Tools and Methods to Support Opportunistic Human Activity Recognition

David Bannach

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

Thesis Advisor: Prof. Dr. Paul Lukowicz
Co-Advisor: Prof. Dr. Bernhard Sick
Dean: Prof. Dr. Klaus Schneider

Date of Defense: 6.3.2015

D386

Abstract

Today's pervasive availability of computing devices enabled with wireless communication and location- or inertial sensing capabilities is unprecedented. The number of smartphones sold worldwide are still growing and increasing numbers of sensor enabled accessories are available which a user can wear in the shoe or at the wrist for fitness tracking, or just temporarily puts on to measure vital signs. Despite this availability of computing and sensing hardware the merit of application seems rather limited regarding the full potential of information inherent to such senor deployments. Most applications build upon a vertical design which encloses a narrowly defined sensor setup and algorithms specifically tailored to suit the application's purpose. Successful technologies, however, such as the OSI model, which serves as base for internet communication, have used a horizontal design that allows high level communication protocols to be run independently from the actual lower-level protocols and physical medium access. This thesis contributes to a more horizontal design of human activity recognition systems at two stages. First, it introduces an integrated toolchain to facilitate the entire process of building activity recognition systems and to foster sharing and reusing of individual components. At a second stage, a novel method for automatic integration of new sensors to increase a system's performance is presented and discussed in detail.

The integrated toolchain is built around an efficient toolbox of parametrizable components for interfacing sensor hardware, synchronization and arrangement of data streams, filtering and extraction of features, classification of feature vectors, and interfacing output devices and applications. The toolbox emerged as open-source project through several research projects and is actively used by research groups. Furthermore, the toolchain supports recording, monitoring, annotation, and sharing of large multi-modal data sets for activity recognition through a set of integrated software tools and a web-enabled database.

The method for automatically integrating a new sensor into an existing system is, at its core, a variation of well-established principles of semi-supervised learning: (1) unsupervised clustering to discover structure in data, (2) assumption that cluster membership is correlated with class membership, and (3) obtaining at a small number of labeled data points for each cluster, from which the cluster labels are inferred. In most semi-supervised approaches, however, the labels are the ground truth provided by the user. By contrast, the approach presented in this thesis uses a classifier trained on an N-dimensional feature space (old classifier) to provide labels for a few points in an (N+1)-dimensional feature space which are used to generate a new, (N+1)-dimensional classifier. The different factors that make a distribution difficult to handle are discussed, a detailed description of heuristics designed to mitigate the influences of such factors is provided, and a detailed evaluation on a set of over 3000 sensor combinations from 3 multi-user experiments that have been used by a variety of previous studies of different activity recognition methods is presented.

Acknowledgments

I would like to thank Prof. Paul Lukowicz for giving me the opportunity to work in such an interesting research field. I could not imagine doing my PhD with somebody else. I'm deeply impressed by his always positive and inspiring attitude. I would also like to thank Prof. Bernhard Sick for co-examining my thesis and for the valuable discussions we had. I will never forget our surprising encounter on a remote island.

I wish to thank all my colleagues at the ESL at University of Passau and the CSN at UMIT in Tyrol for providing an inspiring and motivating working atmosphere. I specially like to thank Kai Kunze, Georg Ogris, and Kamil Kloch for the countless creative and fruitful discussions.

Throughout the WearIT@Work, MonAMI, and OPPORTUNITY projects I had the pleasure to work together with so many great colleagues which was an enormous enrichment. I want to specially thank Oliver Amft, Alberto Calatroni, Daniel Roggen, Prof. Alois Ferscha, Thomas Ziegert, Tobias Klug, Andreas Zinnen, Thomas Stiefmeier, Marco Luca Sbodio, Thierry Milin, and Jean-Bart Bruno for being such wonderful peers and for the great times we spent together.

I'm enormously grateful to all the contributors of the CRN Toolbox. They made this project coming alive. And I also like to thank the students, which I had the pleasure to supervise, and which implemented many useful parts of the entire toolchain.

Finally, I want to thank the most important people to me, my parents and my brother. They have always been there, supporting me in any way possible. I thank Yvonne and Arwed Boitel for the great motivation – without them I certainly would have missed this whole journey. And I thank Emi Auer for all her love and the support I received in the last years.

Zurich, 9.8.2015
David Bannach

Contents

Chapter 1

Motivation

Recent years have seen a massive growth of mobile, sensor equipped computing devices. Smartphones recently counted for half of all mobile phones in U.S.[1] and for other regions of the world a similar trend is observed, making them to the first real-world pervasive computing platform. Most of the devices can locate themselves within world coordinates using cell tower and WIFI access point information, or a global positioning system (GPS) receiver, and have a full inertial measurement (3-axis accelerometer, -gyroscope, and -compass) unit built-in. Publicly available software development kits and distribution channels for third party applications made development for such platforms very popular. A vast majority of smartphone applications concentrate on the device's graphics capabilities, multi-touch control, and sharing through social networks. Others, that are context-sensitive, mostly are limited to contextual information such as location or device orientation, as those are the pieces that are immediately available. There are only few applications which are going beyond and try to leverage the potential of the available sensors, e.g., a fitness app counting steps while running. Accessory sensors are available to be worn on the body (shoe, wrist) or mounted on bicycles for more accurate fitness tracking, or medical units to measure blood pressure. Gaming consoles use inertial measurement units in wireless controllers or vision based body tracking to let players interact with games in more realistic manner with physical movements.

