normalise them between $[0;1]$. We calculate standard statistical as well as heuristic features for our specific problem. To find a feature set that describes our data set as precisely as possible, we studied features in the time and frequency domain, similar to [3]. We use the Minimum Redundancy Maximum Relevance algorithm (mRMR) to select 20 more relevant features for our entire data set.

Classification

We classified all feature vectors based on all peak-centered and duration-dependent segments using 22 different machine learning algorithms. Support Vector Machines (SVM) with linear, quadratic and cubic and a fine, medium and coarse gaussian kernel were tested. The gaussian SVM kernel with a number of predictors P were set to $\sqrt{P}/4$ for the fine, $\sqrt{P}$ for the medium and $\sqrt{P} * 4$ for the coarse gaussian SVM. We tested the K-Nearest Neighbours (KNN) algorithm with a fine (1 neighbour), coarse (100 neighbours), medium, cosine, cubic and weighted (10 neighbours) kernel. Trees were computed with a maximum number of splits of 100 for fine, 20 for medium and 4 for coarse trees. We processed ensembles using boosted, bagged, RUSBoosted trees, subspace discriminants, subspace KNN, linear and quadratic discriminant algorithms. We computed a 9-fold cross validation to test and train all machine learning algorithms. All classifiers are compared by accuracy and KC to comprise the imbalance of the jump class in comparison to the gait classes to find a suitable classifier for our problem.

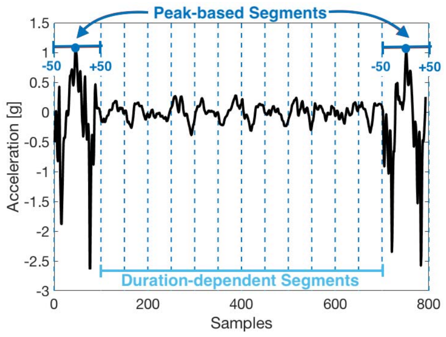


Figure 3. Segmentation explained by the acceleration of a horse performing a transition from canter to walk to canter.

RESULTS

A canter stride or jump was detected correctly and marked as a True Positive (TP) if the maximum deviation d of the detected to the labeled peak is $d \leq 50$ ms. Detected strides or jumps were marked as False Positives (FP), if no label was found within range d. Labeled, but undetected strides or jumps were marked as False Negatives (FN). Correctly undetected instances were referred to as True Negatives (TN).

Stride and Jump Detection Results

We evaluated the threshold h and minimal peak distance r to optimise the f-measure for our peak detector based on our entire data set. This lead to an optimised threshold $h = 0.125$ and minimum peak distance $r = 30$ for our data set. We labeled 2687 canter strides and 137 jumps the ground truth. Our final peak detector based on threshold h and minimal peak distance r detected 2535 correct instances in total, including 107 jumps and 2428 canter strides. The peak detector detected 144 false positives, whereof 102 were walk and 40 trot, as well as one misdetected canter stride and one jump (Table 3). It showed 289 false negatives (259 missed canter strides and 30 jumps) (Table 2).

Type	*#TP*	*%*	*#FP*	*%*	*#FN*	*%*
Canter	2428	90.4	141	5.3	259	9.6
Jump	107	78.1	3	2.2	3	21.9
Total	2535	89.8	144	5.1	289	10.2

Table 2. Performance of our detection method.

Class	*# FP*	*%*	*#FN*	*%*	*Prec.*	*Rec.*	*F-M.*
Walk	102	70.8	-	-	-	-	-
Trot	40	27.8	-	-	-	-	-
Canter	1	0.7	259	89.6	94.5	90.4	92.4
Jump	1	0.7	30	10.4	97.3	78.1	86.6

Table 3. Misdetection rate of our detection method per motion type. E.g. 70.8% of all incorrectly detected instances were walk (left). Precision, Recall and F-Measure of the detection in % (right).

	Walk	*Trot*	*Canter*	*Jump*	*Prec. [%]*	*Rec. [%]*
Walk	**1068**	13	30	0	91	96
Trot	72	**1273**	56	1	95	91
Canter	31	55	**2429**	4	96	96
Jump	0	2	9	**107**	91	96

Table 4. Confusion matrix of the linear SVM classifier. Rows indicate the ground truth, columns indicate predicted labels (left). Precision and Recall of the classifier in % (right).

Classification Results
All gaits and jumps could be classified with an accuracy between 90.2% (coarse tree) and 95.4% (cubic SVM). The KC ranged between 84.2% and 93.0%. For a detailed analysis, we proceeded using the linear SVM with an accuracy of 94.7% and a KC of 91.7%. Due to its lower computational costs, we accept the loss in accuracy of 0.7% and KC of 1.3%. The linear SVM algorithm classified canter strides and jumps with a precision of 96% and 91% and a recall of 96%. Jumps were misclassified as canter strides in most cases. If canter strides were misclassified, most of them were misclassified as trot steps. For walk and trot, the precision is 91% and 95%. The confusion matrix is shown in Table 4. Canter strides were mistaken by jumps in 2 cases (0.1% of all annotated canter strides), whereas jumps were mistaken by canter strides in 9 cases (7.8% of all annotated jumps). Walk was mistaken as canter strides in most cases, whereas trot was misclassified as walk in the majority of all misclassified trot instances.

CONCLUSION
This paper provides a solution for gait and jump classification in modern showjumping and cross-country riding. Data streams of a smartphone were evaluated to detect and classify gaits and jumps in the training of modern sport horses. We used the MODWT to remove gait-specific frequencies and a peak detector to extract features. Our canter stride and jump (peak-based) approach detects single strides and jumps with a precision of 94.6% and 89.8%. We showed that our method classifies gaits using machine learning algorithms with an accuracy of up to 95.4% and a Kappa coefficient of up to 91.7%.

Our study was conducted for the specific purpose of detecting jumps and strides in a given training session. The achieved accuracy of the detection and classification is applicable and it's degree of error acceptable to support the rider in decisions regarding the amount of canter strides to make between fences. Our approach can thus increase the probability of performing a successful jump and potentially prevent accidents due to the rider's miscalculation of canter strides between fences. Our work can be used to generate an evaluation of warm-up and cool-down phases which can be used by riders to analyse and keep track of their training performance. In the future, the computational costs and energy consumption should be assessed and the peak detection and feature selection parameters should be recalculated on a larger data set using an inner cross-validation.

REFERENCES

1. E. Barrey and P. Galloux. 1997. Analysis of the equine jumping technique by accelerometry. *Equine Veterinary Journal* 29, 23 (1997), 45–49.
2. J. Haladjian, B. Brügge, and S. Nüske. 2017. An approach for early lameness detection in dairy cattle. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*. ACM, 53–56.
3. J. Haladjian, J. Haug, S. Nüske, and B. Bruegge. 2018. A wearable sensor system for lameness detection in dairy cattle. *Multimodal Technologies and Interaction* 2, 2 (2018), 27.
4. A. Hill, A. Slamka, Y. Morton, M. Miller, and J. Campbell. 2007. A real-time position, velocity, and physiological monitoring and tracking device for equestian and race training. In *Proceedings of the ION GNSS*.
5. M. Jackson, G. Zeagler, C.and Valentin, A. Martin, V. Martin, A. Delawalla, W. Blount, S. Eiring, R. Hollis, Y. Kshirsagar, and T. Starner. 2013. FIDO-facilitating interactions for dogs with occupations: wearable dog-activated interfaces. In *Proceedings of the 2013 International Symposium on Wearable Computers*. ACM, 81–88.
6. C. Ladha, N. Hammerla, E. Hughes, P. Olivier, and T. Plötz. 2013. Dog's life: Wearable activity recognition for dogs. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 415–418.
7. J. Li, Z. Wang, J. Wang, H. Zhao, S. Qiu, and M. Guo. 2017. Study on the attitude of equestrian sport based on body sensor network. In *2017 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 3653–3658.
8. J. Low, V. Mak, B. Forbes, F. Sepulveda, and C. Yeow. 2016. Development of a wearable gait detection system for racehorses. In *ISBS-Conference Proceedings Archive*, Vol. 33.
9. M. Patterson, J. Doyle, E. Cahill, B. Caulfield, and U. Persson. 2010. Quantifying show jumping horse rider expertise using IMUs. In *Engineering in Medicine and Biology Society, 2010 Annual International Conference of the IEEE*. IEEE, 684–687.
10. R. Thompson, I. Kyriazakis, A. Holden, P. Olivier, and T. Plötz. 2015. Dancing with horses: Automated quality feedback for dressage riders. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 325–336.

6.6 A Wearable Sensor System for Lameness Detection in Dairy Cattle

This publication presents a system to detect deviations from the usual gait in dairy cattle and notify veterinarians. This work builds on top of the paper we present in Section 6.9. It discusses the computations to detect strides and classify them as *normal* or *abnormal* in greater detail. It also discusses and compares different alternatives to segment gait strides.

The author of this Habilitation developed and studied the different signal processing and activity recognition methods described in the article, designed the system and wrote the article.

Authors	Haladjian, J., Haug, J., Nueske, S., & Bruegge, B.
Journal	Multimodal Technologies and Interaction (MDPI)
Number of Pages	15
Type	Journal Article
Review	Peer Reviewed (3 Reviewers)
Year	2018
DOI	https://doi.org/10.3390/mti2020027

Multimodal Technologies and Interaction

Article

A Wearable Sensor System for Lameness Detection in Dairy Cattle [†]

Juan Haladjian [1,*], Johannes Haug [1], Stefan Nüske [2] and Bernd Bruegge [1]

[1] Lehrstuhl für Angewandte Softwaretechnik, Faculty of Informatics, Technical University Munich, Bolzmannstr 3, 85748 München, Germany; johannes.haug@tum.de (J.H.); bruegge@in.tum.de (B.B.)
[2] Lehr- und Versuchsgut Oberschleißheim, Faculty of Veterinary Medicine, Ludwig Maximilian University, St. Hubertusstraße 12, 85764 München, Germany; stefan.nueske@lmu.de
[*] Correspondence: haladjia@in.tum.de; Tel.: +49-89-289-18235

[†] This paper is an extended version of our paper published in the Fourth International Conference on Animal-Computer Interaction, Milton Keynes, United Kingdom, 21–23 November 2017.

Received: 17 April 2018; Accepted: 8 May 2018; Published: 15 May 2018

Abstract: Cow lameness is a common manifestation in dairy cattle that causes severe health and life quality issues to cows, including pain and a reduction in their life expectancy. In our previous work, we introduced an algorithmic approach to automatically detect anomalies in the walking pattern of cows using a wearable motion sensor. In this article, we provide further insights into a system for automatic lameness detection, including the decisions we made when designing the system, the requirements that drove these decisions and provide further insight into the algorithmic approach. Results from a controlled experiment we conducted indicate that our approach can detect deviations in cows' gait with an accuracy of 91.1%. The information provided by our system can be useful to spot lameness-related diseases automatically and alarm veterinarians.

Keywords: gait analysis; anomaly detection; unsupervised machine learning

1. Introduction

Lameness is a manifestation of painful disorders that result in an impaired movement or deviation from normal gait or posture [1]. In dairy cattle, the main causes of lameness are lesions in the claws which cause bacterial infections and swelling in cows' hooves and legs. Lameness causes severe pain and is associated with health issues such as the loss of fertility. Furthermore, lameness causes serious welfare and economic problems in the dairy industry. Some of the costs associated with lameness are the need for veterinary treatment and a reduction in milk production and cow's reproductive performance. At advanced stages of the disease, a lame cow might die or have to be sacrificed. Lameness is a common issue in dairy cows, with some stables having up to 72% lame cows [1].

The earlier a lame cow is identified, the earlier the causes of the disorder can be treated. Currently, lame cows are identified by visual inspection of their walking pattern, which is done by herdsmen. However, automation and the rapid growth in livestock production have led to more cattle and less employees per herd. As a consequence, herdsmen have less time to monitor the health condition of their cows. Automated systems for cow milking, feeding and cleaning are already being used in commercial herds. Neckbands with integrated motion sensors are used to predict whether a cow is undergoing estrus (i.e., the period of sexual fertility in a female mammal). Veterinarians monitor cow physical activity measured by these neckbands to determine the proper time to inseminate a cow. In contrast, systems to detect lameness are rarely used in commercial herds, despite the variety of solutions proposed by the scientific community.

Approaches for automated lameness detection have been studied using computer vision, external sensors such as pressure plates and wearable motion sensors. Most of the previous work on lameness detection using wearable motion sensors studied how to keep track of a cow's physical activity (lying down, standing and walking) [2,3]. However, changes in physical activity due to lameness occur at more advanced stages of the disorder. The first observable symptom of lameness is a change in a cow's usual walking pattern (i.e., gait).

This paper is an extended version of our previous work [4]. In our previous publication, we presented a sensor device and a set of algorithmic steps to detect deviations in cows' usual gait using a wearable motion sensor. The sensor device attached to a cow's hind limb is shown in Figure 1a. Assuming a cow is able to walk normally at the time the sensor is attached to it, our approach creates a model of the usual walking pattern of the cow using a combination of signal processing and machine learning methods. A trained machine learning model is used to detect deviations from the usual gait pattern of a cow later on.

In this article, we provide a more detailed insight of the algorithmic steps we introduced in [4,5]. Furthermore, we make our data available together with the publication, to facilitate the development and validation of other methods that might lead to an improvement in the life quality of cows. Furthermore, we list the requirements we elicited for a wearable sensor device for cows and describe the design decisions we made during this project. The design decisions we made can be used as a basis for designing future multimodal wearable sensing technologies for animals, as other wearable devices for animals will share similar requirements (e.g., low energy consumption, device robustness and waterproofness).

The rest of the paper is structured as follows:

Related Work: Provides an overview of other automated lameness detection systems and highlights how our approach differs from them.

Study Design: Discusses how we designed a study to collect data from 10 cows in order to develop and test our approach.

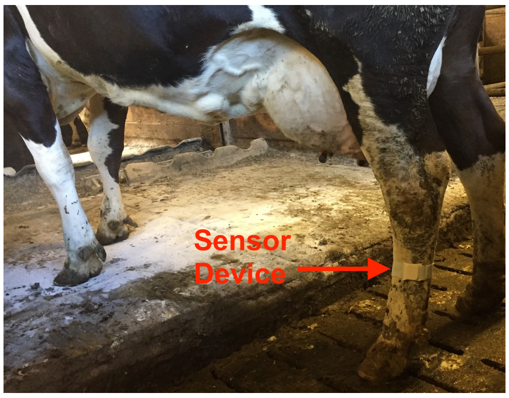
Requirements: Lists requirements we elicited for a wearable cow gait tracking system.

System Design: Lists the design decisions we made for a system able to detect lameness in dairy cattle and explains the rationale behind them.

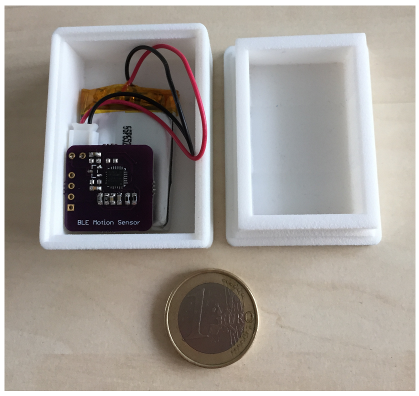
Approach: Describes our approach in detail including the hardware we designed and how we process the signals acquired by the sensor device in order to classify cow strides into *normal* or *abnormal*.

Evaluation: Presents the results of a controlled experiment we conducted in order to validate our approach.

Ethical Considerations: Discusses how we addressed the ethical guidelines in [6] during our study.



(**a**)



(**b**)

Figure 1. Motion sensor attached to a cow's hind left leg taken from [4,5] (**a**) and sensor box (**b**).

2. Related Work

Most modern stables collect data from cows' daily activity such as the amount of milk cows yield and how much food they are fed. Different studies have suggested using this data to predict lameness [7,8]. However, changes in milk yield and feeding behavior due to lameness might manifest days after changes in gait. Detecting a lame cow based on its gait would make it possible to stop further development of the disorder. This would allow veterinarians to treat the cause of the disorder earlier, relieving the cow from pain and restoring its normal function.

Approaches for lameness detection based on gait analysis include those that use computer vision, weight/force sensing or motion sensing. Computer vision approaches extract lameness related information from a video recording, such as the arching of a cow's back [9], the amount of overlapping between a cow's consecutive strides [10] and the angle at which a cow's fetlock joint makes contact with the ground during a stride [11]. Most of these studies have used normal cameras [9–11]. However, Van Hertem et al. [12] studied lameness detection using a 3D camera and Eddy et al. [13] used thermographic cameras.

Weight sensing approaches measure the weight a cow places on each limb while standing on force plates [14] or walking over a force-sensitive mattress [15,16]. Based on this data, information about cows' walking and standing behavior is calculated, such as the length and duration of a stride [15,17], the amount of kicks a cow performs while standing [14], the weight distribution under single hooves [16–18] and the frequency of steps [16].

Computer vision and weight sensing approaches are limited to measuring a few strides per cow and face additional challenges such as the fact that cows near the measuring area might disrupt the measurements [1] and the need for additional technologies to identify the cow being measured.

Motion-based lameness detection approaches rely on motion sensors that are attached to cows' legs and/or neck. Most motion-based lameness detection approaches measure parameters related to cows' daily physical activity, such as the amount of time cows spend lying, standing and walking [19,20], the number of strides cows perform per day [3] and the time of the days when cows start and stop walking [21]. These approaches do not analyze gait per se, but predict lameness based on cows' daily activity.

A few studies—mostly coming from the veterinary medicine community—have investigated cow lameness detection based on motion data. Pastell et al. [22] let lame and non-lame cows walk with accelerometers attached to all four limbs and developed a method based on wavelet analysis to predict the lameness. The study concluded that there is less symmetry in the acceleration of hind legs in lame cows than in healthy cows. Chapinal et al. [23] combined an accelerometer device with a weighing platform to measure the weight distribution, speed and number of steps performed by cows. The system also determines whether cows are lying or standing. In a second study, Chapinal et al. [24] found that the variance of acceleration of front and hind legs could be used to predict gait scores.

These studies compared lame cows with non-lame cows in order to discover differences in their gait and hence, did not consider the differences in the physical behavior and tolerance to pain of each individual cow. Alsaaod et al. [20] found that the variation of physical activity among cows is significantly larger than the variation of physical activity caused by lameness. This suggests that lameness should be regarded on an individual basis rather than comparing a cow's motion to a baseline established from other cows.

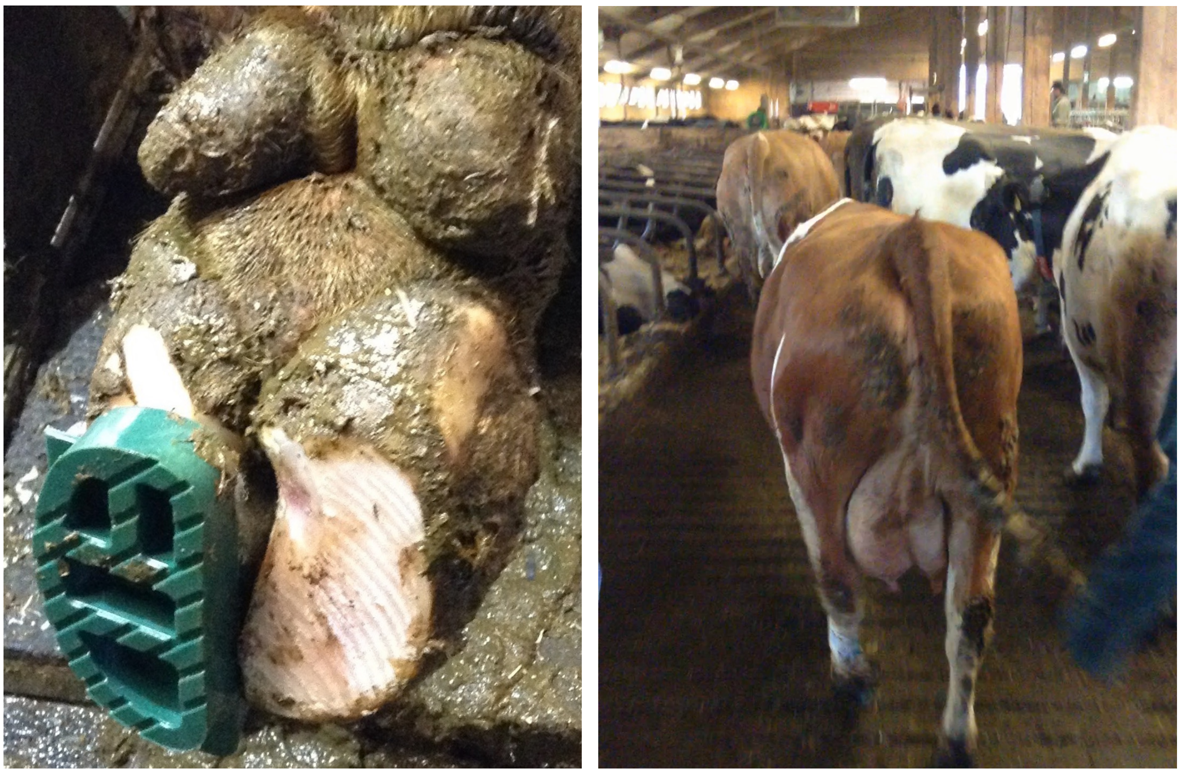
In contrast to previous approaches for lameness detection, our approach compares the gait of a cow to a baseline established by the cow itself during the first hours of use. Our approach is based in anomaly detection, a technique commonly used to detect bank fraud and intrusion in computer networks. The challenge at detecting such events, is that the anomaly data is usually not available at development time in order to train a computing device how to detect the abnormal events. Instead, what these methods do is to learn what the "usual" events are and try to detect anything that is not similar to them. Our approach learns the gait pattern of a particular cow and is able to detect deviations from this gait pattern afterwards. A deviation from the normal gait is the first indicator of a possible lameness. Our approach has two main advantages when compared to previous work on motion-based lameness detection approaches: (1) it takes into consideration the uniqueness of each cow's gait [20] and (2) it requires a single motion sensor attached to a hind limb.