All those applications that incorporate activity recognition are based on a rather vertical design. I.e., the activity recognition layer is tailored to a single application and relying on a narrowly defined set of sensors. The individual systems may do a good job within their specifications, there are however only marginal synergies between them. Hence, each new system has to be built from ground up again. Other, successful technologies have built on a horizontal, layered design, e.g., the OSI model which serves as the fundamental framework for all computer communications. The OSI model defines clear abstraction layers to decouple issues of physical signal transmission from higher level protocols and applications.

Within this thesis we explore the task of creating and running activity recognition systems and we investigate how opportunistic activity recognition systems

[1] `http://blog.nielsen.com/nielsenwire/online_mobile/smartphones-account-for-half-of-all-mobile-phones-dominate-new-phone-purchases-in-the-us`

can autonomously improve their performance at runtime by including new sensors which were not known before.

To the first topic we contribute an integrated toolchain which supports all phases during the development of activity recognition systems. It provides tools for recording large-scale, multimodal datasets (which are necessary for training of the recognition algorithms), tools for annotating and sharing of such data sets, and a flexible toolbox of reconfigurable components for online processing of sensor data streams for activity recognition. The toolchain has been used already in various research projects and publications, and it is freely available as an open-source project[2].

Secondly, we present a novel method for including new sensors into existing activity recognition systems to improve its accuracy without or with minimal user input and to ensure long-term evolution of such systems. The method builds upon well established principles of semi-supervised learning and enhances them to be applicable to the particular problem. Specifically, we provide heuristics that cope with the issues apparent when applying the method to activity recognition systems in real world scenarios, and we present a detailed evaluation on a set of over 3000 sensor combinations from 3 multi-user experiments that have been used by a variety of previous studies of different activity recognition methods.

1.1 Background

In the early 1990s, in a time where computers have just emerged from large mainframe machines to personal computers on people's desktops, Mark Weiser articulated the vision of ubiquitous computing [Wei91] – computers that disappear in the background and unobtrusively interact with the user to deliver their information:

> "Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as a walk in the woods."

Obviously, this vision was well ahead of its time as the hardware and software technology to realize it was just not there. Nevertheless, it inspired and motivated researchers all around the world to work in this direction. The term "pervasive computing" arose to describe the more short-term realizable goals of Weiser's idealistic vision. Distributed systems and mobile computing are named as two basic technologies that enable pervasive computing [Sat01]. Yet, the research agenda goes beyond and also incorporates additional research thrusts such as smart spaces [Har02, Bru00, Wan02] and invisibility or – as an approximation – minimal user distraction [Gar02].

Only applications that are aware of the user's context are able to adapt their behavior in such a way that minimizes its visibility for the user. Early context-aware applications focused on the user's location and presence of other people or resources [Sch94, Pri00, Har02], but soon it became obvious that the user's state – whether physical, physiological, or mental – is also an important aspect of context [Abo99, Sch99b, Sch99a], and it turned out to be more difficult to sense.

[2] `http://contextdb.org/main/_design/contextDB/tools.html`

Smart spaces may have difficulties determining the user's state or high-level activity as the the specific environment is entered usually for a limited time only. In contrast, wearable computers worn close to the body over longer periods of time similar to clothing and accessories have a much higher chance to capture the user's state. Early work concentrated on recognizing human modes of locomotion (e.g., sitting, standing, walking) from body-worn accelerometers by analyzing signal thresholds [Far99, Lee02, Sie03] or by applying neural networks [VL00]. Other work extended the range of recognized context to high-level activities of daily living by just using accelerometer data and machine learning methods [Bao04] or by investigating additional sensors, e.g., microphones (voice, ambient), temperature sensors (skin, ambient), light, and galvanic skin response [Sie03, Kra03]. More recent work showed that recognizing fine-grained physical actions (e.g. opening a door, turning a screw) or gestures is feasible [War06, Sti06b]. Recognizing fine-grained actions may help inferring higher-level activities which contribute valuable context information that allows context-aware applications to automatically adapt their behavior in the sense of minimal user distraction. Gestures that are recognized in real-time may be used to instantly control applications or computer games in an intuitive way.

1.2 Challenges for Activity Recognition

Activity recognition from body-mounted or environmental sensors is based on parameters found in data that was previously recorded from the sensors during execution of the different activities. Often, those parameters are statistical measures. In theory, finding the same parameters in live data streams is enough to determine the current activity. However, real world situations raise additional challenges to activity recognition systems.

Recognition Accuracy The key performance measure for activity recognition systems is the recognition accuracy. Only accurate information about the user's context can help computers or applications to minimize user distraction. Wrong information instantly leads to the opposite effect. The recognition accuracy therefore is a crucial measure. There are several factors that come into play for achieving high accuracy:

Quality of Training Data Most obviously, the machine learning methods need lots of high quality training data in order to build good classifier models. Acquiring and annotating such training data is still an expensive task and it usually has to be repeated for each new sensor setup.

Overfitting Problem The method must avoid overfitting, i.e., prevent the case where the classifier model is too tightly fitted to patterns that are unique to the training data sample and that do not occur equally well in real-life sensor data. The model must be specific enough to capture the patterns stemming from the activities to be recognized and, at the same time, it must be general enough to allow for enough variation in those patterns. Different users, for instance, may perform activities in slightly different ways, or even a single user may change the way how activities are performed, e.g., when getting tired.

Application-specific Optimization Depending on the application, certain aspects of the general recognition accuracy may be more important than others. A context-aware application which tries to minimize user distraction, for instance, is primarily interested in getting context information with high *precision*, i.e., wrong context information would make the application to irritate the user more than if no context information was available. On the other hand, a medical monitoring system would be interested in a high *recall*, i.e., it is important to detect *all* (or as many as possible) instances of the monitored activities. Furthermore, accurate timing of detected activities may be of importance. This may be required at different levels. It may just concern the duration of events, the temporal sequence, or their exact start and end times.

Sensor Availability The availability of sensors in wearable systems and in smart spaces is dynamic. Especially with wearable sensors, the user may put on different sensors at different times, sensors may run out of battery or just fail for other reasons. Placement of sensors on the body may switch depending on user's preferences or they may be moved and rotated unintentionally during wearing. Sensors mounted in the environment may be affected similarly and if the user is moving from one smart space into another her wearable system suddenly faces different environmental sensors and services. Activity recognition systems must be able to cope with the dynamic availability of sensors in order to be useful in real-life situations.

New Sensors During the lifetime of an activity recognition system new sensors can appear eventually which were not known at design time of the system or which have just not been considered. This may for instance be a shoe sensor to support sports monitoring or a smartphone being replaced by a newer generation device which provides extra sensors. Those sensors may provide valuable information for recognizing certain activities or they may help distinguishing some activities in combination with other sensors, and for many other activities they are just not useful at all. But for those activities for which they deliver useful information it would be favorable to integrate them into the activity recognition system in order to improve its performance. The common scheme for building activity recognition systems, however, requires the training data for all sensors to exist beforehand. Methods are necessary to allow for dynamic integration of new sources of information into existing systems depending on the source's usefulness regarding the specific recognition goals of the system.

The addition of new sensors also rises the question of which feature(s) to extract from the new data source, in order to achieve the best recognition accuracy. We will, however, not cover the feature selection in this thesis.

Coding Effort As different the scope of activity recognition systems may be, they have many system architecture related problems in common. Most prominently these are the reading of measurements from the attached sensors, filtering of the sensor signals to reduce unwanted noise and to extract features, classification of the feature vector, and sending the result to the context-aware application. For single-sensor systems this seems

quite straightforward. However, as soon as multiple, distributed sensors of possible different modalities are involved the systems complexity quickly increases. Issues of temporal synchronization and alignment of unequally sampled sensor signals arise, data streams from different sources need to be merged carefully, and selected data channels may require different filtering. Nevertheless, most of such problems occur similarly in many systems and the basic architecture is primarily data driven. There is a large potential of synergy. A common codebase and framework could facilitate the construction of activity recognition systems and would enable researchers to concentrate on problems beyond system architecture. Similarly, the process of recording and annotating large sets of training data for the machine learning methods appears repeatedly for new projects and the required effort is strongly increasing with the number of sensors and the quality targets. This demands for new methods and tools that support this process and that allow for collaborative sharing of efforts.

1.3 Related Work

Many of the challenges for real life activity recognition have been addressed in related work, at least to a certain extent. The achievements are summarized in this section.

1.3.1 Context-Aware Frameworks and Pervasive Middleware

Many approaches have been presented to establish an abstraction layer between applications that demand for contextual information and the actual recognition of context from sensor signals.

Dey [Dey01] proposes an application framework called Context Toolkit for simplifying and unifying the creation of context sensitive applications. Context is defined here as typically being the location, identity, and state of people, groups, and computational and physical objects. The framework abstracts from the possibly complex details of context recognition with the concept of "context widgets". The context widgets allow application developers to concentrate on the core of their application without having to handle the details of how the context information is acquired, similar to how GUI widgets simplify the creation of GUI applications. The focus is on handling context information within the application layer. There is no support for improving the accuracy or simplifying the context recognition from sensor signals.

PCOM [Bec04] is a component system for pervasive computing, focusing on spontaneously networked environments in which devices are connected on-the-fly. Interdependencies of components are contractually specified, i.e., application developers just need to specify their component's requirements on the executing platform, the functionality provided by their component, and its dependencies on other components. The system also provides abstractions for adaption strategies in case the availability or quality of related components is changing. Such strategies basically prioritize possible components based on user preferences.