3. Study Design

As described in our previous publication [25], we collected data from 10 cows while walking with our motion sensor attached to their hind left limb. Cows were chosen to maximize the diversity of age, weight and breed. Table 1 displays demographic information about each cow. We conducted five "runs" per cow. In each run, we let cows walk for approximately 7 min. In three of the runs, cows walked normally and in the other two runs, cows walked with a plastic block attached to the outer claw of either their left or right hind hoof. Runs were executed in different days. We performed only one normal run for cow 4 because it was isolated into a different stable due to pregnancy during the period this study lasted. Statistical information about the data we collected is shown in Table 1.

Figure 2a shows a plastic block attached to the outer claw of the left hind hoof of a cow. Among the other approaches we considered to collect data to validate our algorithm, we found this approach to be the most appropriate for an animal-centered research, because it does not involve pain (e.g., forcing a lame cow to walk) and required a considerable shorter intervention to cows' natural activity. The total intervention lasted approximately 40 min in total as was spread among different days.

We designed the experiment to resemble the conditions in which our approach would be used. We let cows walk in their usual environment rather than isolated walkways specially designed for the experiment. Furthermore, we included motion data of periods when cows stopped walking, turned and got bumped by other cows. Cows walked on two different types of ground: rubber and concrete.



(**a**) (**b**)

Figure 2. Plastic block attached to the outer claw of a cow's left hind hoof taken from our previous work [4] (**a**) and member of our team walking behind a cow in the indoor stable of the Ludwig Maximilian University (LMU) Munich during the data collection (**b**).

Table 1. Demographic data of the cows that took part in the controlled experiment and statistical information about the data we collected. GH = German Holstein, FV = Fleckvieh.

	Demographics			**Data Collected**		
#	**Age (Years)**	**Weight (Kg)**	**Race (DH/FV)**	**Normal (Minutes/Strides)**	**Right (Minutes/Strides)**	**Left (Minutes/Strides)**
1	5	910	31.25/68.75	42.3/1136	6.1/171	6.2/212
2	5	680	68.75/31.25	28.8/985	7.1/181	8.2/231
3	5	720	43.75/56.25	37.7/1166	6.5/202	5.4/162
4	7	560	87.5/12.5	5.5/242	7.6/290	7.8/256
5	6	700	31.25/68.75	48.8/1276	11.9/244	12.1/355
6	4	780	62.5/37.5	20.1/643	5.8/191	6.2/208
7	3	680	0/100	26.3/888	6.1/250	6.4/261
8	5	640	100/0	44.5/1378	8.2/300	8.1/310
9	4	610	0/100	38.8/1410	6.1/203	7.6/241
10	3	700	0/100	24.7/824	7.8/227	7.2/208

4. Requirements

We elicited the requirements for a wearable sensor device in a series of interviews with a veterinary research team from the Ludwig Maximilian University (LMU) and pilot studies at an indoor stable in Munich, Germany. Paci et al. [26] suggested that a wearable device that is not directly relevant to the animal's intentions should ideally not get in its way (i.e., affect its daily activities or experiences). To this end, our goal was to design a system that keeps track of the gait of cows with little influence in their daily life.

Low energy consumption. An intervention to a cow's natural activities is required every time a battery has to be replaced. Furthermore, farmers might not have time to collect every sensor device in an entire herd to replace a battery. Therefore, wearable devices to be used by herds of animals should consume little power and remain functional without intervention from veterinarians for as long as possible, ideally during the lifetime of the animal.

Attachment at a hind limb. The wearable device should be attached at a hind limb for three main reasons. First, cows usually lie down with their front legs bent and spread out their hind legs outwards. Second, when cows undergo oestrus, they jump with their front legs on other cows. Third, lameness is usually associated with diseases (mostly infections) occurring on hind legs.

Water, dirt and weight resistant. Cows in indoor stables are in contact with excrement and urine. Furthermore, cows might lick the device. In addition, cows might weight up to 1000 kg and might step on another cow's device with a sharp hoof. Therefore, the device should be water and dirt resistant and be able to cope with high amounts of forces applied at its surface.

Deployment that maximizes use by cows. There might be little space in cow indoor stables for deploying large devices and providing the device with power might require additional infrastructure. In particular, power and cables are subject to the same robustness requirements mentioned in the previous point. As a consequence, the deployment of a receiver device should take into account the requirements for the device to function (e.g., access to power and connection to transmit data) as well as the practices of the animal species (e.g., placement in the stable where cows walk regularly).

5. System Design

In this section, we list the main design decisions we made for an automated lameness detection system. These decisions derive from the requirements elicited in the previous section.

Local computations. Streaming sensor data from a wearable device is (considerably) more energy costly than performing computations locally. In order to reduce the energy consumption of the device, we decided to perform computations and store computed results on the sensor device and transmit them once a day while the cow is milked.

Custom sensor device. We decided to design our own sensor device with a flat and lightweight form-factor in a robust plastic 3D printed material. The goal of our design was to minimize the risk that a cow injures itself, other cows in the stable (e.g., by bumping the device onto other cows) or the device itself.

Individualized tracking. Alsaaod et al. [20] showed that the variation of physical activity among cows is significantly larger than the variation of physical activity caused by lameness. This suggests that comparing a cow's motion to a baseline established from other cows might not be insightful at detecting lameness. The veterinarians we collaborated with stated that each cow has an individual walking pattern and reacts differently to pain. For this reason, we decided to create an individual gait profile for each cow and detect changes in their locomotion.

Receiver at milking robot. In order to minimize energy consumption, the data should be sent from the wearable device to a nearby receiver. The milking robot represents a good place to collect data recorded by a wearable device for two main reasons. First, cows usually visit the milking robot twice a day. Second, cows remain still in the same area for several minutes during milking, which represents an ideal opportunity for the data transmission.

6. Approach

In this section, we describe our approach for anomaly detection of a cow's gait, including the sensor device that senses motion data and computations it performs. Our approach consists of two phases: the *training phase* and the *detection phase*. The *training phase* builds a model of the usual gait of a cow using a machine learning algorithm. The procedure requires a cow to be healthy during the first hours of use. The *detection phase* classifies the gait of a cow into *normal* or *abnormal* based on a comparison of its current gait with the model created during the *training phase*. Both, the *training phase* and *detection phase* require the following computations:

1. **Data acquisition**. The sensor signals are read from the sensor device and stored in memory.
2. **Preprocessing**. The data is organized in chunks and filtered to eliminate noise.
3. **Stride segmentation**. Cow strides are detected and their boundaries identified.
4. **Feature extraction**. Information describing of a cow's gait is extracted from each stride.

These computations produce a set of features (i.e., information describing the gait of a cow). This set of features is used during the *training phase* to train a machine learning model—we call this *Model Training*. During the *detection phase*, the already trained machine learning model assigns a label *normal* or *abnormal* to each set of features - we call this *Classification*. The following subsections describe each of these computations in more detail. Figure 3 shows an overview of the different computations performed by our approach.

6.1. Sensor Device

The sensor device consists of an electronic device, a battery to power it and a 3D printed enclosure. The electronic device we designed is based on an ARM Cortex-M0 microcontroller, a 6-axis Inertial Measurement Unit (IMU) and a Bluetooth Low Energy (BLE) module. The ARM Cortex-M0 microcontroller operates at 16 MHz and is characterized by its low-power consumption rate and small footprint. The device has 128 kb of flash memory with 8 kb of ram. As a communication module, we decided to use the BLE technology due to it's low-power consumption rate. We designed the electronic device as a two-layer printed circuit board (PCB) placing the motion sensor on the front side and the microcontroller and BLE module on the back side. The dimensions of the electronic device are 21 mm $\times$ 21 mm $\times$ 2.5 mm. Figure 4a shows the front and back sides of the device. The device functions at 3.3 V and is powered by a 2000 mAh battery.

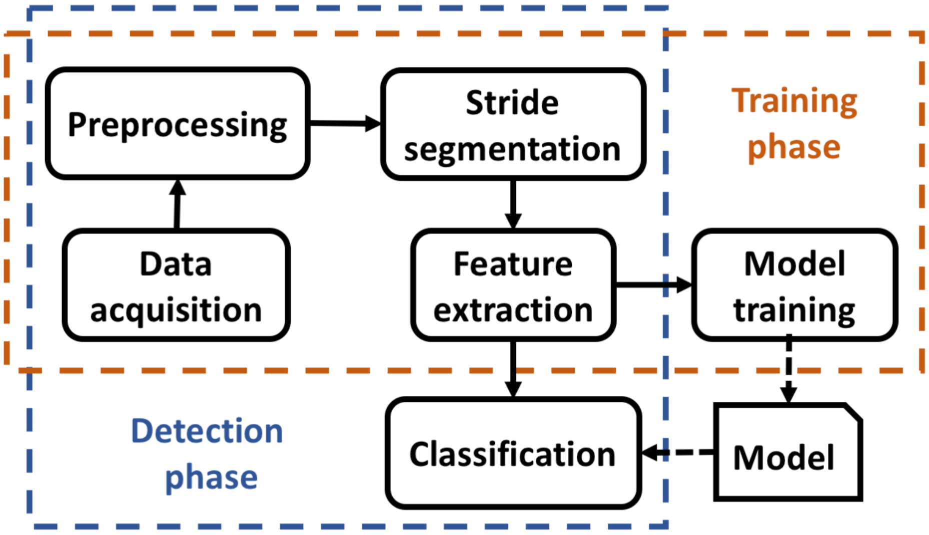
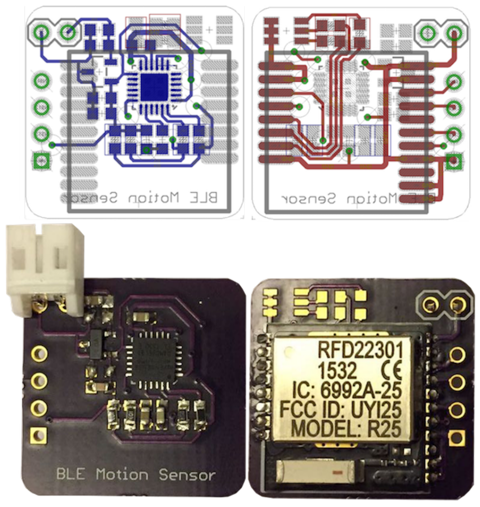
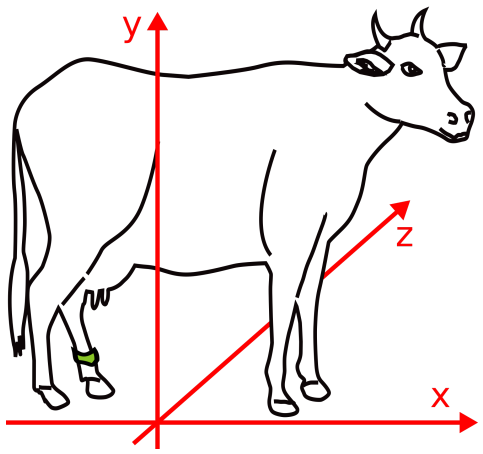


Figure 3. Overview of the *Training* and *Detection* phases. The features extracted are used during the *training phase* to train a machine learning model which is used in the *detection phase* to classify new feature vectors.



(**a**)



(**b**)

Figure 4. Motion sensor and Eagle schematics for front and back sides of the Printed Circuit Board (PCB) (**a**) and the orientation of our device (**b**). Our device is oriented such that the *y*-axis represents vertical accelerations, the *x*-axis is parallel to the cow (i.e., in its walking direction) and the *z*-axis is lateral (i.e., left and right) to a cow.

The enclosure we designed is shown in Figure 1b. We designed the enclosure to have rounded corners and edges to avoid injuries to cows. The device has been printed with the Selective Laser Sintering (SLS) 3D printing technique. This printing technique produces robust and relatively lightweight objects with thin sides. We designed the enclosure in two parts such that a rubber seal can be fit between them in order to make the enclosure water-proof to protect the electronic device and battery from urine and excrement. According to a veterinarian, the sensor enclosure is "*robust, yet thin and lightweight for cows to wear*" and "*should be able to resist forces and strain caused by other cows stepping on it*".

6.2. Data Acquisition

The sensor device measures linear acceleration (without gravity) and orientation (yaw, pitch, roll) at 100 Hz along 3 axes (*x*,*y*,*z*). The accelerometer range is set to ±4 g. Data is obtained with an analog-digital converter resolution of 16 bits. The orientation of the device is depicted in Figure 4b. Data is stored in the device's flash memory and processed every 128 samples (1.28 s). Figure 5a illustrates how the data acquired by the device correlate to different stride phases.

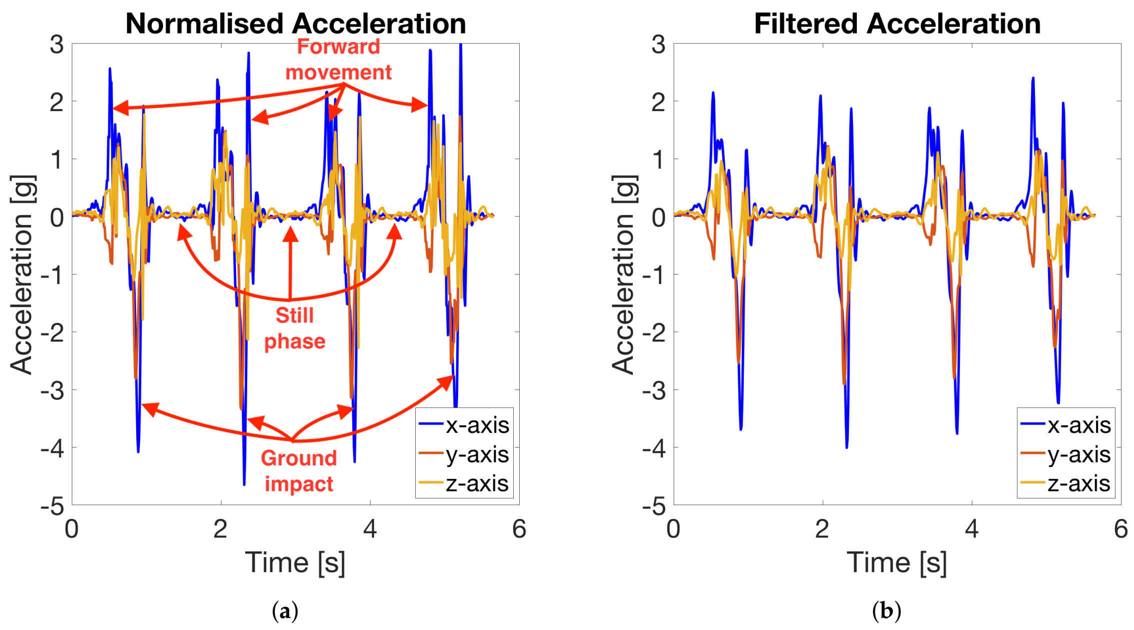


Figure 5. Raw acceleration (**a**) and low-pass filtered acceleration (**b**) for four strides. Forward movements during a stride cause a positive acceleration along the *x*-axis. Hoof impacts with the ground can be seen as peaks in the acceleration, particularly along the *x*- and *y*-axes. Periods while the hoof is in contact with the ground (still phases) have almost zero acceleration.

6.3. Preprocessing

In the preprocessing stage, we filter the signal acquired by the sensor in order to eliminate noise (i.e., information in the signal acquired not related to a cow's gait). Noise might be introduced by the sensor device (e.g., shaking at high frequencies caused by the accelerometer) or by sudden movements such as when a cow is bumped by another cow in the stable. Therefore, we apply a first order Butterworth low-pass IIR filter with a *cutoff frequency* of 20 Hz to the linear acceleration signal. This filter leaves frequencies in the range 0–20 Hz almost unmodified and attenuates frequencies higher than 20 Hz. A comparison of the signal before and after applying the filter is shown in Figure 5.

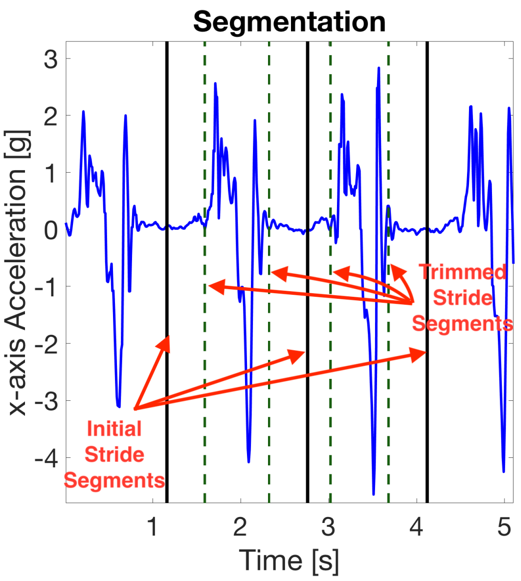
6.4. Stride Segmentation

The purpose of the *Stride Segmentation* is to detect the beginning and ending of a stride. We segment the strides based on a peak detector. We use the acceleration along the *x*-axis based on our observation that this axis has the strongest acceleration range during strides (as can be seen in Figure 5). The *Stride Segmentation* is done in the following three steps:

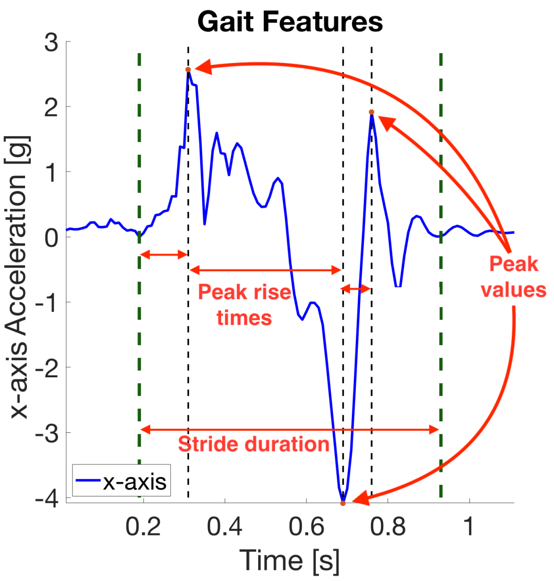
1. Every stride has two upper peaks. We detect the highest peak with a peak detection algorithm. We ignore peaks that are less than 60 samples away from a previously detected peak. This also filters out periods when cows did not walk.

2. Every stride is preceded by periods of small variance in acceleration. We find these periods by searching for the 9-sample window with smallest variance in acceleration among the 70 samples before and after the detected peak. We call the center of these windows *initial stride segments*.
3. Between two initial stride segments, additional samples are included that might not belong to a stride. Therefore, we trim the stride by shifting the initial stride segments towards the peak detected in step 1. The initial stride segments are shifted until the standard deviation of a 6-sample window centered at the shifted stride segment is larger than a constant θ. We found $\theta = 0.2$ empirically.

Figure 6a shows the linear acceleration along the *x*-axis of four consecutive strides with annotations pointing at initial and trimmed stride segments.



(**a**)



(**b**)

Figure 6. Initial and trimmed segments detected with our stride segmentation algorithm applied to linear acceleration along the *x*-axis (**a**) and illustration of the gait features on the linear acceleration along the *x*-axis (**b**).

6.5. Feature Extraction

For each stride segmented, we compute a set of gait and statistical features. Gait features are measurements specific of a stride. Every stride is characterized by three peaks: two upper peaks and one lower peak. We first detect all three peaks. If any of the peaks could not be found, we ignore the stride. This might happen if the cow shortly lifted a leg or got bumped by another cow. For all three peaks, we compute its peak value and rise time. The rise times are computed as the difference in samples to the previous peak. The rise time of the first peak is computed as the difference in samples to the first sample in the trimmed stride segment. In addition, the total duration of the stride is added to the feature set. Figure 6b illustrates how the gait features are computed based on the three peaks of a single stride.

Statistical features are measures to extract information from data sets. We extract the following statistical features: mean, median, standard deviation, Zero Crossing Rate (ZCR), Peak-to-Peak amplitude (P2P), Root Mean Square (RMS) and Average Acceleration Variation (AAV) for every stride. ZCR is a measure of the amount of times a signal crosses the zero value. A high ZCR might indicate a highly intense or periodic activity. P2P is the difference between the maximum and minimum acceleration value in a stride and provides information about the intensity of a stride. RMS is the square root of the mean of the values in a stride squared. This measurement provides information about the amount of acceleration and variation in a stride. AAV is calculated as the sum of the absolute differences between consecutive samples in a stride normalized by the number of samples. AAV provides an indication of how sudden changes in acceleration happen within a stride. These measurements are commonly used for activity recognition applications and have been recently used for fall-detection and gait analysis in humans [27,28].

The list of gait and statistical features are enumerated in Table 2. Gait features are computed on linear acceleration and statistical features are computed on linear acceleration, rotation and magnitude of acceleration. ZCR is only computed on the linear acceleration. This gives us a total of 21 gait features and 45 statistical features per stride. A window might contain several strides. We average the features extracted from the same window. A vector containing the 66 averaged features is called *stride instance*.

6.5.1. Feature Normalization

After extracting these features, we normalize every feature in the stride instance to have zero mean and a standard deviation in the range $[-1, 1]$. We do this by subtracting the mean and dividing by the standard deviation of every stride instance used during the *training phase*. This is a required computation for the machine learning algorithms. Not normalizing the features could cause features with a large range of values to have a larger influence on the outcome of the classification.

6.5.2. Feature Grouping

After extracting and normalizing the features, we group consecutive stride instances by computing the average of each feature F_i in γ consecutive strides. Reducing the number of stride instances that have to be classified leads to less computations and a longer battery life. Furthermore, we observed that grouping stride instances increased the accuracy of the classification. We determined $\gamma = 3$ empirically.

6.6. Model Training and Classification

The *Model Training* step trains a machine learning algorithm to classify stride instances as *normal* or *abnormal*. It should be noted that *Model Training* and *Classification* are two different steps, as shown in our overview in Figure 3. We describe both steps in this section because they are closely related to each other.

To classify stride instances into *normal* and *abnormal*, we use a Support Vector Machine (SVM) classifier. SVM is a classification algorithm that calculates a boundary that maximizes the distance between instances of two different classes in an N-dimensional space. A one-class SVM classifier finds a boundary around instances of one class and classifies new observations based on their distance to this boundary. We train a one-class SVM classifier using *normal* stride instances.