RUNES [Cos07] is a middleware for networked embedded environments which is designed for highly dynamic and reconfigurable systems running on heterogenous sensor networks. The individual devices are able to reconfigure their behavior and their information dissemination strategies as they become damaged, e.g., under emergency conditions. Operators may dynamically reprogram the networked sensors and actuators. The core of RUNES is designed to be platform- and language independent such that it may be deployed on small sensor nodes. Middleware components are deployed regarding the dynamic resource constraints.

These are only a few examples of context-aware frameworks and pervasive middleware (more can be found in [End05]), still they reveal a mayor issue with such middleware: while they focus on abstracting and communicating contextual information there is no real support for the complex process of context- or activity recognition. Yet, without a reliable base of context recognition those frameworks can not manifest their full potential.

A prominent example of a successful abstraction model is the OSI network stack [Zim80]. It cleanly separates lower-level data transmission tasks from higher-level protocol implementations. It does this at seven distinct layers which cover the complete range from physical media access up to the application. Hence, it allows applications to communicate independently of underlying protocol implementations and actual physical connection medium. Frameworks for context- and activity recognition could profit from such clean abstraction layers and thorough implementations. One approach of a layered architecture for pervasive systems is shown e.g. in [Agü12], yet it is still a high-level approach, neglecting the specific requirements at the lower layers.

In the audio processing and data visualization domains it is common to model the processing chain as a set of independent processing components which are configured and connected in a graphical editor [Puc88, She96]. This approach may benefit the activity recognition domain where streams of sensor data need to be processed in real-time.

1.3.2 Online Activity Recognition Systems

The following works highlight the challenges that arise at the level of context and activity recognition, where the actual sensing hardware is interfaced and where the sensor signals are processed and evaluated in real time.

Schmidt et al. [Sch99a] proposed a layered architecture for real-time context recognition from low-level sensors. Raw values received from a sensor are collected in "cues" for which a symbolic value is assigned. Cues either provide a summary of the values over time or they help to extract features from the raw data that characterize the data over the last period of time. At the next level, a set of logical rules transforms the cues from multiple sensors into a description of current context. Finally, the provided scripting primitives allow applications to react on context changes to implement the desired behavior.

Randell et al. [Ran00] presented a low power activity recognition system integrated into a tour guide application. The authors first collected data from a two-axis accelerometer from 10 subjects performing 6 different activities (sitting, standing, walking straight/up/down, running) for training and evaluating the system. Two features (RMS, 2 seconds integral) were computed on each sensor channel and fed into a neural network classifier. An accuracy of around 95%

was achieved and the system already delivered valuable information to the tour guide application which could adapt its behavior to the user's locomotion mode.

In [Cho08] the evolution of an embedded activity recognition system (MSP, also used in [Les06]) is described. The MSP is an integrated wearable device with ARM CPU, wireless connectivity, and various sensors. The system architecture evolved in an iterative process that revealed a core set of activity recognition component requirements. The initial deployments focused on gathering real-world activity traces for offline analysis. The unreliable Bluetooth connection forced them to use a wired connection for logging the data streams. A second version of the device incorporated enough storage and CPU power to record the sensor data locally and to process them in real-time for activity recognition. The classifiers were trained offline and then implemented on the device, configurable through an XML file. Extended feature selection and classification methods have been studied but they have still to be implemented on the wearable device. The system was used in several projects concerning the sensing of user physical activity level.

The above examples all revealed online activity recognition systems running on dedicated hardware. The earlier ones had a rather narrow scope and their algorithms were tightly tailored to the application needs and to the low power limitations of the hardware platform. The last example could already benefit from advanced hardware to support more sophisticated activity recognition algorithms. However, those systems were not designed to support a wide range of applications and have limited flexibility. Particularly, the available algorithms are tailored to application needs and no efforts are presented to facilitate addition of new algorithms or recombination of existing ones for different application targets.

1.3.3 Activity Recognition Studies

Bao et al. [Bao04] presented a study of activity recognition from accelerometers mounted on the body. A set of 20 everyday household activities were recorded from 20 subjects without direct researcher supervision. The subjects themselves labeled start and end time of each activity. Before starting a recording, the clocks of the individual sensor boards were synchronized. Additionally, the boards were shaken together at the beginning and end of each recording and after the recording, the peaks of the distinct shaking pattern in the signals were visually aligned to compensate for individual clock skew. From the recorded data a set of features (mean, energy, frequency-domain entropy, and correlation) were extracted with a sliding/jumping window method. The feature vectors were used for training and evaluating different classifiers (C4.5 decision tree, naive Bayes) with the WEKA machine learning toolkit. C4.5 performed best with 84% overall accuracy (leave-one-subject-out validation). Only few activities such as "stretching" and "folding laundry" were often confused because of similar feature vectors.

Lester et al. [Les06] presented a study of activity recognition from a single body-mounted sensing unit covering diverse sensing modalities, including acceleration, magnetic field, ambient sound, air pressure, humidity, temperature, and ambient light. Data was recorded from 12 subjects performing 8 activities (sitting, standing, walking straight/up/down, riding elevator up/down, brushing teeth). An observer was instructing the subjects and labeled start and end

times of activities by using a simple annotation program on an iPAQ handheld device. The data from the sensing unit was recorded on a small notebook computer carried by the subject. In an offline analysis, the data was processed in three stages. First, a set of 9 features (time and frequency domain) was computed with a jumping window method, resulting in a 651 dimensional feature vector at 4 Hz. Then, AdaBoost was used to automatically select the 50 best features and to learn an ensemble of discriminative classifiers. Finally, the output of those classifiers were used as input into HMMs to ensure temporal smoothness. On the 8 activities an overall accuracy of nearly 90% was achieved, and limiting the number of sensing modalities to three (audio, pressure, acceleration) did yield similar results. Of the 30 hours of recorded data only 12 hours could be used because of memory limitations in the offline analysis.

Those studies highlight the effort needed for training and evaluating reliable activity recognition methods. Besides presenting activity recognition methods targeted for real world scenarios there is no mention of any tools to facilitate the process of recording, handling, or utilizing large activity recognition data sets, except for the WEKA machine learning toolkit.

1.3.4 Data Labeling

A major effort in training of activity recognition methods to specific scenarios is the creation of properly annotated training data sets. The manual labeling of training instances, which is normally necessary, is time consuming and error prone. Below is a selection of works which investigated alternative methods that reduce the amount of labeled data needed for activity recognition.

Szewcyzk et al. [Sze09] compared four different labeling strategies. They used data from a smart apartment which had event based sensors installed (e.g., motion detectors) and the goal was to recognize high-level activities such as sleeping, eating, personal hygiene, preparing meal, working at computer, watching TV, and "other". Annotating the raw sensor data (timestamped events) took the most time and yielded the worst classification accuracy. Including feedback from the experiment subjects (the residents) reduced the time needed for annotation and improved the classification accuracy. Visualizing the recorded data in a 3D model of the apartment helped to reduce annotation time by a factor of 2 (3.3 with feedback) and further increased recognition accuracy. However, implementing and debugging the visualization took the most time.

Stikic et al. [Sti11] investigated new strategies to reduce the amount of labeled data needed for long-term activity recognition by applying "weakly supervised" learning methods. Specifically, they propagated labels from known instances to the unlabeled data using a graph structure. The graph connects two nodes (data points) if they are close to each other in feature space or in time. Applying this method on datasets of high-level daily activities, the experience sampling interval could be increased up to 60 minutes with almost no accuracy loss, and the method could successfully deal with labeling errors of up to 20% of the data.

In earlier work, Stikic et al. [Sti08b] explored the use of semi-supervised techniques for activity recognition. They found self-training and co-training to be capable to learn from very limited amount of labeled training data. With a pool based active learning method they achieved comparable or sometimes higher accuracy than with fully supervised methods.

1.3.5 Opportunistic Activity Recognition

Instead of deploying sensors specific to application scenarios, opportunistic activity recognition systems [Rog13] follow a new paradigm in which the activity recognition methods themselves adapt to the available sensor data.

Kunze et al. [Kun08] presented a set of heuristics to handle sensor displacement in onbody activity recognition systems. The method is based on ignoring rotation dominated accelerometer signal segments and using a gyroscope to compensate for lost rotation information. In making activity recognition displacement tolerant the robustness for long term deployment increases and with it the usability and user acceptance.

Förster et al. [För09] introduced a different approach for robustness against unintentional sensor displacement. The method is based on online unsupervised classifier self-calibration. On reoccurring context instances they apply an online learning algorithm to adjust the decision boundaries according to the changed statistics of the classes. The approach proved feasible to compensate for small sensor displacements, e.g., when a sensor is slowly slipping along a limb segment. Abrupt and larger displacements can not be handled sufficiently by this method.

Calatroni et al. [Cal11, Cal10] presented a method for transferring the activity recognition capability of an existing sensor node to a newly deployed and untrained node, without user intervention. Labels of recognized activities from the first nodes are transferred to the latter, which incrementally associate their sensor signals to the received labels. Using behavioral assumptions such as "open drawer" implies "standing" or that the user walks away after closing a drawer, the method is shown to work across different sensing modalities in a posture and modes of locomotion recognition scenario.

1.3.6 Thesis Objectives

With this thesis we aim to find tools that are structuring and simplifying the creation of efficient real-world activity recognition systems such that different activity recognition methods may be studied in real-world situations with less effort on the implementation side and with more focus on the actual goal of the methods. Secondly, we aim at methods that enable such systems to continuously and autonomously evolve over time by opportunistically exploiting available resources in changing environments.

1.4 Contributions

This thesis makes contributions at two different stages of real-world activity recognition systems. The first concerning the tools for creating and handling such systems, the second dealing with long-term automatic adaptation and evolution of the system to address the changing sensor infrastructure.