The boundary of the classifier is defined such that a fraction ω of the instances is classified as *abnormal*. The constant ω is used to define how 'compact' the boundary around *normal* stride instances is. A smaller ω leads to more stride instances classified as *abnormal* and a larger ω leads to more instances classified as *normal*. The one-class SVM computes a distance $dist(S_i)$ for each new stride instance S_i. Our SVM classifier classifies stride instances as *normal* if $dist(S_i) < \tau$ or as *abnormal* otherwise. Figure 7 sketches the meaning of parameters ω and τ.

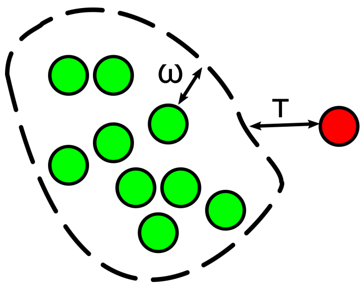


Figure 7. Green and red dots represent *normal* and *abnormal* stride instances, respectively. ω determines the distance of the boundary to the *normal* stride instrances. τ is a threshold to the distance of stride instances: stride instances with a distance larger than τ are classified as *abnormal*.

Table 2. Gait and statistical features used by our approach. Features labeled as *accel* are computed on all three axes of the linear acceleration and features labeled as *all* are computed on every axis of the linear acceleration, rotation and on the magnitude vector of the linear acceleration.

	Feature	**Signal**	**#**
Gait	peak values	accel	9
	rise times	accel	9
	stride duration	accel	3
Statistical	mean	all	7
	median	all	7
	STD	all	7
	ZCR	accel	3
	P2P	all	7
	RMS	all	7
	AAV	all	7
	Total		66

7. Evaluation

In this section, we present and discuss our results. Our approach intends to detect abnormal gait. Therefore, our positive class is the class of *abnormal* stride instances. We define the following variables:

True Positive (TP) Amount of *abnormal* stride instances classified as such.
True Negative (TN) Amount of *normal* stride instances classified as such.
False Positive (FP) Amount of *normal* stride instances classified as *abnormal*.
False Negative (FN) Amount of *abnormal* stride instances classified as *normal*.

We validated our results based on the metrics: accuracy, specificity and sensitivity, defined as follows:

- Accuracy: The ability of our approach to classify stride instances correctly. It answers the question: *"what percent of the classified stride instances is correct?"*. Accuracy is calculated as: $(TN + TP)/(TN + TP + FN + FP)$.
- Specificity: The ability of our approach to identify *normal* stride instances. It answers the question: *"when a cow walks normally, what percent of its stride instances does our approach classify as 'normal'?"*. This is also referred to as "true negative rate" and computed as: $TN/(TN + FP)$.
- Sensitivity: The ability of our approach to identify *abnormal* stride instances. It answers the question: *"when a cow walks abnormally, what percent of its stride instances does our approach classify as 'abnormal'?"*. This is also referred to as "true positive rate" and computed as: $TP/(TP + FN)$.

We computed the accuracy, specificity and sensitivity for a particular cow by means of the leave-one-out cross-validation technique as follows:

1. We trained the SVM algorithm with N-1 *normal* stride instances, where N is the total number of *normal* stride instances for a specific cow.
2. We used the model to classify the *normal* stride instance that was not used to train the algorithm and every *abnormal* stride instance.
3. We repeated steps 1 and 2 N times; each time we left out a different *normal* stride instance.
4. We averaged the accuracies, specificities and sensitivities computed in step 3.

7.1. Results

Table 3 shows the accuracy, specificity and sensitivity of our approach for each cow. We used the parameters: $\omega = 0.15$ and $\tau = -0.6$. We found these parameters empirically with the goal to maximize the average accuracy of the classification for all cows. According to these results, our approach has an average accuracy of 91.1% (specificity: 91.6% and sensitivity: 74.2%). These results imply that our approach would classify 8.4% of the stride instances of cows walking normally as *abnormal*. In contrast, when cows do indeed walk abnormally, our approach would classify 74.2% of their stride instances as *abnormal*.

Table 3. Accuracy, specificity and sensitivity of our approach at detecting *abnormal* stride instances. Parameters used were $\omega = 0.15$ and $\tau = -0.6$.

#	Accuracy	Specificity	Sensitivity
1	95.8%	96.3%	83.3%
2	94.6%	95.1%	78.6%
3	81.9%	82.0%	78.6%
4	96.4%	97.2%	66.7%
5	97.3%	97.6%	80.0%
6	81.3%	81.7%	70.6%
7	95.4%	96.6%	62.5%
8	92.6%	93.0%	77.8%
9	87.2%	87.6%	73.1%
10	88.2%	88.7%	70.6%

7.2. Discussion

The results we obtained suggest that our approach is able to detect a deviation from a cow's usual walking pattern after this deviation occurs. These results meet the requirements of the system we propose, as stride classifications are meant to be tracked over time rather than used in isolation. In particular, missing steps of a lame cow (i.e., a low sensitivity) is acceptable because a cow will perform several steps a day even if it is lame (e.g., to get access to food). The lowest sensitivity obtained by our approach was 62.5% (for cow #7). That means 62.5% of its steps would still be detected and could be used to determine whether the cow needs treatment. Furthermore, our approach would be able to emit alerts if a cow should reach an advanced stage of a lameness disease that prevents it from walking altogether. On the other hand, a low specificity could make veterinarians loose trust in our system. The lowest specificity we obtained was 81.7% (for cow #6). This indicates 18.3% of its strides would be detected as abnormal when the cow is actually walking normally. However, 18.3% abnormal step detection would be usual for this cow, and would suddenly rise to 70.6% when it becomes lame. Figure 8 shows an example illustrating how the gait for a particular cow could be shown by a user interface.

We argue that the deviation from normal gait caused by lameness is more radical than the change in gait caused by the plastic block which we used to obtain a ground truth set in our study. This is because cows suffering lameness will try to avoid pain by bearing as little weight as they need to on the affected hoof. As a consequence, lame cows perform considerably shorter strides or stop using one limb all together. This causes an asymmetry in the gait, which is observable visually. In contrast, the change caused by the plastic block is more subtle. We were not able to assess visually whether a cow was walking with a plastic block or not by looking at its gait. As a consequence, we believe our approach might be more accurate at detecting deviations in gait caused by lameness than those caused by a plastic block.

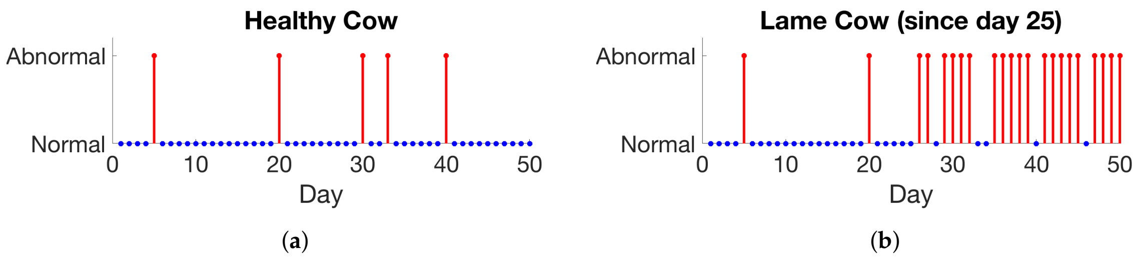


Figure 8. Visualization of the average classification performance of our approach for a cow walking normal (**a**) and after it becomes lame (**b**).

8. Ethical Considerations

Our research required cows to take part in an experiment. In order to ensure an ethically appropriate treatment of the cows during our experiment, we designed it based on the ethical guidelines proposed by Mancini [6] as follows:

1. *Respecting and caring for every participant without discrimination.* The participants of this experiment were cows of different ages and breeds. We did not harm any of the them or make any discrimination as for the selection of the specific cow subjects or treatment they received during the experiment.
2. *Garnering participants mediated and contingent consent.* We conducted this experiment together with a professional veterinarian team who are the legal representatives of the cows that participated in the experiment. Both veterinarians know the needs and welfare requirements of these cows and gave us their consent to conduct the experiment. Furthermore, they accompanied and supported us throughout the entire experiment to ensure these requirements were met.
3. *Doing research that is relevant to participants and consistent with their welfare.* The results of our research suggest that it is possible to automatically detect a condition that is painful for cows and highly detrimental to their health (e.g., might lead to death if not treated early enough). Therefore, our research has the potential to benefit the individual cows that participated in the experiment, as well as other cows. This research was conducted in the natural environment of the participating cows, an indoor stable located in the outskirts of Munich, Germany.
4. *Avoiding research procedures that may be harmful to participants.* According to the veterinarians that supported us throughout this study, attaching a sensor device and plastic block to cows and encouraging them to walk for less than 10 min did not cause any lasting harm to these cows. Veterinarians trimmed cows before attaching the plastic block to ensure the block was placed and fit properly to the claw. Trimming cow claws is a procedure undertaken to maintain a healthy hoof condition and prevent injury and disease. In addition, we limited the walking sessions to a maximum of 10 min per day and continued the data recording on a different day in order to reduce the level of fatigue caused to the cows.

5. *Assessing research proposals and obtaining expert support.* The cow interventions performed in this study were done by professional veterinarians and were approved by the ethics committee of the Ludwig Maximilian University (LMU) in Munich, Germany to ensure no harm was done to the cows.

9. Conclusions

We presented and evaluated a system to detect changes in cows' usual gait that might occur due to a lameness-related disease. Our approach considers the differences in gait of a cow by comparing its walking pattern to a baseline established for that particular cow during the first hours of use. Our system could be used by veterinarians to keep track of the health condition of the cows in a herd. In particular, veterinarians might decide to examine a cow if the number of detected *abnormal* stride instances has exceeded considerably the usual amount for that particular cow.

In the future, our approach will have to be validated in a longer-term field study. In particular, it would have to be studied how veterinarians use our system in practice (e.g., how much they trust our system even in the presence of false positive detections). Furthermore, the usage of our system be validated for a longer period of time to study possible long-term effects that we did not observe during our first study. Furthermore, the energy consumption and battery duration of the wearable device should be optimized in a way that does not affect the accuracy of the stride classification.

Author Contributions: Ju.H., S.N. and B.B. conceived the experiments to collect cow gait data; Ju.H. and Jo.H. analyzed the data and developed the algorithms; Ju.H. wrote this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Van Nuffel, A.; Zwertvaegher, I.; Van Weyenberg, S.; Pastell, M.; Thorup, V.M.; Bahr, C.; Sonck, B.; Saeys, W. Lameness detection in dairy cows: Part 2. Use of sensors to automatically register changes in locomotion or behavior. *Animals* **2015**, *5*, 861–885. [CrossRef] [PubMed]
2. Nielsen, L.R.; Pedersen, A.R.; Herskin, M.S.; Munksgaard, L. Quantifying walking and standing behaviour of dairy cows using a moving average based on output from an accelerometer. *Appl. Anim. Behav. Sci.* **2010**, *127*, 12–19. [CrossRef]
3. Mazrier, H.; Tal, S.; Aizinbud, E.; Bargai, U. A field investigation of the use of the pedometer for the early detection of lameness in cattle. *Can. Vet. J.* **2006**, *47*, 883. [PubMed]
4. Haladjian, J.; Hodaie, Z.; Nüske, S.; Brügge, B. Gait Anomaly Detection in Dairy Cattle. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction*; ACM: New York, NY, USA, 2017; pp. 8:1–8:8. [CrossRef]
5. Haladjian, J.; Brügge, B.; Nüske, S. An approach for early lameness detection in dairy cattle. In Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers, Maui, HI, USA, 11–15 September 2017; pp. 53–56.
6. Mancini, C. Towards an animal-centred ethics for Animal–Computer Interaction. *Int. J. Hum.-Comput. Stud.* **2017**, *98*, 221–233. [CrossRef]
7. Hertem, T.V.; Maltz, E.; Antler, A.; Romanini, C.E.B.; Viazzi, S.; Bahr, C.; Schlageter-Tello, A.; Lokhorst, C.; Berckmans, D.; Halachmi, I. Lameness detection based on multivariate continuous sensing of milk yield, rumination, and neck activity. *J. Dairy Sci.* **2013**, *96*, 4286–4298. [CrossRef] [PubMed]
8. De Mol, R.M.; André, G.; Bleumer, E.J.B.; der Werf, J.T.N.; De Haas, Y.; Van Reenen, C.G. Applicability of day-to-day variation in behavior for the automated detection of lameness in dairy cows. *J. Dairy Sci.* **2013**, *96*, 3703–3712. [CrossRef] [PubMed]
9. Poursaberi, A.; Bahr, C.; Pluk, A.; Van Nuffel, A.; Berckmans, D. Real-time automatic lameness detection based on back posture extraction in dairy cattle: Shape analysis of cow with image processing techniques. *Comput. Electron. Agric.* **2010**, *74*, 110–119. [CrossRef]

10. Song, X.; Leroy, T.; Vranken, E.; Maertens, W.; Sonck, B.; Berckmans, D. Automatic detection of lameness in dairy cattle-Vision-based trackway analysis in cow's locomotion. *Comput. Electron. Agric.* **2008**, *64*, 39–44. [CrossRef]
11. Pluk, A.; Bahr, C.; Poursaberi, A.; Maertens, W.; Van Nuffel, A.; Berckmans, D. Automatic measurement of touch and release angles of the fetlock joint for lameness detection in dairy cattle using vision techniques. *J. Dairy Sci.* **2012**, *95*, 1738–1748. [CrossRef] [PubMed]
12. Van Hertem, T.; Viazzi, S.; Steensels, M.; Maltz, E.; Antler, A.; Alchanatis, V.; Schlageter-Tello, A.A.; Lokhorst, K.; Romanini, E.C.B.; Bahr, C.; et al. Automatic lameness detection based on consecutive 3D-video recordings. *Biosyst. Eng.* **2014**, *119*, 108–116. [CrossRef]
13. Eddy, A.L.; Van Hoogmoed, L.M.; Snyder, J.R. The role of thermography in the management of equine lameness. *Vet. J.* **2001**, *162*, 172–181. [CrossRef] [PubMed]
14. Pastell, M.; Kujala, M. A Probabilistic Neural Network Model for Lameness Detection. *J. Dairy Sci.* **2007**, *90*, 2283–2292. [CrossRef] [PubMed]
15. Maertens, W.; Vangeyte, J.; Baert, J.; Jantuan, A.; Mertens, K.C.; De Campeneere, S.; Pluk, A.; Opsomer, G.; Van Weyenberg, S.; Van Nuffel, A. Development of a real time cow gait tracking and analysing tool to assess lameness using a pressure sensitive walkway: The GAITWISE system. *Biosyst. Eng.* **2011**, *110*, 29–39. [CrossRef]
16. Chapinal, N.; Tucker, C.B. Validation of an automated method to count steps while cows stand on a weighing platform and its application as a measure to detect lameness. *J. Dairy Sci.* **2012**, *95*, 6523–6528. [CrossRef] [PubMed]
17. Van Nuffel, A.; Vangeyte, J.; Mertens, K.C.; Pluym, L.; De Campeneere, S.; Saeys, W.; Opsomer, G.; Van Weyenberg, S. Exploration of measurement variation of gait variables for early lameness detection in cattle using the GAITWISE. *Livest. Sci.* **2013**, *156*, 88–95. [CrossRef]
18. Der Tol, P.P.J.; Metz, J.H.M.; Noordhuizen-Stassen, E.N.; Back, W.; Braam, C.R.; Weijs, W.A. The pressure distribution under the bovine claw during square standing on a flat substrate. *J. Dairy Sci.* **2002**, *85*, 1476–1481. [CrossRef]
19. Thorup, V.M.; Munksgaard, L.; Robert, P.E.; Erhard, H.W.; Thomsen, P.T.; Friggens, N.C. Lameness detection via leg-mounted accelerometers on dairy cows on four commercial farms. *Animal* **2015**, *9*, 1704–1712. [CrossRef] [PubMed]
20. Alsaaod, M.; Römer, C.; Kleinmanns, J.; Hendriksen, K.; Rose-Meierhöfer, S.; Plümer, L.; Büscher, W. Electronic detection of lameness in dairy cows through measuring pedometric activity and lying behavior. *Appl. Anim. Behav. Sci.* **2012**, *142*, 134–141. [CrossRef]
21. Yunta, C.; Guasch, I.; Bach, A. Short communication: Lying behavior of lactating dairy cows is influenced by lameness especially around feeding time. *J. Dairy Sci.* **2012**, *95*, 6546–6549. [CrossRef] [PubMed]
22. Pastell, M.; Tiusanen, J.; Hakojärvi, M.; Hänninen, L. A wireless accelerometer system with wavelet analysis for assessing lameness in cattle. *Biosyst. Eng.* **2009**, *104*, 545–551. [CrossRef]
23. Chapinal, N.; De Passillé, A.M.; Rushen, J.; Wagner, S. Automated methods for detecting lameness and measuring analgesia in dairy cattle. *J. Dairy Sci.* **2010**, *93*, 2007–2013. [CrossRef] [PubMed]
24. Chapinal, N.; de Passille, A.M.; Pastell, M.; Hänninen, L.; Munksgaard, L.; Rushen, J. Measurement of acceleration while walking as an automated method for gait assessment in dairy cattle. *J. Dairy Sci.* **2011**, *94*, 2895–2901. [CrossRef] [PubMed]
25. Haladjian, J.; Ermis, A.; Hodaie, Z.; Brügge, B. iPig: Towards Tracking the Behavior of Free-roaming Pigs. In Proceedings of the Fourth International Conference on Animal-Computer Interaction; Milton Keynes, UK, 21–23 November 2017; pp. 10:1–10:5. [CrossRef]
26. Paci, P.; Mancini, C.; Price, B.A. Towards a wearer-centred framework for animal biotelemetry. In Proceedings of the Measuring Behaviour 2016, Dublin, Ireland, 25–27 May 2016.
27. Cola, G.; Avvenuti, M.; Vecchio, A.; Yang, G.Z.; Lo, B. An on-node processing approach for anomaly detection in gait. *IEEE Sens. J.* **2015**, *15*, 6640–6649. [CrossRef]
28. Abbate, S.; Avvenuti, M.; Bonatesta, F.; Cola, G.; Corsini, P.; Vecchio, A. A smartphone-based fall detection system. *Pervasive Mob. Comput.* **2012**, *8*, 883–899. [CrossRef]

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

6.7 KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries

This publication describes a smart textile bandage and companion iPad and Apple Watch applications to track the performance of different rehabilitation exercises performed by patients recovering from an Anterior Cruciate Ligament (ACL) injury. This paper presents a novel calibration technique to measure the degree of flexion of a leg using a motion sensor, describes the signal processing methods used by the system and provides an evaluation of the usability of the system.

The author of this Habilitation led the development of the system, studied different activity recognition methods to assess the execution of the different rehabilitation exercises and wrote the paper.

Authors	Haladjian, J., Bredies, K., & Bruegge, B.
Conference	International Conference on Wearable and Implantable Body Sensor Networks (BSN)
Number of Pages	4
Type	Short Paper
Review	Peer Reviewed (3 Reviewers)
Year	2018
DOI	10.1109/BSN.2018.8329646

2018 IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN)
4-7 March 2018
Las Vegas, Nevada, USA

KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries

Juan Haladjian[1], Katharina Bredies[2] and Bernd Brügge[1]

***Abstract*— Patients of an Anterior Cruciate Ligament (ACL) injury engage in a set of rehabilitation exercises to recover mobility and strength of their injured leg. We developed a smart bandage and user interface that provides live feedback to patients while exercising and computes a series of performance metrics used by orthopedists for assessment of the patient's recovery. The bandage uses smart textile components such as elastic conductive threads and snap buttons. We discuss how the smart textile components contribute to desirable properties of the bandage, such as robustness, washability and user comfort. Furthermore, we present a technique for calibrating the motion sensors on the bandage which is suitable for patients of ACL with limited mobility. Finally, we present the results of a controlled experiment with 10 patients with the goal to assess the accuracy of the bandage's measurements.**

I. INTRODUCTION

Tear of Anterior Cruciate Ligament (ACL) is a severe knee injury that occurs mostly among athletes. The rehabilitation after the injury can last as long as a year and often includes physical therapy, strength exercises and frequent visits to physiotherapists and doctors. Successful recovery of an ACL injury relies on the appropriate execution of rehabilitation exercises with the goal of recovering full range of motion, strength and coordination.

Currently, patients sustaining an ACL injury perform the rehabilitation exercises mostly unsupervised and lack quantitative ways to measure the quality and track performance of their exercising. Orthopedists also lack tools to assess patients' rehabilitation progress and still have to rely on subjective observations. Furthermore, orthopedists and patients meet at time intervals as long as three months and the treatment is decided upon observations during these meetings without consideration of the patient's recovery progress in the periods between visits.

In this paper, we introduce KneeHapp Textile, a smart compression bandage that supports different rehabilitation exercises performed by patients recovering from an ACL injury in order to recover flexibility and muscle strength. We address the construction and integration of textile sensors and connections and propose software solutions to quantify the progress of the rehabilitation after the ACL injury. We present KneeHapp's user interface on an iPad and Apple Watch.

[1]Juan Haladjian and Bernd Brügge are with Chair for Applied Software Engineering, Faculty of Computer Science at the Technical University Munich
[2]Katharina Bredies is with the Design Research Lab at the University of Arts Berlin
Corresponding author is Juan Haladjian: haladjia@in.tum.de

II. Related Work

Different studies have investigated rehabilitation using wearables and mobile devices [5], [2]. In contrast, only a few studies have used smart textiles to support rehabilitation after an injury [3], [4], [1]. In addition, most of the research in the field of computer-assisted rehabilitation addresses injuries other than ACL. In contrast to other injuries (e.g. osteoarthritis), patients recovering from ACL injury are athletes that want to take up their training routines after recovery. As a consequence, the rehabilitation of an ACL injury involves a series of exercises to recover mobility and strength of the injured leg and to ensure the athlete is ready to go back to sports. We investigate the application of smart textile technologies to exercise-intensive rehabilitation processes.