1.4.1 Software Tools for Creating Online Activity Recognition Systems

We provide an efficient, reconfigurable, software toolbox for rapid prototyping of online activity recognition systems. This toolbox builds upon reusable, parametrizable components which can flexibly be connected to establish the

specific data acquisition and processing chain needed for each application. It simplifies the construction of activity recognition systems to just connecting and configuring components in a GUI in many cases. Specifically, a broad set of components for

- reading live data from various sensors or external software tools,
- filtering in time and frequency domain and extracting features,
- merging and splitting individual data streams,
- segmentation of data streams,
- classifying data stream segments into defined activity classes, and
- writing live data or results to output devices or external tools

is provided, embedded in a framework which supports the dynamic instantiation and configuration of individual components and which assures the data flow between the components. The architecture is designed to be open for extensions such as, e.g., components for evaluation. The software is freely available as an open source project[3] and is used in various research projects and publications.

We investigate the clock synchronization problem in heterogenous sensor networks and present a solution for automatically synchronizing data streams received from such sensors. Our synchronization method is based on the detection of physical events across multiple sensors of arbitrary modality to determine the clock offset of the involved sensors. A solution is provided to, among all detected events, exactly identify the event on each data stream which corresponds to the same physical event, regardless of the sensing modality. We describe fundamental properties and bounds of our event-based synchronization approach. In particular, we show that the event timing relation is transitive for sensor groups with shared members. In three studies with real-world sensor data, including a data set with four different sensing systems recorded from 5 subjects and 5 hours in total, we show the feasibility of our approach and achieve automatic stream synchronization with less than 0.3 s error for 80% of the synchronization actions.

We provide an integrated toolchain to support the entire process of creating online recognition systems, which includes

- recording of large-scale, multimodal data sets,
- annotating, managing, and sharing of the data sets,
- utilizing the data sets for training and evaluation of activity recognition algorithms, and
- deploying them in online activity recognition systems.

The toolchain is built around the above mentioned activity recognition toolbox with additional tools to support the various stages, and a central, web-based database which serves as a portal to access all components of the toolchain. The database portal is publicly accessible at `contextdb.org` and already provides

[3] http://crnt.sf.net

a large, annotated data set recorded from 72 sensors belonging to 10 different modalities in totally 60 recording sessions.

The above software tools evolved in the attempt to answer the question if the creation of activity- and context recognition systems can be structured in a way that allows for an autonomous reconfiguration and evolution of such systems.

1.4.2 Automatic Integration of New Sensors into Existing Systems

An important scenario for opportunistic activity recognition systems is the appearance of a new sensor which possibly can contribute valuable information to better discriminate the activities to be recognized. The inclusion of such a sensor may hence improve overall system accuracy. Instead of manually retraining the system with the new sensor we investigate ways for automatic integration of the new sensor using the existing recognition model and unlabeled or very sparsely labeled data from the new sensor, but without knowing a priori the usefulness of the new sensor.

The goal would be that with the addition of the new sensor the recognition accuracy would increase for at least some occasions without negatively affecting the performance in other occasions. The trivial approach to achieve this is to

1. find the cluster structure in the extended feature space from the joint sensors set,

2. identify points within the clusters that can be labeled with high confidence using the original classifier model,

3. propagate the labels from such points to entire clusters based on the assumption that clusters and classes are correlated, and

4. use the cluster labels to improve accuracy for points that can not be accurately classified with the old sensor alone.

For trivial setups this approach may provide the desired results but, unfortunately, in real life activity recognition systems three basic assumptions are not met and those are the ones that make this problem challenging:

1. the cluster structures found in the extended feature space do not always correspond to the activity classes,

2. the specific relationship between original and new feature space not always allows to label at least a few data points in each cluster, and

3. the availability of a large amount of representative training data is not always given.

The core contribution of this thesis, regarding opportunistic activity recognition, is to explore extensions to the trivial heuristics and to show their potential to:

1. achieve an improvement with the more complex distributions that usually appear in real life sensor data, and to

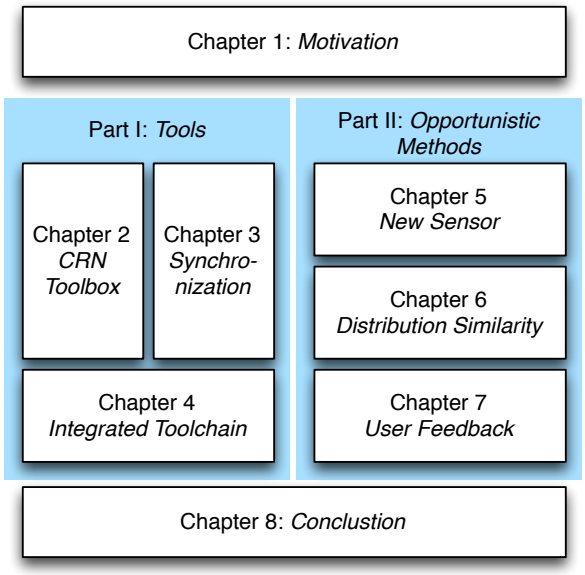


Figure 1.1: Thesis outline.

2. reduce the probability that an unfavorable distribution (or a sensor that does not provide useful information) leads to a performance decrease.