III. Requirements

Based on a series of interviews with two professional orthopedists who conduct ACL reconstruction surgeries on a daily basis, we identified a set of rehabilitation exercises performed by patients recovering from an ACL injury:

- *Knee bends*. After the surgery, patients perform mostly knee bends to recover mobility of their knees.
- *Squats*. During the first weeks after the surgery, patients suffer from muscular atrophy on the injured leg. A common exercise for strength recovery are knee squats and different variations of it.
- *One-leg hop*. Towards the end of the rehabilitation, orthopedists should assess whether patients are ready to start doing sports again. One-leg hop is an exercise in which patients should jump forward on one leg as far as possible and land stably.
- *Side hops*. Another exercise to assess patients performance and strengthen muscle is side hops. Side hops requires patients to hop side-wise on one leg over a distance of 20-30 cm during a period of 10 to 60 seconds.

IV. KneeHapp

KneeHapp measures the performance and provides live feedback to patients while performing different ACL rehabilitation exercises. The KneeHapp system consists of a smart compression bandage, an iPad App. The smart compression bandage acquires and processes sensor data and delivers the computed results to the iPad App via Bluetooth Low Energy. The iPad App displays live feedback about the quality of the rehabilitation exercises and keeps track of the rehabilitation performance. An additional interface on the Apple Watch lets

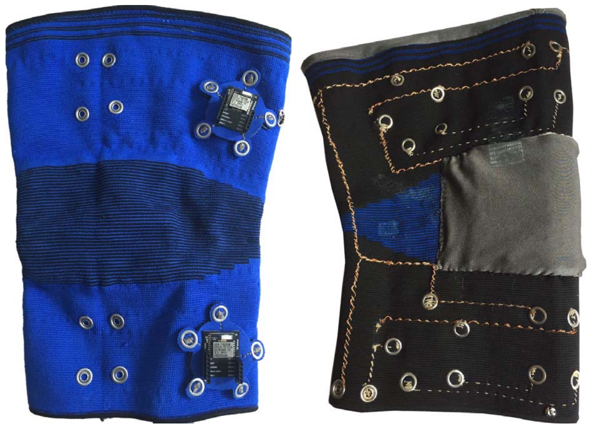


Fig. 1: Outer layer (left) and inner layer (right) of the KneeHapp bandage.

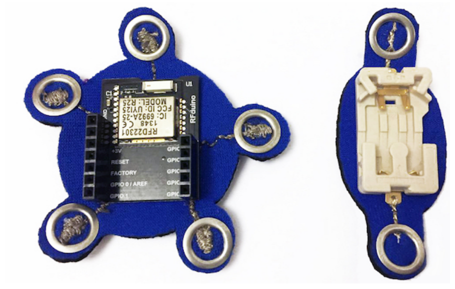


Fig. 2: Smart textile patches hosting a microcontroller (left) and a battery holder (right).

patients to control specific parameters during an exercising session.

A. *Smart Textile Bandage*

We adapted a conventional compression bandage used by patients to reduce swelling after a knee surgery. We used this compression bandage as a substrate for integration of electronics. The outer side of the bandage hosts two motion sensors, two microcontrollers and a coin cell battery holder. The inner side contains an electric circuit made of elastic conductive threads. An additional sleeve in the inner side protects the circuits from sweat and can be removed for washing purposes. Figure 1 shows outer and inner layers of the bandage.

We decided to connect the electronics to the electric circuit using elastic conductive thread based on three main design goals: robustness, user comfort and low energy consumption. Elastic conductive threads are less prone to snapping when compared to other materials to connect electronic devices such as conventional cables or conductive thread. This is of particular relevance because connections laid along the knee are prone to a considerable amount of strain while patients perform exercises involving knee bends. We also considered a design involving two separate microcontroller units that transmit data wirelessly. We discarded this design because the need for wireless data transmission to synchronize the signals produced by upper and lower microcontroller units would increase energy consumption rates considerably. Furthermore, conductive threads do not constrain user movements more than the actual bandage.

All conventional electronic elements are sewn on smart textile patches and equipped with snap buttons as conductive contacts to the main bandage. This enables users to remove the electronics for replacement or washing purposes. Figure 2 shows two smart textile patches.

B. *Software Computations*

KneeHapp supports different rehabilitation exercises. This section describes how KneeHapp processes data from the aforementioned sensors in order to provide feedback to patients while exercising and compute performance metrics that can be tracked over time.

1) Range of Motion: KneeHapp calculates the angle of flexion of the leg based on the Euler angles computed by lower and upper IMUs. KneeHapp supports two calibration approaches. By convention, the angle of flexion of the leg is equal to zero when the leg is relaxed on a flat surface. Therefore, one calibration approach supported by KneeHapp determines the sensor alignment while patients extend their leg on a flat surface. However, most patients are not able to fully extend their leg after the surgery. In order to address this issue, we considered different calibration approaches proposed in the literature, including pose (i.e. users perform a predefined pose) and functional (i.e. users perform specific movements) calibration. But since patients might not be able to perform specific poses or movements after surgery, we devised a novel calibration approach that consists of three steps:

1) Wear the bandage on the healthy leg and measure the orientation of the IMU while the leg is laid on a flat surface.
2) Measure the angle of flexion while the leg is slightly bent by placing the back of the knee on top of any object. This provides a calibration offset.
3) To 'transfer' the calibration offset to the injured leg, the user should wear the bandage on the injured leg and measure the angle of flexion while placing the back of the knee on top of the same object.

These steps are illustrated in Figure 3. Because the angle of flexion is measured on top of the same object for both legs, the angle of flexion for both legs is, by definition, the same. The difference between the measured angles of flexion is used as an offset and added to the angles measured on the injured leg. This calibration approach does not require additional equipment such as wedges or individuals who perform the measurements. KneeHapp provides visual feedback live to patients about their current angle of flexion and about the maximal angles of flexion, extension, and hyper-extension (i.e. extending a limb over 180°) of the leg, as shown in Figure 4.

2) One-leg Squat: KneeHapp provides live feedback to patients while performing one-leg squats in three ways. First, it calculates the angle of flexion of the leg during the squat and triggers a visual and auditive feedback when patients

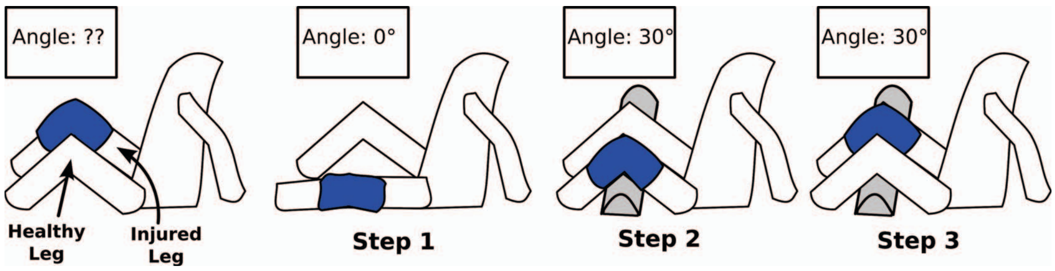


Fig. 3: Illustration of our calibration procedure with an example. In step 1, KneeHapp is calibrated on the healthy leg, which the patient can extend on a flat surface. In step 2, KneeHapp calculates the angle of flexion while the leg is placed above an object. In step 3, KneeHapp assumes the angle of flexion of the injured leg is the same as on the healthy leg when placed on the same object. After step 3, patients can begin to calculate the range of motion of their injured knee.

achieve a squat angle of 60°. Second, it computes the degree of medial collapse of a patient's knee and warns patients in case they reach a threshold of 10°. Third, it computes the degree of shaking of the leg, which is computed as the standard deviation of the magnitude of the linear acceleration produced by the IMUs.

3) One-leg Hop: KneeHapp measures and keeps track of the performance of one-leg hops by comparing the duration of the hop done with each leg. This is done in two steps. First, a second order Butterworth low-pass filter is applied to reduce noise in the signal acquired by the upper motion sensor with a $cutoff = 15Hz$. Second, the different phases of the jump (i.e. jumping, flying, landing) are estimated using a pre-established set of thresholds calibrated to each phase.

4) Side Hops: KneeHapp counts the number of side hops performed by patients in a configurable period of time. This is done in three steps: *preprocessing*, *detection* and *disambiguation*. In the *preprocessing* step, the linear acceleration along the vertical and forward axes are filtered using a Resistor-Capacitor (RC) low-pass filter with a time constant $\tau = 0.25$. Usually, each hop performed by the user produces a high and a low peak in the signal. The *detection* step uses a peak detection algorithm to count the number of high and low peaks in the filtered signal. In order to avoid counting bumps caused when patients do not land stably after a hop, KneeHapp ignores peaks with a distance smaller or equal to 15 samples to a previously detected peak. The *disambiguation* step compares the number of lower and upper peaks detected and returns the median.

C. User Interface

KneeHapp's user interface is displayed on an iPad and Apple Watch. The iPad enables patients to keep track of their rehabilitation progress and provides live feedback while exercising. Figure 4 shows a screenshot of the iPad interface displaying the angles of flexion, extension and hyperextension computed by KneeHapp.

The interface on the Apple Watch enables patients to start and stop the exercises and provides feedback about the quality of the performed exercises. Starting and stopping exercises directly from the wrist is convenient at the beginning of the rehabilitation because patients might have limited mobility and towards the end because patients perform squats and jump-based exercises. The interface on the Apple Watch also displays exercise countdowns, angles of flexion of the leg and indicates when specific angles have been reached (e.g. medial collapse) by means of vibrations and sound. Figure 5 shows two screenshots of the interface on the Apple Watch for the *Range of Motion* and *Squat* exercises.

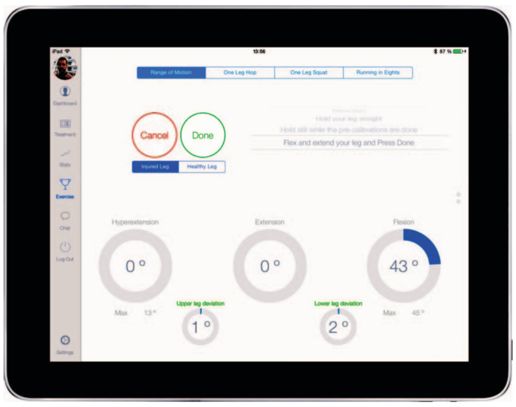


Fig. 4: iPad App displaying the *Exercises* tab. The *Exercises* tab provides live feedback about the performance of the selected exercise.

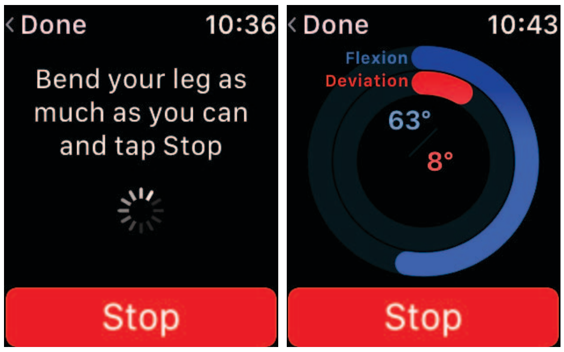


Fig. 5: Interface on the Apple Watch displaying instructions to start the range of motion measurement (left) and the computed angles during a squat (right).

TABLE I: Answers to a usability questionnaire given to patients. -2: *Strongly disagree*; -1: *Disagree*; 0: *Neutral*; +1: *Agree*; +2: *Strongly Agree*.

	-2	-1	0	+1	+2
KneeHapp provides me with helpful information during the Range of Motion exercise	0	0	0	2	3
KneeHapp provides me with helpful information during the One-Leg Squat exercise	0	0	0	3	2
KneeHapp helps me to perform the One-Leg Squat exercise correctly	0	0	0	4	1
The iPad and Watch App were easy to use	0	0	0	4	1
The bandage was comfortable to wear	0	0	1	3	1
KneeHapp would motivate me to perform my rehabilitation exercises	0	0	1	1	3
I would use KneeHapp	0	0	0	3	2

V. Controlled Experiment

We evaluated the accuracy of the computations performed by KneeHapp in a controlled experiment where we collected sensor data from 10 patients (3 female, 7 male, age range: 20-26, one of them was 51 years old) at different stages in the recovery after an ACL injury while performing the following exercises: range of motion measurements, one-leg hops and side-hops, as follows:

1) Range of Motion: We calibrated KneeHapp with our calibration approach and measured four different angles of flexion per leg. In total, we measured 80 angles. We considered three approaches to obtain reference angles of flexion. In contrast to other studies, we decided not to use a goniometer as reference because measurements taken from goniometers are subjective to each observer [6]. Another approach we considered was placing point markers at the ankle, knee and hips to reconstruct the actual positions of the bones. However, we discarded this approach because of the difficulty to locate specific bones accurately on some subjects due to muscle and fat. We decided to align straight markers along the subject's upper and lower leg and calculated the angle between the markers digitally on a 2D photograph. The angles of flexion calculated by KneeHapp correlated to the reference angles with an average error of $4.82°(\pm\ 3.92°)$. These results suggest KneeHapp provides more accurate measurements than goniometers, which have been shown to have an average intra-observer variability of $9.6°$[6].

2) One-leg Hop: We asked subjects to perform a total of six hops (a short hop, a middle hop and a long hop, once on each leg). We collected a total of 60 hops. We placed a camera at ground level and recorded jumps at 240 frames per second. We determined the duration of each hop by counting the amount of frames elapsed between jumping and landing in the video recordings. We considered as a jumping frame the first frame in which the subject's foot was not in contact with the ground anymore and landing frame as the first frame where the foot made contact with the ground again. We measured the absolute difference between the calculated duration and the reference duration and normalized the difference by the reference duration. KneeHapp detected jump durations with an average accuracy of 77.10% ($\pm$ 18.57%).

3) Side Hops: We asked subjects to perform as many side hops as they could in a period of 10 seconds. In order to capture the differences in strength and motoric skills between individuals' primary and secondary legs, subjects performed side hops on both legs. We collected a total of 395 hops. In order to obtain reference values, we counted and video-recorded the hops performed by the subjects. KneeHapp computed the amount of side-hops performed by subjects with an average accuracy of 96.8% ($\pm$ 22%).

We evaluated the squat exercise qualitatively together with the usability of KneeHapp in a series of interviews with patients of an ACL injury. We improved KneeHapp with the insights gained during these interviews and handed in a questionnaire at the end, which is shown in Table I.

The results presented in this section meet the accuracy requirements of a system to support the rehabilitation after an ACL injury, as they are meant to aid orthopedists to make treatment decisions. In particular, a performance score for each leg could be computed based on several repetitions of the assessment exercises. The improvement of this score over time together with the score attained when performing the exercises with the non-injured leg could be provided to orthopedists to support them at making treatment decisions.

VI. Conclusion

We presented KneeHapp Textile, a smart textile system that supports different rehabilitation exercises performed throughout the rehabilitation of an ACL injury. The results of our controlled experiment suggest that KneeHapp Textile computes the different exercise performance metrics reliably. Therefore, KneeHapp can be used by patients to obtain live feedback about the execution of the exercises (e.g., shaking and deviation during a squat) and by orthopedists to make better-informed treatment decisions.

References

[1] Swamy Ananthanarayan, Miranda Sheh, Alice Chien, Halley Profita, and Katie Siek. Pt Viz: Towards a Wearable Device for Visualizing Knee Rehabilitation Exercises. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1247–1250, New York, NY, USA, 2013. ACM.

[2] Mobolaji Ayoade and Lynne Baillie. A Novel Knee Rehabilitation System for the Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2521–2530, New York, NY, USA, 2014. ACM.

[3] Ceara Ann Byrne, Claudia B Rebola, and Clint Zeagler. Design Research Methods to Understand User Needs for an Etextile Knee Sleeve. In *Proceedings of the 31st ACM International Conference on Design of Communication*, SIGDOC '13, pages 17–22, New York, NY, USA, 2013. ACM.

[4] Guido Gioberto, Cheol-Hong Min, Crystal Compton, and Lucy E Dunne. Lower-limb Goniometry Using Stitched Sensors: Effects of Manufacturing and Wear Variables. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, ISWC '14, pages 131–132, New York, NY, USA, 2014. ACM.

[5] Juan Haladjian, Zardosht Hodaie, Han Xu, Mertcan Yigin, Bernd Bruegge, Markus Fink, and Juergen Hoeher. KneeHapp: A Bandage for Rehabilitation of Knee Injuries. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 181–184. ACM, 2015.

[6] Matthew Ockendon and Robin Gilbert. Validation of a Novel Smartphone Accelerometer-Based Knee Goniometer. *Journal of Knee Surgery*, 25(04):341–346, may 2012.

6.8 iPig: Towards Tracking the Behavior of Free-roaming Pigs

This publication presents an application to track the activities of free-roaming pigs in rural regions in Africa. The motivation for this work is to provide veterinarians a better insight into the behaviors of pigs that cause them to get infected with parasites (e.g. Taenia solium) that can be transmitted to humans through the ingestion of uncooked meat. This paper presents an activity recognition algorithm, a system architecture to enable veterinarians to track their pigs' locations live and discusses an approach to track pig positions using a small sized device with low energy consumption rates.

The author of this Habilitation elicited the requirements for an activity tracker for pigs in collaboration with veterinarians and led a team of students who developed the user interface. Furthermore, he collected data from a pig in a stable in Munich, studied the performance of an activity recognition system and wrote the paper.

Authors	Haladjian, J., Ermis, A., Hodaie, Z., & Bruegge, B.
Conference	Fifth International Conference on Animal-Computer Interaction
Number of Pages	5
Type	Short Paper
Review	Peer Reviewed (3 Reviewers)
Year	2017
DOI	https://doi.org/10.1145/3152130.3152145

iPig: Towards Tracking the Behavior of Free-roaming Pigs

Juan Haladjian
Technical University Munich
Munich, Germany
haladjia@in.tum.de

Ayca Ermis
Georgia Institute of Technology
Georgia, USA
aycaermis@gatech.edu

Zardosht Hodaie
Technical University Munich
Munich, Germany
hodaie@in.tum.de

Bernd Brügge
Technical University Munich
Munich, Germany
bruegge@in.tum.de

ABSTRACT

Many farmers and families in poor rural areas in the developing world keep pigs as a resource for income. Most of these pigs are not kept in stables but let to roam freely. While this enables poor farmers to keep livestock without vast investments and sets pigs free from stables, it also increases the transmission rate of infectious diseases among pigs and to humans. Currently, there is a lack of knowledge on free-roaming pigs' behavior. In particular, veterinarians are interested in correlations between pig behaviors and the presence of an infectious disease. In this paper, we present the iPig system, a wearable motion sensor to track the physical activity of pigs and a user interface to enable veterinarians to keep track of the activities of the pigs in a herd. The motion sensor inserted inside a pig's ear classifies its physical activities into 'walking', 'eating' and 'resting'. A daily report about pig activities is displayed to veterinarians over a user interface on a tablet device. Results of a first pilot study suggest that iPig could classify pig physical activities with an accuracy of up to 95.8%. We also discuss the rationale behind the wearable for pigs we designed following an animal-centered design methodology.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation: Miscellaneous

Author Keywords
iPig, wearable. activity recognition, animal, pig

INTRODUCTION

The increasing demand for meat has made pork grow in popularity, mainly because pigs have low associated costs to acquire and raise [11]. In particular, pigs grow faster and have more offspring than sheep and cattle, eat leftover food, and are easy to sell [6]. As a consequence, several farmers and families in developing countries such as Zambia, Kenya, Uganda and Tanzania have started keeping pigs as a resource for savings and income [6].

One of the reasons why pigs are a cost-effective option for farmers is their natural ability to scavenge for food. In some countries in the developing world, pigs are not kept in stables but are allowed to roam freely. This has enabled poor farmers to maintain livestock without vast investments (e.g. infrastructure, foodstuff). Despite its economical benefits, allowing pigs to roam freely increases the rate of transmissions of infectious diseases among pigs and to humans [11].

One infectious disease of particular relevance in the case of free-roaming pigs is Cysticercosis. Cysticercosis is a major cause of epilepsy in the developing world. The disease is caused by a parasite called *Taenia Solium*. The parasite is transferred in a cyclic fashion from pigs to humans through undercooked or raw meat and from humans to pigs through human feces. In rural areas where there is low provision of latrines, people often defecate openly, offering a possibility for pigs to ingest human feces with infective parasite eggs. Cysticercosis is suspected to cause epileptic strokes and death to pigs, as it does to humans.

As a measure against the propagation of infectious diseases in poor African countries, this research aims at gaining understanding into the daily behavior of free-roaming pigs. A previous study used GPS technology to track the movements of free-roaming pigs in western Kenya to gain knowledge about areas visited, distances traveled and amounts of time pigs spent outside their 'home range' [11]. This study found that pigs travel an average distance of more than 4 km in a 12-hour period and spend an average 47% of their time outside their homestead of origin. In addition to tracking the positions visited by pigs daily, the physical activities of pigs could provide relevant information to understand where and how pigs might get infected and even predict the presence of an infection.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACI2017, November 21–23, 2017, Milton Keynes, United Kingdom

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-5364-9/17/11…$15.00

DOI: https://doi.org/10.1145/3152130.3152145

In this paper, we report on the iPig system, an on-going effort to gain knowledge about free-roaming pig behavior. In particular, we have developed a wearable device able to classify free-roaming pig physical activity and a user interface that provides aggregated statistical information about pig's daily behavior to veterinarians. The contribution of this paper is threefold. First, we describe the requirements we elicited for a wearable device for free-roaming pigs. Second, we address the design trade-offs we faced during its development. The requirements we elicited and design decisions we made might prove useful in the development of wearable devices for pigs or other animals. Third, we present the first insights of a study we conducted in order to determine the performance of different supervised machine learning algorithms at classifying pig physical activity.

REQUIREMENTS

In order to elicit the requirements for designing a wearable device to keep track of free-roaming pigs, we conducted a series of interviews with two veterinarians that work with pigs in a daily basis. The first veterinarian performs research in the field of infectious diseases caused by free-roaming pigs in poor regions in Africa. The second one has over ten years experience treating and working with pigs in a stable near Munich, Germany. Based on the interviews, we identified the following requirements:

RQ1 - Safety and welfare. Free-roaming pigs have been reported to walk distances of more than 4 km daily [11]. The design of a wearable device should not limit the pig's ability to walk and scavenge for food or increase its energy expenditure. Furthermore, pigs have a sensitive skin and less hair than most mammals. The wearable device or strap bands should be designed not to cause any injury to pigs' skin.