We discuss different factors that make a distribution difficult to handle, provide a detailed description of heuristics designed to mitigate the influences of such factors, and present a detailed evaluation on a set of over 3000 sensor combinations from 3 multi-user experiments that have been used by a variety of previous studies of different activity recognition methods.

1.5 Thesis Outline

The thesis is structured into two parts with the first part covering tools for building and maintaining activity recognition system, and the second part presenting the in-depth study of a novel method for opportunistic activity recognition systems. Figure 1.1 illustrates the outline of the thesis. Most chapters are based on a scientific publication. Table 1.1 summarizes the relation of the chapters and the individual publications.

1.5.1 Part I: Rapid Prototyping Framework for Activity Recognition

The first part is divided into three chapters. It starts out by presenting the Context Recognition Network (CRN) Toolbox in Chapter 2, a flexible rapid prototyping and runtime environment for context and activity recognition systems. Chapter 3 deals with synchronization of multi-modal data streams in heterogenous sensor networks, and Chapter 4 finally consolidates the concepts

into an integrated toolchain for creating and maintaining activity recognition systems.

1.5.2 Part II: Opportunistic Methods for Activity Recognition

The second part of this thesis is a sequence of three chapters pursuing the same topic of a novel method for opportunistic activity recognition systems. The basic idea and concept of adding a new sensor to an existing system is presented in Chapter 5 together the initial evaluation. An advanced approach based on distribution similarity is investigated in Chapter 6, and in Chapter 7 we explore two extensions of the latest approach which integrate minimal user feedback to selectively boost the method's performance.

The thesis is concluded in Chapter 8 with a summary of the key findings and outlook.

Table 1.1: Relation of thesis chapters and publications.

Chapter 2	David Bannach, Oliver Amft, and Paul Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7(2):22–31, 2008. David Bannach, Kai Kunze, Paul Lukowicz, and Oliver Amft. Distributed modular toolbox for multi-modal context recognition. In *Proceedings of the 19th International Conference on Architecture of Computing Systems*, volume 3894 of *LNCS*, pages 99–113. Springer, 2006.
Chapter 3	David Bannach, Oliver Amft, and Paul Lukowicz. Automatic event-based synchronization of multimodal data streams from wearable and ambient sensors. In *Proceedings of the 4th European conference on Smart sensing and context*, EuroSSC'09, pages 135–148, Berlin, Heidelberg, 2009. Springer-Verlag.
Chapter 4	David Bannach and Paul Lukowicz. Integrated tool chain for recording, handling, and utilizing large, multimodal context data sets for context recognition systems. In *Workshop Proceedings of the 24th International Conference on Architecture of Computing Systems 2011*, pages 311–320. VDE, 2011. David Bannach, Kai Kunze, Jens Weppner, and Paul Lukowicz. Integrated tool chain for recording and handling large, multimodal context recognition data sets. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Ubicomp '10, pages 357–358, New York, NY, USA, 2010. ACM.
Chapters 5, 6, 7	David Bannach, Bernhard Sick, and Paul Lukowicz. Automatic adaptation of mobile activity recognition systems to new sensors. In *Workshop Mobile sensing challenges, opportunities and future directions at Ubicomp 2011.*

Part I

Rapid Prototyping Framework for Activity Recognition

Chapter 2

Prototyping- and Runtime Toolbox

This chapter introduces the Context Recognition Network (CRN) Toolbox which permits fast implementation of activity- and context recognition systems. It utilizes parameterizable and reusable software components and provides a broad set of online algorithms for multi-modal sensor input, signal processing, and pattern recognition. It features mechanisms for distributed processing and support for mobile and wearable devices.

Within this chapter we present the concept of the CRN Toolbox and highlight different case studies indicating its merit in industrial projects, as educational tool for students, and processing engine in activity recognition demonstrators. Moreover, we summarize user evaluation results.

David Bannach, Oliver Amft, and Paul Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7(2):22–31, 2008.

David Bannach, Kai Kunze, Paul Lukowicz, and Oliver Amft. Distributed modular toolbox for multi-modal context recognition. In *Proceedings of the 19th International Conference on Architecture of Computing Systems*, volume 3894 of *LNCS*, pages 99–113. Springer, 2006.

Today, the development of activity recognition systems is mostly done in two phases. First the recognition method (sensor setup, feature set, classifiers, classifier parameters, fusion methods etc.) is designed. In this first phase experimental data is mostly fed offline into conventional rapid prototyping tools such as Matlab. These tools provide a rich reservoir of "off the shelve", parameterizable algorithms and visualization methods. Thus, different system variants can be tested quickly without the need for time consuming implementation work.

Unfortunately most such simulation environments are not suitable for actually running applications, especially in mobile and pervasive environments. In general, they depend on custom "engines" or libraries requiring large memory footprints and high computing power. Consequently, the implementation of activity recognition applications is mostly done in a separate, second phase. The selected algorithms are implemented in an appropriate programming language

and then distributed to specific devices. Issues that need to be addressed include sensor interfaces, synchronization of the sensor signals, and optimization for specific devices (e.g., floating-point or fixed-point calculation).