RQ2 - Robustness. Pigs roll on mud and dirt and are exposed to rain. Furthermore, pigs tend to bite themselves and might try to remove the device as long as they can reach to it with their mouth. Therefore, the wearable device should be water proof, resistant to bites and ideally, be placed at parts of the pig's body where the pig cannot reach it with its mouth.

RQ3 - Low Cost. Free-roaming pigs as well as wearable devices are prone to getting stolen in countries under development. In order to reduce the chances that the device gets stolen and the losses in case it does, the wearable device should have a low cost.

RQ4 - Energy efficiency. Neither farmers or veterinarians might be willing to invest time in removing the wearable devices from every pig in a herd in order to replace or recharge the battery regularly. Therefore, it is critical that the wearable device functions as long as possible without human intervention.

RQ5 - Adaptability. Rapid growth of an animal might cause a strapped wearable device not to fit anymore, cause discomfort and even endanger the animal (e.g. by strangling it when strapped around its neck [3]). Pigs grow on average from 12 kg to 30 kg between their second and fifth month of life [6]. The wearable device should be designed to fit and adapt to the different sizes of a pig.

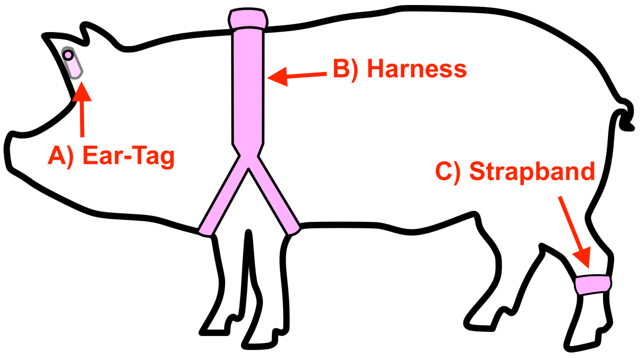


Figure 1. Different designs for a wearable device to be used by pigs.

ANIMAL-CENTERED DESIGN

We designed the wearable device to based on the requirements listed in the previous section and the ethical principles proposed by Mancini [10] with the goal to maximize the welfare of the pigs using the device. We designed the wearable device iteratively by conducting field studies to test our prototypes on different pigs in a stable located in the outskirts of Munich, Germany.

We considered three possible designs for the wearable device: A) an ear-tag, B) a harness fastened around the pig's torso and C) a strap band for the leg. These alternatives are illustrated in Figure 1. We discarded limbs as an alternative to attach a device quickly during the first test, as we observed clear signs of discomfort and lack of acceptance from pigs, which tried to remove the device by biting it. We concluded that pigs would not consent to attach the device to any of their limbs, hence this design would violate the second ethical principle proposed by Mancini: *Garnering participants' mediated and contingent consent* [10]. More importantly, veterinarian suggested that attaching a device to any limb of a pig could cause discomfort and injury to the pig and be easily removed by it.

Alternative B) consisted of a leather harness strapped around the pig's torso. This design had the advantage that it allowed for a bigger device size, hence a larger battery could be placed on it that could last for months. However, during a field test, we observed two major limitations to this design. First, it caused injuries to the pig's skin due to rubbing between the strap band and the pig's skin while walking with the harness attached for a week. Second, the harness would have to be adapted or replaced after a few weeks due to pigs' growth (RQ5). Attaching a wearable device that causes discomfort and injury to an animal would not be acceptable from an animal-centered ethical point of view. Therefore, we decided against this design.

We decided to place the wearable device inside a pig's ear. The device is powered with a knob containing a coin-cell battery that is attached from the outer side of the ear. This design has three main advantages. First, the ear of a pig does not grow in size as much as other parts of its body. As a consequence, a device might remain inserted in a pig's ear for longer periods

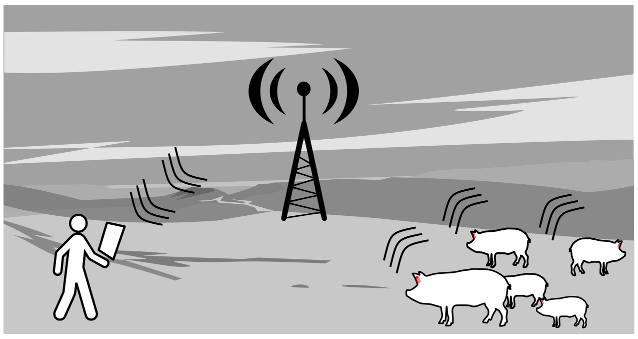


Figure 2. iPig's architecture.

of time (RQ5). Second, the ear cannot be reached by the pig's mouth and is relatively protected from mud and dirt, thus reducing potential damage inflicted upon the device (RQ2). Additionally, the fact that ear tags for pig identification are in use in some commercial farms suggest that the ear has been found to be a convenient place for attachment of electronics by the veterinary medicine community. This also means the practice of punching a wearable device through the ear has been approved by the ethical committees regulating the animal practices in those farms. Attaching a wearable device that causes pain to an animal raises the ethical question as to what extent do its benefits justify the pain and harm involved during the attachment. We argue that our research might improve the life quality of pigs and humans equally by preventing potentially more harmful diseases. An image of a 3D printed casing of the wearable device is shown in Figure 4 (NanoHub).

IPIG

iPig consists of two main components: a wearable device worn by pigs and a user interface on a tablet. The wearable device classifies pig activities based on motion data and sends a daily activity report to the tablet device. The tablet device aggregates the daily activity reports received from different wearable devices and displays statistical information about pig activities and areas where the activities took place. The user interface is shown in Figure 3 and the iPig's architecture is illustrated in Figure 2. In this section, we describe in detail the design of the wearable device and user interface.

Wearable Device

In order to address the strict constrains in terms of size and weight posed by the placement of the device inside a pig's ears, we decided to design a custom motion sensor. We call our wearable device *NanoHub*. The *NanoHub* and contains a 6-axis Inertial Measurement Unit (IMU), an ARM Cortex M0 microcontroller and a Bluetooth Low Energy (BLE) module. In addition, we designed a larger device called *MicroHub*. The *MicroHub* contains an SD card reader and additional electronics to facilitate the software development and testing. We used this device to collect raw motion data from pigs in order to develop our machine learning algorithm. Both devices were designed by Figure 4 shows a comparison of the *MicroHub* and *NanoHub*. Both devices were developed by a company in Munich called InteractiveWear[1].

[1]http://www.interactive-wear.de/

Activity Recognition

One key aspect of the iPig system is knowing what physical activity pigs are doing. Activity recognition based on IMU data has been intensively studied in humans [2, 9]. A few studies have been done on quadrupeds. Cornou et al. have proposed a method based on multi-process Kalman filter to classify pig activities based on acceleration patterns [4]. Ladha et al. [8] developed a wearable device able to detect physical activities linked to the wellbeing of dogs. In cows, several studies investigated physical activity recognition in order to detect lameness [1, 7]. Most of these approaches have used supervised machine learning to classify physical activities.

In another study, Cornou et al. [5] used sensor networks and motion sensors strapped to pig's neck for detecting oestrus based on pigs' acceleration. Furthermore, Thompson et al [12] used IMUs for classifying animals posture (standing, sitting, lateral and sternal lying). None of these studies have focused on pigs roaming in the wild.

Our approach for classifying pig activities consists of three steps:

1. *Preprocessing*. We divide IMU data (linear acceleration and rotation) in windows of 50 samples. We low-pass filter every window using a second order Butterworth low-pass filter at 20 Hz. After applying the filter, we compute the magnitude of each linear acceleration vector. This produces a data set with 7 data points for each time step: linear acceleration and rotation along three axes x,y,z and the magnitude of the linear acceleration. This gives us a matrix of 7x50 values for each window.

2. *Feature Extraction*. For each 7x50 values-window, we compute 5 features: mean, median, standard deviation, Zero Crossing Rate (ZCR) and Peak-to-Peak amplitude (P2P). We selected this set of features based on the fact that they can be implemented on a wearable device at a low computational cost. The feature extraction results in a vector of 35 features which we call "feature vector".

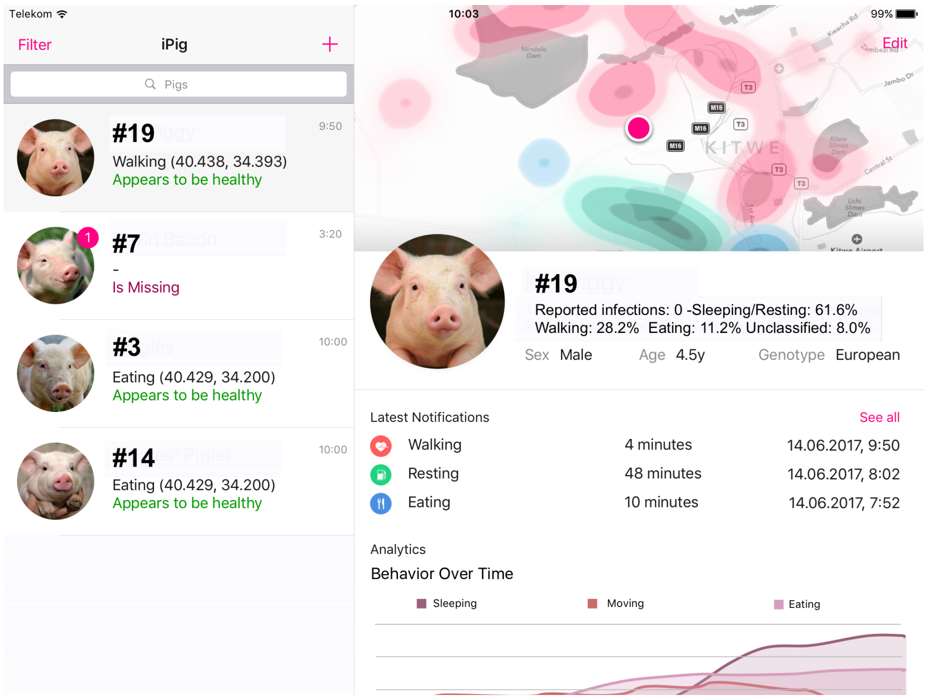
3. *Classification*. We classify the feature vector using a trained machine learning model into one of the following classes: "walking", "eating" or "resting". In the next section, we discuss the classification performance of different machine learning models we studied.

PILOT STUDY

We conducted a study to gain insight into the accuracy of different supervised machine learning models at classifying physical pig activity. This pilot study was realized in a stable in the outskirts of Munich, Germany with the consent of the veterinarian who are legally responsible for the pigs to ensure that to harm was done to the pig during the study.

Setup

We selected a pig randomly from a stable with an outdoor space. The outdoor space contained wild vegetation that enabled pigs to wander and scavenge for food. We attached the sensor to an ear tag worn by the pig and let the pig walk around freely. When the pig stopped walking for too long, we walked


Telekom 10:03 99%

Filter iPig +

Pigs

#19	Walking (40.438, 34.393) Appears to be healthy	9:50
#7	- Is Missing	3:20
#3	Eating (40.429, 34.200) Appears to be healthy	10:00
#14	Eating (40.429, 34.200) Appears to be healthy	10:00

Edit

#19

Reported infections: 0 -Sleeping/Resting: 61.6%
Walking: 28.2% Eating: 11.2% Unclassified: 8.0%

Sex Male Age 4.5y Genotype European

Latest Notifications See all

Walking	4 minutes	14.06.2017, 9:50
Resting	48 minutes	14.06.2017, 8:02
Eating	10 minutes	14.06.2017, 7:52

Analytics

Behavior Over Time

Figure 3. iPig's user interface implemented on an iPad device. On the left side of the interface, a list of pigs is shown, together with it's location and health condition. This enables veterinarians to have an overview of the pigs' health condition and location. One or more pigs can be selected from the list. The right side of the interface displays information about the pig or pigs selected on the left side. The physical activities the selected group of pigs performed in a selectable time range are displayed overlaid on a map of the region at the locations where the activities took place. The activity color mapping goes as follows: walking - red; resting - green; eating - blue. In addition, statistical information about the behavior of the currently selected pigs is shown as a table and timeline.

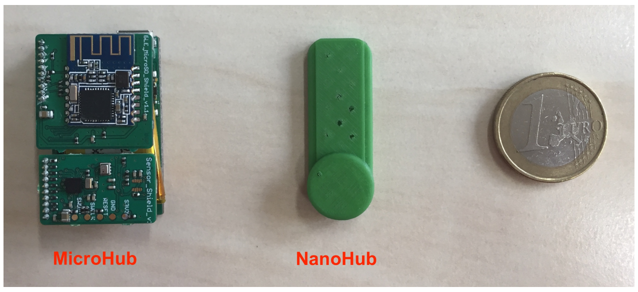


Figure 4. MicroHub (left) and prototype of the NanoHub (middle). Both devices contain a microcontroller, a 6-axis IMU, a BLE module and a battery. However, the MicroHub contains an additional SD card reader needed for development purposes.

behind it to ensure it continued walking and scavenging for food. Figure 5 shows a member of the team walking behind a pig.

We recorded motion data at 100 Hz and videotaped the entire experiment. We then labeled the motion data into periods as: 'walking', 'eating', 'resting' based on the video. We recorded a total of 282s (walking), 8.5s (eating) and 29s (resting). This gave us the following amounts of feature vectors to classify: 1130 (walking), 35 (eating), 118 (resting).

Results

We evaluated the performance of different supervised machine learning models, including Support Vector Machines (SVM), k-Nearest Neighbors (kNN) and linear discriminant. We tried two types of SVM kernels: a cubic and a quadratic kernel and studied the performance of the kNN model with k=1 and k=10 neighbors. Table 1 shows the accuracy, precision and recall of the different machine learning models we studied. The results were validated using the 10-fold cross validation technique. We selected these algorithms based on their public availability.

Figure 5. Image taken during the pilot study with a pig.

Discussion

These results suggest that the wearable device we designed would classify pig activities into walking, eating or resting with an accuracy of 95.8% and a relatively low misprediction rate (precision: 75,4% and recall: 86,6%).

While these results provide first insights into how accurately pig physical activities could be classified using a motion sensor

Table 1. Performance of different supervised machine learning models at classifying pig physical activity.

Model	Accuracy	Precision	Recall
SVM (cubic Kernel)	95,87%	75,42%	86,63%
SVM (quadratic Kernel)	95,79%	71,87%	88,46%
KNN (k=1)	94,54%	80,04%	81,16%
KNN (k=10)	95,17%	65,26%	88,16%
Linear Discriminant	94,34%	76,71%	77,14%

attached inside the pig's ear, our experiment has two main limitations. First, the data used to train the model and to validate the model was acquired with the same pig. This could have biased the model to perform well on the pig that participated on the experiment but might perform less accurately when tested with motion data from other pigs. Second, we acquired considerably more data for the "walking" activity. This would cause our model to achieve a high accuracy by favoring the "walking" prediction.

In the future, a larger data set should be acquired for longer periods of time and for several pigs (ideally of different races and sizes). This would produce a larger dataset that could be used to obtain a more generalisable machine learning model. Furthermore, the pigs in the experiment should be let free to perform their natural activities with no human influence in order not to bias their behavior.

CONCLUSION

We have presented an on-going work to enable veterinarians to keep track of free-roaming pigs. In particular, we presented a wearable device able to classify pig physical activities. The results of our pilot study suggest our approach is viable and would even enable the tracking of pig activities with an accuracy of 95.8%. However, a longer-term in-field deployment of the iPig system is necessary to determine the performance of our approach under a more unconstrained setting and would serve for identification of issues with respect to the requirements we elicited.

We assessed different designs for the wearable device and decided for a design meant to be inserted inside a pig's ear. The fact that pigs in many commercial farms already use ear tags opens up a possibility for our wearable device to be integrated in the ear at almost no added cost. Keeping track of pigs' physical activities could be useful to monitor and better address the needs of pigs in stables as well (e.g. predict disease and pregnancy).

ACKNOWLEDGMENTS

We would like to thank Veronika Schmidt and Dr. Lian Thomas for their supervision and guidance during this project and Dr. Stefan Nüske and the staff of the Ludwig Maximilian University Munich for their support during the pilot study.

REFERENCES

1. Maher Alsaaod, Christoph Römer, Jens Kleinmanns, Kathrin Hendriksen, Sandra Rose-Meierhöfer, Lutz Plümer, and Wolfgang Büscher. 2012. Electronic detection of lameness in dairy cows through measuring pedometric activity and lying behavior. *Applied Animal Behaviour Science* 142, 3 (2012), 134–141.
2. Akin Avci, Stephan Bosch, Mihai Marin-Perianu, Raluca Marin-Perianu, and Paul Havinga. 2010. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In *Architecture of computing systems (ARCS), 2010 23rd international conference on*. VDE, 1–10.
3. Ruth M Casper. 2009. Guidelines for the instrumentation of wild birds and mammals. *Animal behaviour* 78, 6 (2009), 1477–1483.
4. Cécile Cornou and Søren Lundbye-Christensen. 2008. Classifying sows' activity types from acceleration patterns: an application of the multi-process kalman filter. *Applied Animal Behaviour Science* 111, 3 (2008), 262–273.
5. Cécile Cornou, Jens Vinther, and Anders Ringgaard Kristensen. 2008. Automatic detection of oestrus and health disorders using data from electronic sow feeders. *Livestock Science* 118, 3 (2008), 262–271.
6. Cate E Dewey, Jared M Wohlgemut, Mike Levy, and Florence K Mutua. 2011. The impact of political crisis on smallholder pig farmers in western Kenya, 2006–2008. *The Journal of Modern African Studies* 49, 3 (2011), 455–473.
7. Juan Haladjian, Bernd Brügge, and Stefan Nüske. 2017. An approach for early lameness detection in dairy cattle. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*. ACM, 53–56.
8. Cassim Ladha, Nils Hammerla, Emma Hughes, Patrick Olivier, and Thomas Ploetz. 2013. Dog's life: wearable activity recognition for dogs. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 415–418.
9. Oscar D Lara and Miguel A Labrador. 2013. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials* 15, 3 (2013), 1192–1209.
10. Clara Mancini. 2017. Towards an animal-centred ethics for Animal–Computer Interaction. *International Journal of Human-Computer Studies* 98 (2017), 221–233.
11. Lian F Thomas, William A de Glanville, Elizabeth A Cook, and Eric M Fèvre. 2013. The spatial ecology of free-ranging domestic pigs (Sus scrofa) in western Kenya. *BMC veterinary research* 9, 1 (2013), 46.
12. Robin Thompson, Stephanie M Matheson, Thomas Plötz, Sandra A Edwards, and Ilias Kyriazakis. 2016. Porcine lie detectors: Automatic quantification of posture state and transitions in sows using inertial sensors. *Computers and electronics in agriculture* 127 (2016), 521–530.

6.9 Gait Anomaly Detection in Dairy Cattle

This publication describes an activity recognition algorithm to detect deviations in the usual gait of dairy cattle. The algorithm builds a model of the usual gait of a cow by segmenting its gait strides and using them to train an unsupervised machine learning algorithm. After the first hours of usage, our algorithm is able to detect deviations from the trained model. The paper describes our algorithm and discusses how we evaluated it.

The author of this Habilitation collected the data in a stable in Munich, studied the performance of different recognition methods and wrote the paper.

Authors	Haladjian, J., Hodaie, Z., Nueske, S., & Bruegge, B.
Conference	Fourth International Conference on Animal-Computer Interaction
Number of Pages	8
Type	Full Paper
Review	Peer Reviewed (3 Reviewers)
Year	2017
DOI	https://doi.org/10.1145/3152130.3152135

Gait Anomaly Detection in Dairy Cattle

Juan Haladjian
Technical University Munich
Munich, Germany
haladjia@in.tum.de

Zardosht Hodaie
Technical University Munich
Munich, Germany
hodaie@in.tum.de

Stefan Nüske
Ludwig Maximilian University
Munich, Germany
stefan.nueske@lmu.de

Bernd Brügge
Technical University Munich
Munich, Germany
bruegge@in.tum.de

ABSTRACT
Cow lameness is a common welfare issue in the dairy industry that causes severe health and life quality issues to cows, including pain and a reduction in their life expectancy. The earlier a lame cow is detected, the earlier and more effectively it can be treated. A change in the gait of the cow is the earliest symptom of lameness. Currently, lame cows are detected by visual inspection performed by herdsmen, which is subjective and time consuming. We present an approach to automatically detect anomalies in the walking pattern of a cow as a possible indicator of lameness. The detection is done by a wearable motion sensor attached to a cow's hind left leg. Our approach builds an individual model of the usual walking pattern of a cow during the first minutes of use and detects deviations from this model afterwards. Results from a controlled experiment we conducted indicate that our approach can detect deviations in cows' gait with an accuracy of 91.1%. This information can be used by veterinarians to keep track of changes in the walking pattern of cows and to decide whether to treat a specific cow.

ACM Classification Keywords
H.5.2 Information interfaces and presentation: Miscellaneous

Author Keywords
lameness, cow, wearable, animal, anomaly detection, gait analysis

INTRODUCTION
Lameness is a manifestation of painful disorders that result in an impaired movement or deviation from normal gait or posture [18]. In dairy cattle, the main causes of lameness are lesions in the claws which cause bacterial infections and swelling in cows' hooves and legs. Lameness causes severe pain and is associated with health issues such as the loss of fertility. Furthermore, lameness causes serious welfare and economic problems in the dairy industry. Some of the costs associated with lameness are the need for veterinary treatment and a reduction in milk production and cow's reproductive performance. At advanced stages of the disease, a lame cow might die or have to be sacrificed by humans. Lameness is a common issue in dairy cows, with some stables having up to 72% lame cows. [18].

The earlier a lame cow is identified, the earlier the causes of the disorder can be treated. Currently, lame cows are identified by visual inspection of their walking pattern, which is done by herdsmen. However, automation and the rapid growth in livestock production have led to more cattle and less employees per herd. As a consequence, herdsmen have less time to monitor the health condition of their cows.

Automated systems for cow milking, feeding and cleaning are already being used in commercial herds. Neckbands with integrated motion sensors are being used in commercial herds to predict whether a cow is undergoing oestrus (i.e. the period of sexual fertility in a female mammal). These neckbands are used by veterinarians to determine the proper time to inseminate a cow. In contrast, systems to detect lameness are rarely used in commercial herds, despite the variety of solutions proposed by the scientific community.

Approaches for automated lameness detection include those based on computer vision and pressure sensing. These approaches require expensive equipment and are limited to measuring a few steps per cow, which might not be enough to accurately detect lameness. Motion sensors have also been studied to detect lameness. Most of these approaches keep track of a cow's physical activity (lying down, standing and walking) [11, 10]. However, changes in physical activity due to lameness occur at more advanced stages of the disorder. The first observable symptom of lameness is a change in a cow's usual walking pattern (i.e. gait).

We present an approach to detect deviations in cows' usual walking pattern using a wearable motion sensor. Assuming the cow is healthy and walks normally at the time the sensor is attached to it, our approach creates a model of the usual walk-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ACI'17, November 21–23, 2017, Milton Keynes, United Kingdom
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5364-9/17/11...$15.00
DOI: https://doi.org/10.1145/3152130.3152135

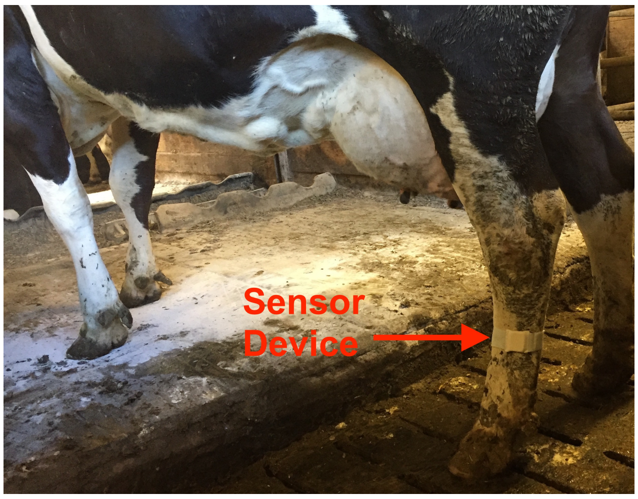


Figure 1. Motion sensor attached to a cow's hind left leg.

ing pattern for each individual cow during the first minutes of use and detects deviations from that pattern later on. The cows that have been detected as having a deviation in their usual walking pattern can be investigated by veterinarians later on. Our approach requires a single motion sensor attached to cows' hind left leg. The motion sensor is shown in Figure 1.

The rest of the paper is structured as follows. The *Related Work* Section provides a comprehensive overview of other automated lameness detection systems and highlights how our approach relates to them. In the *Approach* Section, we describe the hardware we designed, the signal acquired by our device while a cow is in motion and how our algorithm works. In particular, we present a wearable motion sensor and a set of algorithmic steps to train the model and classify cow steps into *normal* or *abnormal*. The *Controlled Experiment* Section presents the results of a controlled experiment we conducted in order to validate our approach.

RELATED WORK

Most modern stables collect data from cows' daily activity such as the amount of milk cows yield and how much food they are fed. Different studies have suggested using this data to predict lameness [7, 5]. However, changes in milk yield and feeding behavior due to lameness might manifest days after changes in gait. Detecting a lame cow based on its gait would make it possible to stop further development of the disorder. This would allow veterinarians to treat the cause of the disorder earlier, relieving it from pain and restoring its normal function.

Approaches for lameness detection based on gait analysis include those based on computer vision, weight sensing and motion sensing. Computer vision approaches extract lameness-related parameters from video, such as the arching of a cow's back [15], the amount of overlapping between a cow's consecutive steps [16] and the angle at which a cow's fetlock joint makes contact with the ground during a step [14].

Weight sensing approaches measure the weight a cow places on each limb while standing on force plates [12] or walking over a force-sensitive mattress [8]. Based on this data, information about cows' walking and standing behavior is calculated, such as the length and duration of a stride [8], the amount of kicks a cow performs while standing [12] and the weight distribution under single hooves [6].

Computer vision and weight sensing approaches require expensive devices and are limited to measuring a few steps per cow. These approaches face additional challenges such as the fact that cows near the measuring area might disrupt the measurements [18] and the need for additional technologies to identify the cow being measured.

Motion-based lameness detection approaches rely on motion sensors that are attached to cows' legs and/or neck. Most motion-based lameness detection approaches measure parameters related to cows' daily physical activity, such as the amount of time cows spend lying, standing and walking [17, 2], the number of steps cows perform per day [10] and the time of the days when cows start and stop walking [19]. These approaches do not analyze gait per se, but predict lameness based on cows' daily activity.

Two studies have investigated cow gait for lameness detection based on motion data. Pastell et al. [13] applied wavelet analysis to accelerometer data and found that there is less symmetry in the acceleration of hind legs in lame cows than in healthy cows. Chapinal et al. [3] found that the variance of acceleration of front and hind legs could be used to predict gait scores. These studies compared lame cows with non-lame cows in order to discover differences in their gait.

The approaches mentioned so far are based on the models that do not consider the differences in the physical behavior and tolerance to pain of each individual cow. However, Alsaaod et al. [2] found that the variation of physical activity among cows is significantly larger than the variation of physical activity caused by lameness. This suggests that lameness should be regarded as a per-cow basis rather than comparing a cow's motion to a baseline established from other cows.

Our approach compares the gait of a cow to a baseline established by the cow itself during the first minutes of use. The approach is based in anomaly detection, a technique commonly used to detect bank fraud and intrusion in computer networks. We chose to detect anomalies in gait because of two main reasons. First, a deviation from the normal gait is the first indicator of a possible lameness. Second, this approach takes into consideration the uniqueness of each cow's gait [2].

APPROACH

In this section, we describe our approach for anomaly detection in a cow's gait, including the sensor device that senses motion data and the anomaly detection approach. Our anomaly detection approach consists of two phases: the training phase and the detection phase. The training phase builds a model of the usual gait of a single cow. The detection phase classifies the gait of a cow into *normal* or *abnormal* based on a comparison of its current gait with a model created based on the gait of that particular cow during the first minutes of use of our wearable device. The training phase is done in four steps:

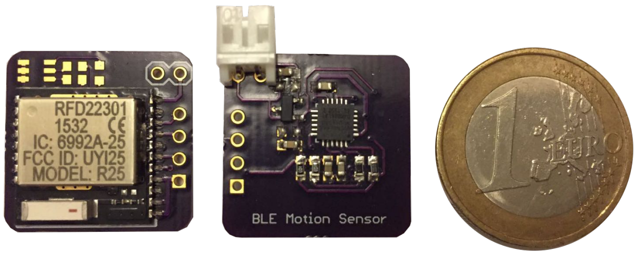


Figure 2. The front (left) and back (right) side of the sensor device.

data preprocessing, step segmentation, feature extraction and model training.

Sensor Device

We designed a sensor device containing an ARM Cortex-M0 microcontroller, a 6-axis Inertial Measurement Unit (IMU) and a Bluetooth Low Energy (BLE) module. We aimed for a low cost, low energy consumption and small footprint design. The ARM Cortex-M0 microcontroller operates at 16 MHz and is characterized by its low-power consumption rate and small footprint. The MPU-6050 from Invensense measures acceleration and rotation and performs sensor fusion directly on the chip. This allows for an energy-efficient and accurate computation of the device's orientation. The orientation of the hoof is relevant for lameness detection because lame cows often step at a particular angle in order to avoid pain [14]. As a communication module, we decided on BLE due to it's low-power consumption feature. We designed the circuit as a two-layer board placing the motion sensor on the front side and the microcontroller and BLE module on the back side. The dimensions of the circuit board are 21 x 21 x 2.5 mm. Figure 2 shows the front and back sides of the circuit board. The device functions at 3.3 v and is powered by a 400 mAh battery.

Preprocessing

The sensor device measures linear acceleration and rotation at 100 Hz along 3 axes (x,y,z). The device is oriented such that the y-axis represents vertical accelerations, the x-axis is parallel to the cow (i.e. in its walking direction) and the z-axis is lateral (i.e. left and right) to a cow. Figure 3 illustrates the correlation between walking strides and accelerometer signals along x- and y-axes.

The preprocessing step divides the incoming acceleration and rotation signals into chunks (windows) - units that are further processed and classified as 'normal' or 'abnormal' in the following steps. The choice of the window size has an impact on the accuracy of the anomaly detection as well as on the hardware requirements. The larger the size of a window, the more information that can potentially be extracted from it. However, the need to process larger windows of data leads to larger memory footprints, more computations and higher energy consumption rates.

We chose a chunk size of 3000 samples (30 seconds) by following a greedy heuristic approach using the average accuracy of our anomaly detection approach as metric for optimization. This parameter can be decreased to lower the requirements to

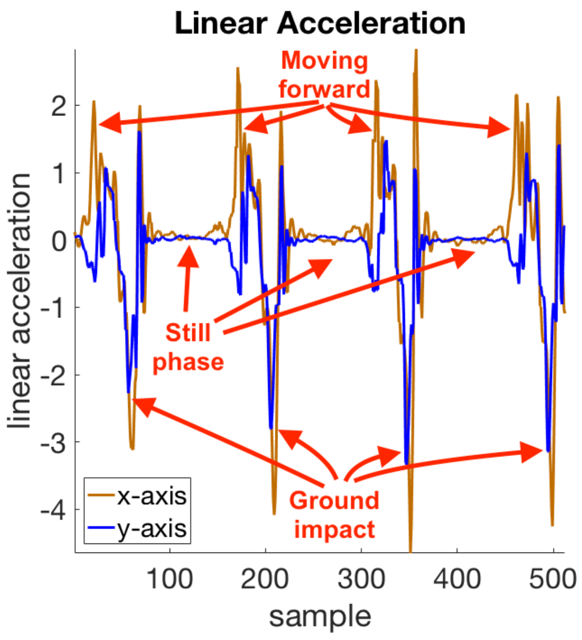


Figure 3. Linear acceleration along x- and y-axes during four strides. Forward movements during a stride cause a positive acceleration along the x-axis. Impacts with the ground can be seen as peaks in the acceleration along the y-axis. Furthermore, the periods while the hoof is in contact with the ground (still phases) have almost zero acceleration.

the hardware device or increased for higher accuracy of the anomaly detection.

In order to remove noise caused by the accelerometer, we apply a second order Butterworth low-pass filter at 20 Hz to each window. Low-pass filters eliminate high frequencies in a signal making it smoother. After applying the filter, we compute the magnitude of each linear acceleration vector. This produces a total of 7 data sets: linear acceleration and rotation along three axes and magnitude of linear acceleration.

Step Segmentation

The purpose of the step segmentation is to detect the beginning and ending of a step. We segment the steps based on the peaks produced during the stride. We chose to do the step segmentation based on the linear acceleration along the y-axis because the highest peaks in the signal during a cow's stride are caused by the impact of its hoof with the ground. The step segmentation is done in the following three steps:

1. Every step has two upper peaks. We detect the highest peak with a peak detection algorithm. We ignore peaks that are less than 60 samples away from a previously detected peak. This also filters out periods when cows did not walk.

2. Every step is preceded by periods of small variance in acceleration. We find these periods by searching for the 9-sample window with smallest variance in acceleration among the 70 samples before and after the detected peak. We call the center of these windows *initial step segments*.

3. Between two initial step segments, additional samples are included that might not belong to a step. Therefore, we trim

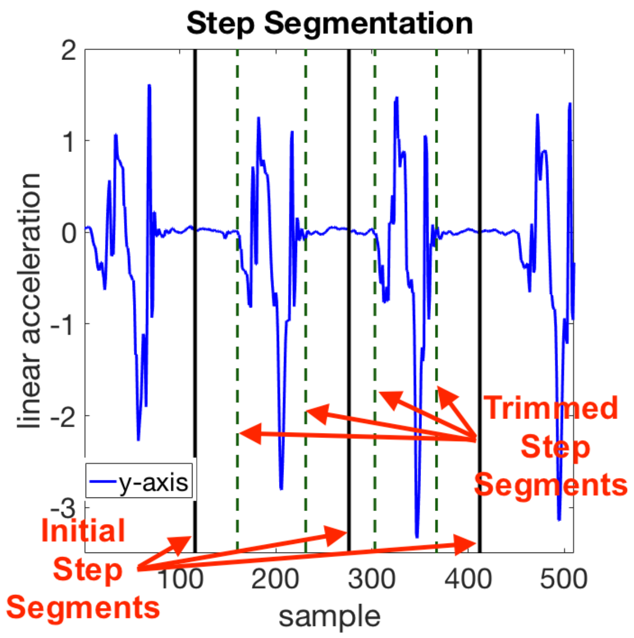


Figure 4. Initial and trimmed segments detected with our step segmentation algorithm applied to linear acceleration along the y-axis.

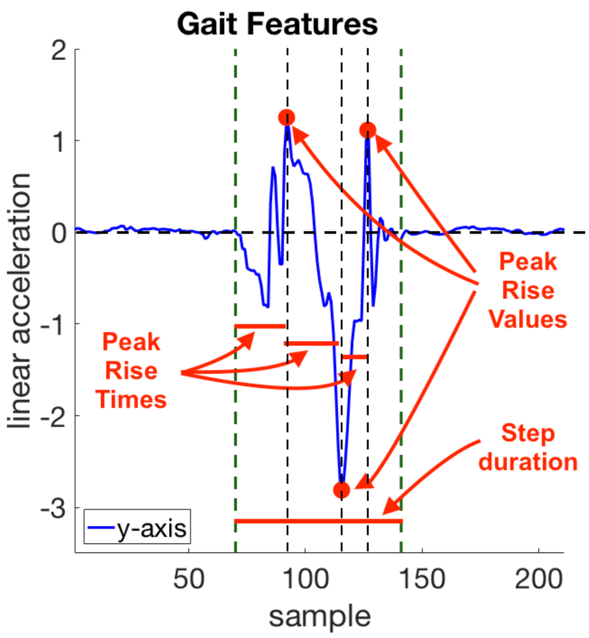


Figure 5. Gait features: peak rise values, peak rise times and step duration. These seven features are computed for all three accelerometer axes. This figure illustrates the computation of these features on the linear acceleration along the y-axis.

the step by shifting the initial step segments towards the peak detected in step 1. The initial step segments are shifted until the standard deviation of a 6-sample window centered at the shifted step segment is larger than 0.2.

Figure 4 shows the linear acceleration along the y-axis of four consecutive steps with annotations pointing at initial and trimmed step segments.

Feature Extraction

For each step segmented, we compute a set of gait and statistical features. Gait features are measurements specific of a step. Every step is characterized by three peaks: two upper peaks and one lower peak. We first detect all three peaks. If any of the peaks could not be found, we ignore the step. This might happen if the cow shortly lifted a leg or got bumped by another cow. For all three peaks, we compute its rise value and time. The rise times are computed as the difference in samples to the previous peak. The rise time of the first peak is computed as the difference in samples to the first sample in the trimmed step segment. In addition, the total duration of the step is added to the feature set. Figure 5 illustrates how the gait features are computed based on the three peaks of a single step.

Statistical features are measures to extract information from data sets. We extract the following statistical features: mean, median, standard deviation, Zero Crossing Rate (ZCR), Peak-to-Peak amplitude (P2P), Root Mean Square (RMS) and Average Acceleration Variation (AAV) for every step. ZCR is a measure of the amount of times a signal crosses the zero value. A high ZCR might indicate a highly intense or periodic activity. P2P is the difference between the maximum and minimum acceleration value in a step and provides information about the intensity of a step. RMS is the square root of the mean of the values in a step squared. This measurement provides information about the amount of acceleration and variation in a step. AAV is calculated as the sum of the absolute differences between consecutive samples in a step normalized by the number of samples. AAV provides an indication of how sudden changes in acceleration happen within a step. These measurements are commonly used for activity recognition applications and have been recently used for fall-detection and gait analysis in humans [4, 1].

The list of gait and statistical features are enumerated in table 1. Gait features are computed on linear acceleration and statistical features are computed on linear acceleration, rotation and magnitude of acceleration. ZCR is only computed on the linear acceleration. This gives us a total of 21 gait features and 45 statistical features per step. A window might contain several steps. We average the features extracted from the same window. A vector containing the 66 averaged features is called *gait observation*.

Model Training

Our anomaly detection approach is based on a one-class Support Vector Machine (SVM) classifier. Binary SVM classifiers calculate a boundary that maximizes the distance between observations (i.e. samples) of two different classes. Our one-class SVM classifier finds a boundary around observations of the *normal* class and classifies new observations based on their distance to this boundary. The classifier is trained automatically based on the gait of a cow acquired during the first minutes of use.

Table 1. Gait and statistical features used by our approach. Features labeled as *accel* are computed on all three axes of the linear acceleration and features labeled as *all* are computed on every axis of the linear acceleration, rotation and on the magnitude vector of the linear acceleration.

	Feature	Signal	#
Gait features	rise values	accel	9
	rise times	accel	9
	step duration	accel	3
Statistical features	mean	all	7
	median	all	7
	STD	all	7
	ZCR	accel	3
	P2P	all	7
	RMS	all	7
	AAV	all	7
Total			66

The boundary of our classifier is defined such that a fraction *outlier fraction* of the observations in the training set is classified as *abnormal*. The *outlier fraction* is used to define how 'compact' the boundary around *normal* gait observations is. The smaller the *outlier fraction*, the higher the specificity of the model (i.e. the less false alarms the model produces). On the other hand, a small *outlier fraction* reduces the sensitivity of the model (i.e. more prone to missing *abnormal* gait observations).

Anomaly Detection

New gait observations are classified as *normal* or *abnormal* according to their distance to the boundary of the *normal* observation set. Gait observations are classified as *abnormal* if and only if their distance to the boundary of *normal* gait observations is smaller than the *abnormality threshold*. The performance of the classifier is determined by the *outlier fraction* and *abnormality threshold* parameters.

CONTROLLED EXPERIMENT

This section describes a controlled experiment we designed following an animal-centered research methodology and applying ACI ethical guidelines.

Animal-centered Design

Previous work in automatic lameness recognition studied the gait of lame cows by forcing cows with different degrees of lameness to walk while their motion was being recorded. However, forcing an animal in pain to walk would be unacceptable for an animal-centered research approach and would violate several ethical principles accepted in ACI [9].

We have developed an approach to detect deviations from usual gait in cows. One way to evaluate our approach would be to attach our sensor device to several cows and wait until their gait changes (e.g. due to lameness or pregnancy). However, this could take several months, which makes this approach impractical for research purposes.

Another alternative we considered was to let cows walk under normal conditions and have experts determine whether any of the steps or sequences of steps were different from its usual gait. However, this study would require correlating every step of a cow in a video recording to the motion signal and would be prone to the subjective definition of 'abnormal' step.

Instead, our evaluation method is inspired by other research on anomaly detection, in which the detection of seldom occurring events were evaluated by simulating those events [4, 1]. For example, Cola et al. [4] evaluated their approach to detect anomalies in human gait by letting subjects walk with strap bands around their knee.

In order to cause a change in the walking pattern of cows, we decided to attach a plastic block to one of their hind hooves. Veterinarians usually attach such plastic blocks to cow hooves in order to relieve pain and allow an injured claw to heal. Attaching a plastic block to a hind hoof causes a similar change in walking pattern as lameness: in both cases, cows shift their weight towards the opposite hind limb. We attached blocks to the hind hooves because hind limbs are the ones most commonly affected by lameness. Figure 6 shows a plastic block attached to the outer claw of a left hind hoof.

Our procedure consisted of the following steps:

1. We let cows walk normally during a period of less than 10 minutes. We repeated this procedure on three different days in order to reduce fatigue and stress caused to the animal subject.

2. We attached a plastic block to one hind-hoove and let the cow walk with it.

3. We attached a plastic block to the opposite hoove and let the animal walk with it.

We consider this approach to be the most appropriate for an animal-centered research among the other approaches we considered because it does not involve pain and requires a considerable shorter intervention to cows' daily activity. The intervention to cows' daily activities lasted approximately 40 minutes: 10 minutes for the attachment of the sensor and recording of normal walking (this was repeated on three different occasions) and 30 minutes for the attachment of the plastic blocks.

Ethical Considerations

Our research required cows to take part in an experiment. In order to ensure an ethically appropriate involvement of the cows in our experiment, we designed our experiment based on the ethical guidelines proposed by Mancini [9]. We addressed these principles as follows:

1. *Respecting and caring for every participant without discrimination.* The participants of this experiment were cows of different ages and breeds. We did not harm any of the them or make any discrimination as for the selection of participants or treatment they received during the experiment.

2. *Garnering participants mediated and contingent consent.* We conducted this experiment together with two professional veterinaries who are the legal representatives of the cows that participated in the experiment. Both veterinaries know the needs and welfare requirements of these cows and

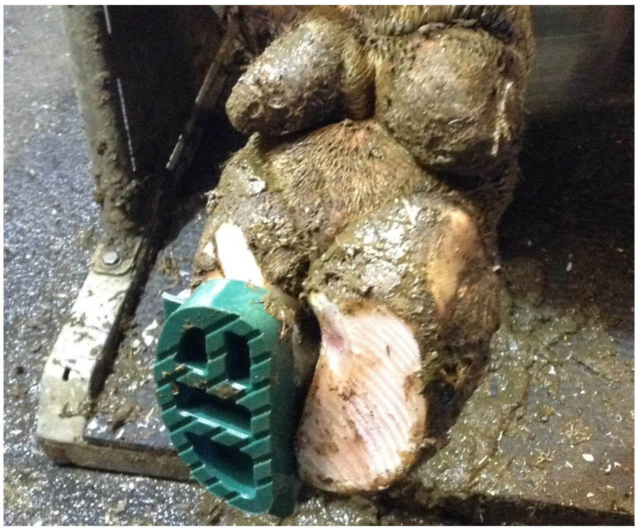


Figure 6. Plastic block attached to the outer claw of a cow's left hind hoof.

gave us their consent to conduct the experiment. Furthermore, they accompanied us and supported us throughout the entire experiment to ensure these requirements were met.

3. *Doing research that is relevant to participants and consistent with their welfare.* The results of our research suggest it is possible to automatically detect a condition that is painful for cows and highly detrimental to their health (e.g. might lead to death if not treated early enough). Our research has the potential to benefit the individual cows that participated in the experiment, as well as other cows. This research was conducted in the natural environment of the participating cows, an indoor stable located in the outskirts of Munich, Germany.

4. *Avoiding research procedures that may be harmful to participants.* According to the veterinaries that supported us throughout this study, attaching a sensor device and plastic block to cows and encouraging them to walk for less than 10 minutes did not cause any lasting harm to these cows. Veterinaries trimmed cows before attaching the plastic block to ensure the block was placed and fit properly to the claw. Trimming claws is a procedure undertaken to maintain a healthy hoof condition and prevent injury and disease. In addition, we limited the walking sessions to a maximum of 10 minutes per day and continued the data recording on a different day in order to reduce the level of fatigue caused to the cows.

5. *Assessing research proposals and obtaining expert support.* The cow interventions performed in this study were done by professional veterinarians and were approved by the ethics committee of the Ludwig Maximilian University in Munich, Germany to ensure no harm was done to the cows.

Data Collection

We recorded the motion of 10 cows while walking. Cows were chosen to maximize the diversity of age, weight and breed. Table 2 displays demographic information about each cow.

Cow	Age (Years)	Weight (Kg)	Breed (GH% / FV%)
1	5	910	31,25 / 68,75
2	5	680	68,75 / 31,25
3	5	720	43,75 / 56,25
4	7	560	87,5 / 12,5
5	6	700	31,25 / 68,75
6	4	780	62,5 / 37,5
7	3	680	0 / 100
8	5	640	100 / 0
9	4	610	0 / 100
10	3	700	0 / 100

Table 2. Information about the cows that took part in the controlled experiment. GH = German Holstein, FV = Fleckvieh.

Figure 7. A member of our team walks behind a subject cow during the controlled experiment.

We conducted three runs per cow. During the first run, cows walked normally. During the other two runs, cows walked with a plastic block attached to the outer claw of either their left or right hind hoof. We refer to gait observations produced by the first run as *normal* and to gait observations produced by the other two runs as *abnormal* gait observations. Each run lasted approximately 7 minutes.

We designed the experiment to resemble the conditions in which our approach would be used. We let cows walk in their usual environment rather than isolated walkways specially designed for the experiment. Furthermore, we included motion data of periods when cows stopped walking, turned and got bumped by other cows. Cows walked on two different types of ground: rubber and concrete. Figure 7 shows a cow walking during the experiment.

Data Analysis

We measured the performance of our approach according to the following metrics:

- Accuracy: The ability of our approach to classify gait observations correctly. It answers the question: what percent

Cow	Accuracy	Specificity	Sensitivity
1	95,8%	96,3%	83,3%
2	94,6%	95,1%	78,6%
3	81,9%	82,0%	78,6%
4	96,4%	97,2%	66,7%
5	97,3%	97,6%	80,0%
6	81,3%	81,7%	70,6%
7	95,4%	96,6%	62,5%
8	92,6%	93,0%	77,8%
9	87,2%	87,6%	73,1%
10	88,2%	88,7%	70,6%

Table 3. Accuracy, specificity and sensitivity of our approach at detecting *abnormal* gait observations.

of the classified gait observations (classified as *normal* or *abnormal*) is correct? Accuracy is calculated as: *number correctly classified instances / number total instances*.

- Specificity: The ability of our approach to identify *normal* gait observations. It answers the question: when a cow walks normally, what percent of its gait observations does our approach classify as *normal*? This is also referred to as true negative rate and computed as: *number instances detected as normal / number normal instances*.

- Sensitivity: The ability of our approach to identify *abnormal* gait observations. It answers the question: when a cow walks abnormally, what percent of its gait observations does our approach classify as *abnormal*? This is also referred to as "true positive rate" and computed as: *number instances detected as abnormal / number abnormal instances*.

The aforementioned measurements were computed by means of the leave-one-out cross-validation technique. First, we trained the SVM model with N-1 *normal* gait observations, where N is the total number of *normal* gait observations for a specific cow. Second, we used the model to classify the *normal* gait observation that was not used to train the model and every *abnormal* gait observation. We repeated this procedure N times; each time we left out a different *normal* gait observation. We averaged the accuracies, specificities and sensitivities obtained by the trained models in each repetition.

Results

Table 3 shows the accuracy, specificity and sensitivity of our approach for each cow involved in the experiment. Parameters used were: *outlier fraction* $= 0.15$ and *lameness threshold* $= -0.6$. We selected these values to maximize the sensitivity and specificity achieved by our approach for every cow based on the data collected during the controlled experiment.

Our approach classified gait instances with an average accuracy of 91.1% among all the cows (specificity = 91.6% and sensitivity = 74.2%). According to these results, our approach classifies 8.4% of the gait observations of a cow walking normally as *abnormal*. In contrast, when cows do indeed walk abnormally, our approach classifies 74.2% of their gait observations as *abnormal*. These results suggest our approach detects a deviation from a cow's usual walking pattern after this deviation occurs.

CONCLUSION

We presented and validated a new approach to detect deviation in cows' usual gait. In contrast to other gait monitoring approaches, our approach does not require expensive equipment or a specific setup and can be used in stables as well as outdoors. Furthermore, our approach considers the differences in gait of each cow by comparing the walking pattern of a cow to a baseline established for that particular cow during the first minutes of use of our device.

The results of the controlled experiment we conducted suggest that our approach would detect deviations from usual gait with an accuracy of 91.1%. A system based on the approach we propose could be used by veterinarians to gain information about the health condition of the cows in a herd. In particular, veterinarians might decide to examine a cow if the number of detected *lame* gait observations has exceeded considerably the usual amount for that particular cow. Our approach generates new gait observations every 15 seconds while cows walk.

We argue that the deviation from normal gait caused by lameness is more radical than the change in gait caused by the plastic block which we used to evaluate our approach. This is because cows suffering lameness will try to avoid pain by bearing as little weight as possible in the affected hoof. As a consequence, lame cows perform considerably shorter strides or stop using one limb all together. This causes an asymmetry in the gait, which is observable visually. In contrast, the change caused by the plastic block is more subtle. We were not able to assess visually whether a cow was using a plastic block or not by looking at its gait. As a consequence, we believe our approach might be more accurate at detecting deviations in gait caused by lameness than those caused by a plastic block. In the future, our approach will have to be validated in a longer-term field study.

In this work, we did not study the power consumption rate of our wearable device. However, a low power consumption rate that enables the device to function for several months is important in order to deploy our approach in a stable for longer periods of time. We will be able to reduce the power consumption of our wearable device considerably by different means. First, we will study the trade-off between accuracy of the computations and energy consumption when reducing the sampling frequency. Second, we will select a subset of the features we proposed in this paper. With less features, less computations will have to be executed by the wearable device, which will come at the cost of a loss in accuracy. Furthermore, we will study other unsupervised machine learning algorithms for anomaly detection that might require less computations.

ACKNOWLEDGMENTS

We would like to thank Dan Siewiorek for reviewing this paper, Ayca Ermis for helping collect the data used in this study and Nina Rittweg and the staff from the Lehr- und Versuchsgut Oberschleissheim from the Ludwig Maximilian University for supporting and guiding this project.

REFERENCES

1. Stefano Abbate, Marco Avvenuti, Francesco Bonatesta, Guglielmo Cola, Paolo Corsini, and Alessio Vecchio. 2012. A smartphone-based fall detection system. *Pervasive and Mobile Computing* 8, 6 (2012), 883–899.

2. Maher Alsaaod, Christoph Römer, Jens Kleinmanns, Kathrin Hendriksen, Sandra Rose-Meierhöfer, Lutz Plümer, and Wolfgang Büscher. 2012. Electronic detection of lameness in dairy cows through measuring pedometric activity and lying behavior. *Applied Animal Behaviour Science* 142, 3 (2012), 134–141.

3. Nuria Chapinal, Anne Marie de Passille, Matti Pastell, Laura Hänninen, Lene Munksgaard, and Jeff Rushen. 2011. Measurement of acceleration while walking as an automated method for gait assessment in dairy cattle. *Journal of dairy science* 94, 6 (2011), 2895–2901.

4. Guglielmo Cola, Marco Avvenuti, Alessio Vecchio, Guang-Zhong Yang, and Benny Lo. 2015. An on-node processing approach for anomaly detection in gait. *IEEE Sensors Journal* 15, 11 (2015), 6640–6649.

5. R M De Mol, G André, E J B Bleumer, J T N der Werf, Y De Haas, and C G Van Reenen. 2013. Applicability of day-to-day variation in behavior for the automated detection of lameness in dairy cows. *Journal of dairy science* 96, 6 (2013), 3703–3712.

6. P P J der Tol, J H M Metz, E N Noordhuizen-Stassen, W Back, C R Braam, and W A Weijs. 2002. The pressure distribution under the bovine claw during square standing on a flat substrate. *Journal of dairy science* 85, 6 (2002), 1476–1481.

7. T Van Hertem, E Maltz, A Antler, C E B Romanini, S Viazzi, C Bahr, A Schlageter-Tello, C Lokhorst, D Berckmans, and I Halachmi. 2013. Lameness detection based on multivariate continuous sensing of milk yield, rumination, and neck activity. *Journal of Dairy Science* 96, 7 (2013), 4286–4298.

8. Willem Maertens, Jürgen Vangeyte, Jeroen Baert, Alexandru Jantuan, Koen C Mertens, Sam De Campeneere, Arno Pluk, Geert Opsomer, Stephanie Van Weyenberg, and Annelies Van Nuffel. 2011. Development of a real time cow gait tracking and analysing tool to assess lameness using a pressure sensitive walkway: the GAITWISE system. *Biosystems Engineering* 110, 1 (2011), 29–39.

9. Clara Mancini. 2017. Towards an animal-centred ethics for Animal–Computer Interaction. *International Journal of Human-Computer Studies* 98 (2017), 221–233.

10. Hamutal Mazrier, Shlomit Tal, Eliezer Aizinbud, and Uri Bargai. 2006. A field investigation of the use of the pedometer for the early detection of lameness in cattle. *The Canadian Veterinary Journal* 47, 9 (2006), 883.

11. Lars Relund Nielsen, Asger Roer Pedersen, Mette S Herskin, and Lene Munksgaard. 2010. Quantifying walking and standing behaviour of dairy cows using a moving average based on output from an accelerometer. *Applied Animal Behaviour Science* 127, 1 (2010), 12–19.

12. Matti Pastell and Minna Kujala. 2007. A Probabilistic Neural Network Model for Lameness Detection. *Journal of Dairy Science* 90, 5 (2007), 2283–2292.

13. Matti Pastell, Johannes Tiusanen, Mikko Hakojärvi, and Laura Hänninen. 2009. A wireless accelerometer system with wavelet analysis for assessing lameness in cattle. *Biosystems engineering* 104, 4 (2009), 545–551.

14. Arno Pluk, Claudia Bahr, Ahmad Poursaberi, Willem Maertens, Annelies Van Nuffel, and Daniel Berckmans. 2012. Automatic measurement of touch and release angles of the fetlock joint for lameness detection in dairy cattle using vision techniques. *Journal of dairy science* 95, 4 (2012), 1738–1748.

15. Ahmad Poursaberi, Claudia Bahr, Arno Pluk, Annelies Van Nuffel, and Daniel Berckmans. 2010. Real-time automatic lameness detection based on back posture extraction in dairy cattle: Shape analysis of cow with image processing techniques. *Computers and Electronics in Agriculture* 74, 1 (2010), 110–119.

16. Xiangyu Song, Toon Leroy, Erik Vranken, Willem Maertens, Bart Sonck, and Daniel Berckmans. 2008. Automatic detection of lameness in dairy cattle-Vision-based trackway analysis in cow's locomotion. *Computers and electronics in agriculture* 64, 1 (2008), 39–44.

17. Vivi Morkore Thorup, L Munksgaard, P-E Robert, H W Erhard, P T Thomsen, and N C Friggens. 2015. Lameness detection via leg-mounted accelerometers on dairy cows on four commercial farms. *animal* 9, 10 (2015), 1704–1712.

18. Annelies Van Nuffel, Ingrid Zwertvaegher, Liesbet Pluym, Stephanie Van Weyenberg, Vivi M Thorup, Matti Pastell, Bart Sonck, and Wouter Saeys. 2015. Lameness detection in dairy cows: Part 1. How to distinguish between non-lame and lame cows based on differences in locomotion or behavior. *Animals* 5, 3 (2015), 838–860.

19. C Yunta, I Guasch, and A Bach. 2012. Short communication: Lying behavior of lactating dairy cows is influenced by lameness especially around feeding time. *Journal of dairy science* 95, 11 (2012), 6546–6549.

6.10 A Smart Textile Sleeve for Rehabilitation of Knee Injuries

This publication presents a first prototype of our KneeHapp system to track the rehabilitation progress of patients after an Anterior Cruciate Ligament (ACL) injury. This paper introduces the hardware (smart textile bandage and its electronics) and software application and describes the different rehabilitation exercises it supports.

The author of this Habilitation elicited the requirements together with orthopedists specialized in ACL injuries, developed the recognition algorithms to assess how well patients execute the different rehabilitation exercises, led a team of students who developed the user interface and wrote the paper.

Authors	Haladjian, J., Scheuermann, C., Bredies, K., & Bruegge, B.
Conference	Proceedings of the 2017 ACM International Symposium on Wearable Computers
Number of Pages	4
Type	Poster paper
Review	Peer Reviewed (3 Reviewers)
Year	2017
DOI	https://doi.org/10.1145/3123024.3123151

A Smart Textile Sleeve for Rehabilitation of Knee Injuries

Juan Haladjian
Technical University Munich
haladjia@in.tum.de

Constantin Scheuermann
Technical University Munich
constantin.scheuermann@in.tum.de

Katharina Bredies
University of Arts Berlin
katharina.bredies@udk-berlin.de

Bernd Bruegge
Technical University Munich
bruegge@in.tum.de

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
UbiComp/ISWC'17 Adjunct, September 11–15, 2017, Maui, HI, USA.
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5190-4/17/09...$15.00.
https://doi.org/10.1145/3123024.3123151

Abstract

A tear of the Anterior Cruciate Ligament (ACL) is a severe knee injury that requires up to a year of rehabilitation. Patients sustaining an ACL injury perform rehabilitation exercises mostly at home unsupervised. Orthopedists meet patients at time intervals as long as three months and lack quantitative ways to measure and keep track of the rehabilitation progress of each patient during these intervals. We present KneeHapp, a smart bandage and sock to support and keep track of the rehabilitation progress after an ACL injury. KneeHapp measures the quality of different ACL rehabilitation exercises and provides feedback to patients and orthopedists. We describe in detail how we constructed KneeHapp making use of smart textile technologies and provide insight into its software and quality of its measurements.

Author Keywords

KneeHapp; smart textile; e-textile; wearable; rehabilitation; knee

ACM Classification Keywords

J.m [Computer Applications]: Miscellaneous

Introduction

Tear of Anterior Cruciate Ligament (ACL) is a severe knee injury that occurs mostly among athletes. Currently, patients

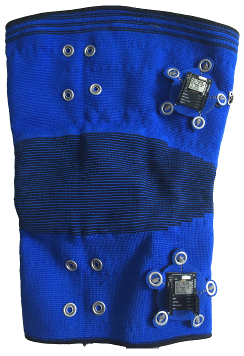


Figure 1: Outer layer the KneeHapp bandage.

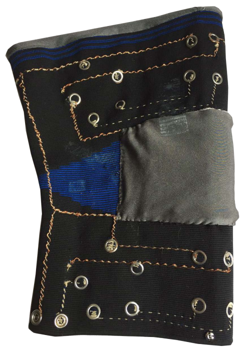


Figure 2: Inner layer of the KneeHapp bandage.

sustaining an ACL injury perform rehabilitation exercises mostly unsupervised and lack ways to measure the quality and track performance of their exercising. Orthopedists also lack tools to assess patients' rehabilitation progress and still rely on subjective observations to decide on the treatment.

Different studies have investigated rehabilitation of knee injuries using wearables and mobile devices [3, 1, 4, 5]. In contrast, the application of smart textiles to support rehabilitation of knee injuries remains vastly unexplored. Furthermore, most research in the field of computed-assisted rehabilitation focuses on different injuries [1, 5, 2]. The rehabilitation of an ACL injury has a specific set of requirements which derive from the fact that most patients of ACL injuries are athletes who engage in a rehabilitation program with the goal to return to sports.

In this paper, we introduce KneeHapp Textile, a smart bandage and smart sock that support different phases of the rehabilitation of an ACL injury including the recovery of flexibility, muscle strength, and final assessment to support orthopedists determine whether patients are ready to go back to sports. We address the construction and integration of textile sensors and connections and propose software solutions to quantify ACL rehabilitation progress. KneeHapp can be used by patients to obtain feedback about the quality of their exercising and by doctors to gain information about each patient's rehabilitation progress. KneeHapp is intended to be used by patients at home.

Background

We conducted a series of interviews with two orthopedists who perform ACL surgeries on a daily basis. Based on our interviews, we identified the following problems in the current practice:

Mobility Recovery

After a knee surgery, patients lose mobility on their knees. Recovering a certain range of motion (RoM) is required before patients can continue with the rehabilitation program. During first weeks after surgery, patients need to know the degree of flexion and extension of their knees.

Strength Recovery

After the injury and during the mobility recovery phase, patients suffer from muscular atrophy on the injured leg. Therefore, the second phase of the rehabilitation focuses on muscle building. A commonly performed exercise for strength recovery are one-leg squats and different variations of it. During a one-leg squat, patients stand on the injured leg, bend it as much as they can and then go back up into their initial position without deviating their knees from the line between the ankles and hips.

Currently, orthopedists and physiotherapists lack convenient ways to measure the angle of flexion of the leg during a squat. Therefore, the performance metric currently being used is the amount of squat repetitions. However, orthopedists agree that it would be convenient to have an objective way to measure *shaking* of the leg during a squat.

Back to Sports Assessment

Towards the end of the rehabilitation, orthopedists should assess whether patients are ready to start doing sports again. In order to do this, orthopedists compare the performance of the injured and healthy legs while executing different exercises, such as one-leg hops. One-leg hop is an exercise in which patients should jump forward on one leg as far as possible and land stably. Orthopedists measure the distance of the hop by placing a meter on the ground next to the area where the patient jumps.

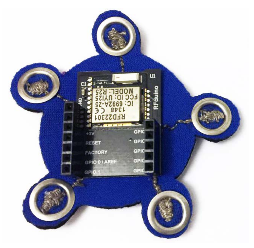


Figure 3: *Smart textile patches* hosting a microcontroller.

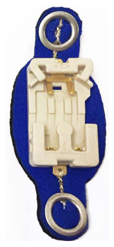


Figure 4: *Smart textile patches* hosting a battery holder.

KneeHapp Textile

The KneeHapp system consists of a smart compression bandage, a smart sock and an iPad App.

Smart Bandage
As a substrate for integration of the electronics, we initially considered strap bands that would be fastened to the upper and lower leg. We decided for the compression bandage for two reasons. First, patients have to wear a compression bandage after the surgery for two weeks. Second, the sensors in the bandage are placed in the same relative distance to each other. This avoids the risk of inconsistent measurements caused by a misplacement of the sensors, as might be the case when users misplace the strap bands.

The KneeHapp bandage has two layers of textile. The outer layer serves as a substrate for attachment of electronics. Two motion sensors, two microcontrollers and a coin cell battery holder are attached to the outer layer. The inner layer contains an electric circuit made of elastic conductive fiber. The inner layer also contains an integrated textile sensor that measures the amount of pressure being applied at the knee. Figures 1 and 2 show outer and inner layers of the bandage. An additional sleeve in the inner side protects the circuits from sweat and can be removed for washing purposes.

Electronic devices are sewn into *smart textile patches*. *Smart textile patches* are pieces of textile used to host an electronic device that are connected to the circuit in the bandage via metallic snap buttons. This enables users to remove the electronics for replacement or washing purposes. Figures 3 and 4 show a *smart textile patch*.

Smart Sock
The back-to-sport assessment involves mostly jump exercises. In order to gather information about the quality

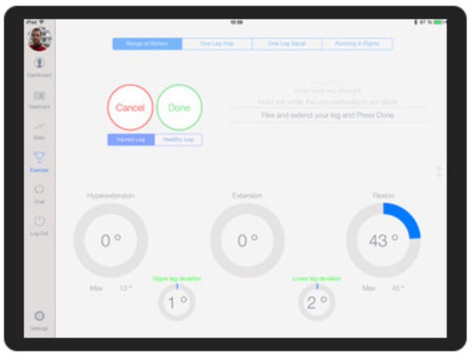


Figure 5: iOS App showing the angle of flexion of the leg.

of these exercises, we integrated pressure sensitive textile material in a sock. The pressure sensitive material has been realized by mixing conductive and non-conductive fibers. When the fibers are squeezed, the conductive fibers get closer together, which causes a higher conductance of the material. Two snap buttons on the upper side of the sock connect the sock to the bandage. The pressure signal is processed in a microcontroller attached at the lower-leg.

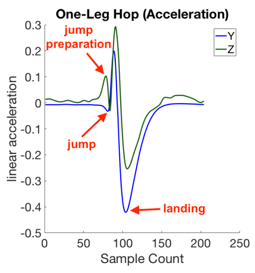


Figure 6: One-leg hop acceleration signal.

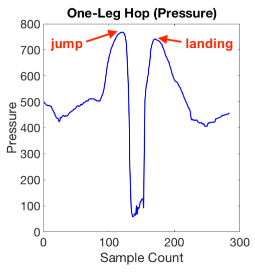


Figure 7: One-leg hop pressure signal.

KneeHapp Software

KneeHapp supports three rehabilitation phases: mobility recovery, strength recovery and the back to sports assessment.

Mobility Recovery

During the first months after the surgery, patients need to know the range of motion of their knees. KneeHapp aggregates the data from the textile pressure sensor at the knee and orientation measured by upper and lower IMUs. By convention, the angle of flexion of a leg should be equal to zero when the leg is relaxed on a flat surface. In order to account for the folding of the textile and the differences in leg anatomy and amount of swelling, we perform an initial calibration to determine the alignment of the IMUs and swelling of the leg.

Strength Recovery

KneeHapp provides live feedback to patients while performing one-leg squats in three ways. First, it calculates the angle of flexion of the leg during the squat and triggers a visual and auditive feedback when the minimal angle of the squat has been achieved. Second, it computes the degree of medial collapse of a patient's knee as the difference between the Euler angles of upper and lower IMUs. Third, it computes the degree of shaking of the leg as the standard deviation of the linear acceleration produced by the IMUs.

Back to Sports Assessment

KneeHapp measures and keeps track of the performance of one-leg hops. The duration of a one-leg hop is estimated based on data from the IMUs and pressure sensor in the smart sock. Figure 6 displays the linear acceleration signal produced by the upper IMU and Figure 7 displays the pressure signal measured by the smart sock during a one-leg hop.

Conclusion

In contrast to most computer-assisted solutions for rehabilitation, KneeHapp makes use of smart textile technology to address robustness, user comfort, energy consumption and modularity. The information measured by KneeHapp Textile could be used to keep track of the motion recovery after the surgery and to help orthopedists determine the degree of recovery of the injured leg.

REFERENCES

1. Ayoade, M., and Baillie, L. A novel knee rehabilitation system for the home. ACM Press (2014), 2521–2530.

2. Byrne, C. A., Rebola, C. B., and Zeagler, C. Design research methods to understand user needs for an etextile knee sleeve. In *Proceedings of the 31st ACM International Conference on Design of Communication*, SIGDOC '13, ACM (New York, NY, USA, 2013), 17–22.

3. Haladjian, J., Hodaie, Z., Xu, H., Yigin, M., Bruegge, B., Fink, M., and Hoeher, J. Kneehapp: a bandage for rehabilitation of knee injuries. In *Adjunct Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ACM (2015), 181–184.

4. Huang, K., Sparto, P. J., Kiesler, S., Smailagic, A., Mankoff, J., and Siewiorek, D. A technology probe of wearable in-home computer-assisted physical therapy. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, ACM (New York, NY, USA, 2014), 2541–2550.

5. Taylor, P., Almeida, G., Kanade, T., and Hodgins, J. Classifying human motion quality for knee osteoarthritis using accelerometers. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (Aug. 2010), 339–343.

Bibliography

[1] S. Abbate, M. Avvenuti, F. Bonatesta, G. Cola, P. Corsini, and A. Vecchio. A smartphone-based fall detection system. *Pervasive and Mobile Computing*, 8(6):883–899, 2012.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[3] A. Akl and S. Valaee. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, compressive sensing. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2270–2273, mar 2010.

[4] K. Altun and B. Barshan. Human Activity Recognition Using Inertial/Magnetic Sensor Units. In A. A. Salah, T. Gevers, N. Sebe, and A. Vinciarelli, editors, *Human Behavior Understanding*, pages 38–51, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] O. Amft. A wearable earpad sensor for chewing monitoring. In *SENSORS, 2010 IEEE*, pages 222–227. IEEE, 2010.

[6] O. Amft, H. Junker, P. Lukowicz, G. Troster, and C. Schuster. Sensing muscle activities with body-worn sensors. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4 pp.–141, apr 2006.

[7] O. Amft, M. Kusserow, and G. Tröster. Probabilistic parsing of dietary activity events. In *4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, pages 242–247. Springer, 2007.

[8] C. Amma, M. Georgi, and T. Schultz. Airwriting: a wearable handwriting recognition system. *Personal and ubiquitous computing*, 18(1):191–203, 2014.

[9] D. Ashbrook and T. Starner. MAGIC: a motion gesture design tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2159–2168. ACM, 2010.

[10] L. Atallah, B. Lo, R. King, and G.-Z. Yang. Sensor positioning for activity recognition using wearable accelerometers. *IEEE transactions on biomedical circuits and systems*, 5(4):320–329, 2011.

[11] M. Ayoade and L. Baillie. A Novel Knee Rehabilitation System for the Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2521–2530, New York, NY, USA, 2014. ACM.

[12] M. Bächlin, K. Förster, and G. Tröster. SwimMaster: a wearable assistant for swimmer. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 215–224. ACM, 2009.

[13] D. Bannach, O. Amft, and P. Lukowicz. Rapid Prototyping of Activity Recognition Applications. *IEEE Pervasive Computing*, 7(2):22–31, apr 2008.

[14] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*, pages 1–17. Springer, 2004.

[15] J. Barth, C. Oberndorfer, C. Pasluosta, S. Schülein, H. Gassner, S. Reinfelder, P. Kugler, D. Schuldhaus, J. Winkler, J. Klucken, and Others. Stride segmentation during free walk movements using multi-dimensional subsequence dynamic time warping on inertial sensor data. *Sensors*, 15(3):6419–6440, 2015.

[16] A. Belaid and J. Haton. A Syntactic Approach for Handwritten Mathematical Formula Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):105–111, jan 1984.

[17] N. B. Bharatula, M. Stäger, P. Lukowicz, and G. Tröster. Empirical study of design choices in multi-sensor context recognition systems. In *IFAWC: 2nd international forum on applied wearable computing*, pages 79–93, 2005.

[18] P. Blank, J. Hoßbach, D. Schuldhaus, and B. M. Eskofier. Sensor-based stroke detection and stroke type classification in table tennis. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 93–100. ACM, 2015.

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

[20] A. Bulling, J. A. Ward, H. Gellersen, and G. Troster. Eye movement analysis for activity recognition using electrooculography. *IEEE transactions on pattern analysis and machine intelligence*, 33(4):741–753, 2010.

[21] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy. Wearable sensors for reliable fall detection. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 3551–3554. IEEE, 2006.

[22] P.-Y. P. Chi and Y. Li. Weave: Scripting cross-device wearable interaction. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3923–3932. ACM, 2015.

[23] H. Cho and S. Yoon. Divide and conquer-based 1D CNN human activity recognition using test data sharpening. *Sensors*, 18(4):1055, 2018.

[24] G. Cola, M. Avvenuti, A. Vecchio, G.-Z. Yang, and B. Lo. An on-node processing approach for anomaly detection in gait. *IEEE Sensors Journal*, 15(11):6640–6649, 2015.

[25] F. Dadashi, A. Arami, F. Crettenand, G. P. Millet, J. Komar, L. Seifert, and K. Aminian. A hidden markov model of the breaststroke swimming temporal phases using wearable inertial measurement units. In *2013 IEEE international conference on body sensor networks*, pages 1–6. Ieee, 2013.

[26] A. K. Dey. Context-aware computing: The CyberDesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.

[27] J. Echterhoff, J. Haladjian, and B. Brügge. Gait Analysis in Horse Sports. In *Proceedings of the Fifth International Conference on Animal-Computer Interaction*, page 3. ACM, 2018.

[28] J. Echterhoff, J. Haladjian, and B. Brügge. Gait and Jump Classification in Modern Equestrian Sports. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, pages 88–91. ACM, 2018.

[29] M. Ermes, J. PÄrkkÄ, J. MÄntyjÄrvi, and I. Korhonen. Detection of Daily Activities and Sports With Wearable Sensors in Controlled and Uncontrolled Conditions. *IEEE Transactions on Information Technology in Biomedicine*, 12(1):20–26, jan 2008.

[30] G. W. Fitzmaurice. Situated information spaces and spatially aware palmtop computers. *Communications of the ACM*, 36(7):38–50, 1993.

[31] N. Gillian and J. A. Paradiso. The gesture recognition toolkit. *The Journal of Machine Learning Research*, 15(1):3483–3487, 2014.

[32] B. H. Groh, M. Fleckenstein, T. Kautz, and B. M. Eskofier. Classification and visualization of skateboard tricks using wearable sensors. *Pervasive and Mobile Computing*, 40:42–55, 2017.

[33] B. H. Groh, F. Warschun, M. Deininger, T. Kautz, C. Martindale, and B. M. Eskofier. Automated Ski Velocity and Jump Length Determination in Ski Jumping Based on Unobtrusive and Wearable Sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):53, 2017.

[34] T. Grosse-Puppendahl, Y. Berghoefer, A. Braun, R. Wimmer, and A. Kuijper. OpenCapSense: A rapid prototyping toolkit for pervasive interaction using capacitive sensing. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 152–159. IEEE, 2013.

[35] S. Ha and S. Choi. Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 381–388. IEEE, 2016.

[36] J. Haladjian. The Wearables Development Toolkit: An Integrated Development Environment for Activity Recognition Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol*, 3(3):submitted, 2019.

[37] J. Haladjian, K. Bredies, and B. Bruegge. Interactex: An integrated development environment for smart textiles. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2016 ACM International Symposium on Wearable Computers*, pages 8–15. ACM, 2016.

[38] J. Haladjian, K. Bredies, and B. Bruegge. KneeHapp Textile: A Smart Textile System for Rehabilitation of Knee Injuries. In *Proceedings of the 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 9–12. IEEE, 2018.

[39] J. Haladjian and B. Bruegge. Teaching wearable device development with the wearables development toolkit. *CEUR Workshop Proceedings*, 2308:27–28, 2019.

[40] J. Haladjian, B. Brügge, and S. Nüske. An Approach for Early Lameness Detection in Dairy Cattle. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 53–56. ACM, 2017.

[41] J. Haladjian, A. Ermis, B. Brügge, W. Plötz, and P. Buschner. Supporting rehabilitation after hip replacement with a mobile device carried in a pocket. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 452–457. ACM, 2017.

[42] J. Haladjian, A. Ermis, Z. Hodaie, and B. Brügge. iPig: Towards Tracking the Behavior of Free-roaming Pigs. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction*, ACI2017, pages 10:1—-10:5, New York, NY, USA, 2017. ACM.

[43] J. Haladjian, J. Haug, S. Nüske, and B. Bruegge. A Wearable Sensor System for Lameness Detection in Dairy Cattle. *Multimodal Technologies and Interaction*, 2(2):27, 2018.

[44] J. Haladjian, Z. Hodaie, S. Nüske, and B. Brügge. Gait Anomaly Detection in Dairy Cattle. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction*, ACI2017, pages 8:1—-8:8, New York, NY, USA, 2017. ACM.

[45] J. Haladjian, Z. Hodaie, H. Xu, M. Yigin, B. Bruegge, M. Fink, and J. Hoeher. KneeHapp: A Bandage for Rehabilitation of Knee Injuries. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 181–184. ACM, 2015.

[46] J. Haladjian, S. Nüske, and B. Brüge. iCow – A new approach for lameness detection. In *Proceedings of 19th International Symposium and 11th Conference Lameness in Ruminants*, pages 236–237, 2017.

[47] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[48] N. Y. Hammerla, S. Halloran, and T. Plötz. Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1533–1540. AAAI Press, 2016.

[49] B. Hartmann, L. Abdulla, M. Mittal, and S. R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07)*, pages 145–154, 2007.

[50] S. Houben, C. Golsteijn, S. Gallacher, R. Johnson, S. Bakker, N. Marquardt, L. Capra, and Y. Rogers. Physikit: Data engagement through physical ambient visualizations in the home. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1608–1619. ACM, 2016.

[51] S. Houben and N. Marquardt. Watchconnect: A toolkit for prototyping smartwatch-centric cross-device applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1247–1256. ACM, 2015.

[52] Y. A. Ivanov and A. F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):852–872, 2000.

[53] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[54] R. Jia and B. Liu. Human daily activity recognition by fusing accelerometer and multi-lead ECG data. In *2013 IEEE International Conference on Signal*

Processing, Communication and Computing (ICSPCC 2013), pages 1–4. IEEE, 2013.

[55] A. Khan, J. Nicholson, and T. Plötz. Activity Recognition for Quality Assessment of Batting Shots in Cricket using a Hierarchical Representation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):62, 2017.

[56] C. Ladha, N. Hammerla, E. Hughes, P. Olivier, and T. Ploetz. Dog's life: wearable activity recognition for dogs. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 415–418. ACM, 2013.

[57] D. Ledo, F. Anderson, R. Schmidt, L. Oehlberg, S. Greenberg, and T. Grossman. Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2583–2593. ACM, 2017.

[58] D. Ledo, S. Houben, J. Vermeulen, N. Marquardt, L. Oehlberg, and S. Greenberg. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 36. ACM, 2018.

[59] F. Li, K. Shirahama, M. Nisar, L. Köping, and M. Grzegorzek. Comparison of feature learning methods for human activity recognition using wearable sensors. *Sensors*, 18(2):679, 2018.

[60] Z. Li, Z. Wei, Y. Yue, H. Wang, W. Jia, L. E. Burke, T. Baranowski, and M. Sun. An adaptive hidden markov model for activity recognition based on a wearable multi-sensor device. *Journal of medical systems*, 39(5):57, 2015.

[61] P. Lukowicz, J. A. Ward, H. Junker, M. Stäger, G. Tröster, A. Atrash, and T. Starner. Recognizing workshop activity using body worn microphones and accelerometers. In *International conference on pervasive computing*, pages 18–32. Springer, 2004.

[62] K. Lyons, H. Brashear, T. Westeyn, J. S. Kim, and T. Starner. Gart: The gesture and activity recognition toolkit. In *International Conference on Human-Computer Interaction*, pages 718–727. Springer, 2007.

[63] B. Mariani, M. C. Jiménez, F. J. G. Vingerhoets, and K. Aminian. On-shoe wearable sensors for gait and turning assessment of patients with Parkinson's disease. *IEEE transactions on biomedical engineering*, 60(1):155–158, 2013.

[64] C. F. Martindale, M. Strauss, H. Gaßner, J. List, M. Müller, J. Klucken, Z. Kohl, and B. M. Eskofier. Segmentation of gait sequences using inertial sensor data in hereditary spastic paraplegia. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1266–1269, jul 2017.

[65] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher. Activity recognition and monitoring using multiple sensors on different body positions. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2006.

[66] J. Meyer, P. Lukowicz, and G. Tröster. Textile pressure sensor for muscle activity and motion detection. In *Wearable Computers, 2006 10th IEEE International Symposium on*, pages 69–72. IEEE, 2006.

[67] A. Millner and E. Baafi. Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In *Proceedings of the 10th International Conference on Interaction Design and Children*, pages 250–253. ACM, 2011.

[68] M. Moreau, S. Siebert, A. Buerkert, and E. Schlecht. Use of a tri-axial accelerometer for automated recording and classification of goats' grazing behaviour. *Applied Animal Behaviour Science*, 119(3-4):158–170, 2009.

[69] J. Muhlsteff, O. Such, R. Schmidt, M. Perkuhn, H. Reiter, J. Lauter, J. Thijs, G. Musch, and M. Harris. Wearable approach for continuous ECG-and activity patient-monitoring. In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 1, pages 2184–2187. IEEE, 2004.

[70] M. Nebeling, T. Mintsi, M. Husmann, and M. Norrie. Interactive development of cross-device user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2793–2802. ACM, 2014.

[71] V. P. Nguyen, S. H. Yoon, A. Verma, and K. Ramani. BendID: Flexible Interface for Localized Deformation Recognition. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 553–557, New York, NY, USA, 2014. ACM.

[72] F. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[73] S. Patel, D. Sherrill, R. Hughes, T. Hester, T. Lie-Nemeth, P. Bonato, D. Standaert, and N. Huggins. Analysis of the Severity of Dyskinesia in Patients with Parkinson's Disease via Wearable Sensors. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06)*, pages 123–126. IEEE, 2006.

[74] R. L. Peiris and S. Nanayakkara. PaperPixels: a toolkit to create paper-based displays. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design*, pages 498–504. ACM, 2014.

[75] L. Peng, L. Chen, Z. Ye, and Y. Zhang. AROMA: A Deep Multi-Task Learning Based Simple and Complex Human Activity Recognition Method Using

Wearable Sensors. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(2):74:1—-74:16, jul 2018.

[76] M. Pfeiffer, T. Duente, and M. Rohs. Let your body move: a prototyping toolkit for wearable force feedback with electrical muscle stimulation. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 418–427. ACM, 2016.

[77] S. Pirttikangas, K. Fujinami, and T. Nakajima. Feature selection and activity recognition from wearable sensors. In *International symposium on ubiquitious computing systems*, pages 516–527. Springer, 2006.

[78] T. Plötz, N. Y. Hammerla, A. Rozga, A. Reavis, N. Call, and G. D. Abowd. Automatic assessment of problem behavior in individuals with developmental disabilities. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 391–400. ACM, 2012.

[79] R. Ramakers, F. Anderson, T. Grossman, and G. Fitzmaurice. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 409–419. ACM, 2016.

[80] R. Ramakers, K. Todi, and K. Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, pages 2457–2466, 2015.

[81] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman. Activity recognition from accelerometer data. In *Aaai*, volume 5, pages 1541–1546, 2005.

[82] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita. Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767, 2016.

[83] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In *Computer applications in archaeology.* Tempus Reparatum, 1998.

[84] V. Savage, C. Chang, and B. Hartmann. Sauron: embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 447–456. ACM, 2013.

[85] V. Savage, S. Follmer, J. Li, and B. Hartmann. Makers' Marks: Physical markup for designing and fabricating functional objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 103–108. ACM, 2015.

[86] G. Schiboni and O. Amft. Automatic dietary monitoring using wearable accessories. In *Seamless Healthcare Monitoring*, pages 369–412. Springer, 2018.

[87] G. Schiboni and O. Amft. Sparse natural gesture spotting in free living to monitor drinking with wrist-worn inertial sensors. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, pages 140–147. ACM, 2018.

[88] B. N. Schilit and M. M. Theimer. Disseminating Active Mop Infonncition to Mobile Hosts. *IEEE network*, 1994.

[89] D. Schuldhaus, C. Zwick, H. Koerger, E. Dorschky, R. Kirk, and B. M. Eskofier. Inertial sensor-based approach for shot/pass classification during a soccer match. In *KDD Workshop on Large-Scale Sports Analytics 2015*, pages 1–4, 2015.

[90] T. Seyed, A. Azazi, E. Chan, Y. Wang, and F. Maurer. Sod-toolkit: A toolkit for interactively prototyping and developing multi-sensor, multi-device environments. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*, pages 171–180. ACM, 2015.

[91] D. Siewiorek, A. Smailagic, and A. Dey. Architecture and Applications of Virtual Coaches. *Proceedings of the IEEE*, 100(8):2472–2488, 2012.

[92] R. E. Stake. *Standards-based and responsive evaluation.* Sage, 2004.

[93] Z. Sun, X. Mao, W. Tian, and X. Zhang. Activity classification and dead reckoning for pedestrian navigation with wearable sensors. *Measurement science and technology*, 20(1):15203, 2008.

[94] M. Sundholm, J. Cheng, B. Zhou, A. Sethi, and P. Lukowicz. Smart-mat: Recognizing and Counting Gym Exercises with Low-cost Resistive Pressure Sensing Matrix. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 373–382, New York, NY, USA, 2014. ACM.

[95] E. M. Tapia, S. S. Intille, and K. Larson. Activity recognition in the home using simple and ubiquitous sensors. In *International conference on pervasive computing*, pages 158–175. Springer, 2004.

[96] R. Thompson, I. Kyriazakis, A. Holden, P. Olivier, and T. Plötz. Dancing with horses: automated quality feedback for dressage riders. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 325–336. ACM, 2015.

[97] P. Trahanias and E. Skordalakis. Syntactic pattern recognition of the ECG. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):648–657, 1990.

[98] G. Valentin, J. Alcaidinho, A. Howard, M. M. Jackson, and T. Starner. Creating Collar-sensed Motion Gestures for Dog-human Communication in Service Applications. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, ISWC '16, pages 100–107, New York, NY, USA, 2016. ACM.

[99] R. Varatharajan, G. Manogaran, M. K. Priyan, and R. Sundarasekar. Wearable sensor devices for early detection of Alzheimer disease using dynamic time warping algorithm. *Cluster Computing*, pages 1–10, 2017.

[100] R. Varatharajan, G. Manogaran, M. K. Priyan, and R. Sundarasekar. Wearable sensor devices for early detection of Alzheimer disease using dynamic time warping algorithm. *Cluster Computing*, 21(1):681–690, mar 2018.

[101] R. Walters, J. C. Principe, and S. . Park. Spike detection using a syntactic pattern recognition approach. In *Images of the Twenty-First Century. Proceedings of the Annual International Engineering in Medicine and Biology Society,*, pages 1810–1811 vol.6, nov 1989.

[102] E. Walton, C. Casey, J. Mitsch, J. A. Vázquez-Diosdado, J. Yan, T. Dottorini, K. A. Ellis, A. Winterlich, and J. Kaler. Evaluation of sampling frequency, window size and sensor position for classification of sheep behaviour. *Royal Society open science*, 5(2):171442, 2018.

[103] Z. Wang, M. Jiang, Y. Hu, and H. Li. An Incremental Learning Method Based on Probabilistic Neural Networks and Adjustable Fuzzy Clustering for Human Activity Recognition by Using Wearable Sensors. *IEEE Transactions on Information Technology in Biomedicine*, 16(4):691–699, jul 2012.

[104] J. A. Ward, P. Lukowicz, G. Troster, and T. E. Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1553–1567, 2006.

[105] T. Westeyn, H. Brashear, A. Atrash, and T. Starner. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *Proceedings of the 5th international conference on Multimodal interfaces*, pages 85–92. ACM, 2003.

[106] C.-J. Wu, S. Houben, and N. Marquardt. Eaglesense: Tracking people and devices in interactive spaces using real-time top-view depth-sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3929–3942. ACM, 2017.

[107] R. Xiao, C. Harrison, and S. E. Hudson. WorldKit: rapid and easy creation of ad-hoc interactive applications on everyday surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 879–888. ACM, 2013.

[108] J. Yang and D. Wigdor. Panelrama: enabling easy specification of cross-device web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2783–2792. ACM, 2014.

[109] K. Yatani and K. N. Truong. BodyScope: a wearable acoustic sensor for activity recognition. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 341–350. ACM, 2012.

[110] M. Zeng, H. Gao, T. Yu, O. J. Mengshoel, H. Langseth, I. Lane, and X. Liu. Understanding and improving recurrent networks for human activity recognition by continuous attention. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, pages 56–63. ACM, 2018.

[111] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th International Conference on Mobile Computing, Applications and Services*, pages 197–205. IEEE, 2014.

[112] B. Zhou, H. Koerger, M. Wirth, C. Zwick, C. Martindale, H. Cruz, B. Eskofier, and P. Lukowicz. Smart soccer shoe: monitoring foot-ball interaction with shoe integrated textile pressure sensor matrix. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, pages 64–71. ACM, 2016.