The Context Recognition Network (CRN) Toolbox (available under LGPL from `http://crnt.sf.net`) presented in this chapter has been developed to combine the two phases and permits quick construction of complex multi-modal context recognition systems, that can be immediately deployed in the targeted environment.

2.1 Related Work

The CRN Toolbox is not intended to be a general purpose pervasive middleware such as RUNES [Cos07] or sensor node operating system as TinyOS [Hil00]. Neither it is a high-level framework for rule-based automation, such as VisualRDK [Wei07]. Instead, it is a tool set specifically optimized for the implementation of multi-modal, distributed activity and context recognition systems running on POSIX operating systems. Like conventional rapid prototyping tools, it contains a collection of ready to use algorithms (e.g., signal processing, pattern classification). Unlike classic event detection in homogeneous sensor networks, as DSWare [Li04], the CRN Toolbox supports complex activity detection from heterogeneous sensors. Moreover, its implementation is particularly optimized for mobile devices. This includes the ability to execute algorithms either in floating-point or fixed-point arithmetic without recoding. With its mature functionality, we believe that the CRN Toolbox will not suffer from limited user acceptance as the Context toolkit framework [Edw03].

The CRN Toolbox contains dedicated building blocks for interfacing a broad range of sensor nodes and support for synchronization, merging, and splitting of data streams. In contrast to the PCOM model [Bec04], which focuses on contract-based spontaneous configuration, the Toolbox relies on a known network topology. Applications can be flexibly distributed among devices (including servers) just by starting the configured Toolbox runtime on the appropriate system. Another important feature is the ability to interface conventional simulation environments such as WEKA (`http://www.cs.waikato.ac.nz/~ml`). The functionality is accessible through a graphical configuration editor that allows the construction of complex applications by connecting and configuring a set of task icons corresponding to different processing steps.

The concepts utilized by the CRN Toolbox, including graphical programming, data driven computation, parameterizable libraries, and distribution are themselves not new. Such concepts have been usedj, e.g., for audio processing and data visualization [Puc88, She96], general signal processing [Sic98], or graphical programming tools [Joh97]. But neither of those frameworks and tools cover all the specific demands of activity- and context recognition systems. Specifically, they are lacking the portability between different devices that may have limited recources, outsourcing of computationally expensive tasks to less limited devices, abstraction of the algorithms from actual data type, and support for synchronizing multimodal data streams from heterogeneous sensors with unknown clocks. The contribution of the CRN Toolbox is having those concepts adapted and integrated in a way, optimal for rapid and efficient implementation of activity- and context recognition systems.

2.2 Toolbox Concept

The concept of the CRN Toolbox stems from the observation that most activity recognition systems are built from a relatively small set of algorithms. These include sliding-window signal partitioning, standard time and frequency domain features, classifiers, and time series or event-based modeling algorithms.

The key differences between systems are in sensor choice, parameterization of algorithms (e.g., size of sliding-window) and data flow. The data flow can be as simple as feeding single-dimensional sensor data to a mean filter and a classifier. This could be a configuration for recognizing sitting and standing from an upper leg accelerometer, for example. It can be as complex as fusing data from tens of heterogeneous sensors, working with different sampling frequencies, different feature computations, and even different classifiers. In such complex systems, sensor subgroups are often handled by different platforms (e.g., different mobile devices and servers for stationary sensors). The implementation must take care of the distributed computation, collection of data, and synchronization of the different data streams.

The CRN Toolbox simplifies the implementation of even such complex, distributed context recognition systems to the following three steps:

1. compiling the Toolbox for all platforms that it needs to run on,
2. selecting and configuring the algorithms and data flow for each platform,
3. starting the Toolbox on each platform with the dedicated configuration.

If algorithms should be analyzed that are not present in the current Toolbox implementation, rapid prototyping tools running on a remote server can easily be interfaced.

Figure 2.1 shows an overview of the CRN Toolbox concept. The step-by-step configuration guide in Section 2.3 presents a simple example for recognizing kitchen activities from the user's on-body sensors.

2.2.1 Reusable Components for Data Stream Processing

The basic building blocks provided by the CRN Toolbox are the reusable and parameterizable components. Conceptually, the components are active objects that operate on data streams. We refer to them as *tasks*. They encapsulate algorithms and data, and have an individual thread of execution. In essence, all tasks are executed in parallel, waiting for *data packets* to arrive at their *in-port*, process the packet's payload according to their algorithm and parameter settings and provide the modified data packet at their *out-port*. Depending on the configured data flow, subsequent tasks will receive the packet for further processing.

The Toolbox provides *reader*- and *writer* tasks for interfacing with in- and output devices, processing algorithms for data filtering and classification as well as components for splitting, merging, and synchronizing of data streams. A summary of tasks that evolved during this thesis is listed in Table 2.1. Every task has an individual number of parameters that control its operation. For example, the K-Nearest Neighbor (KNN) classifier task takes the k, a matrix of training data, and an optional step-size parameter.

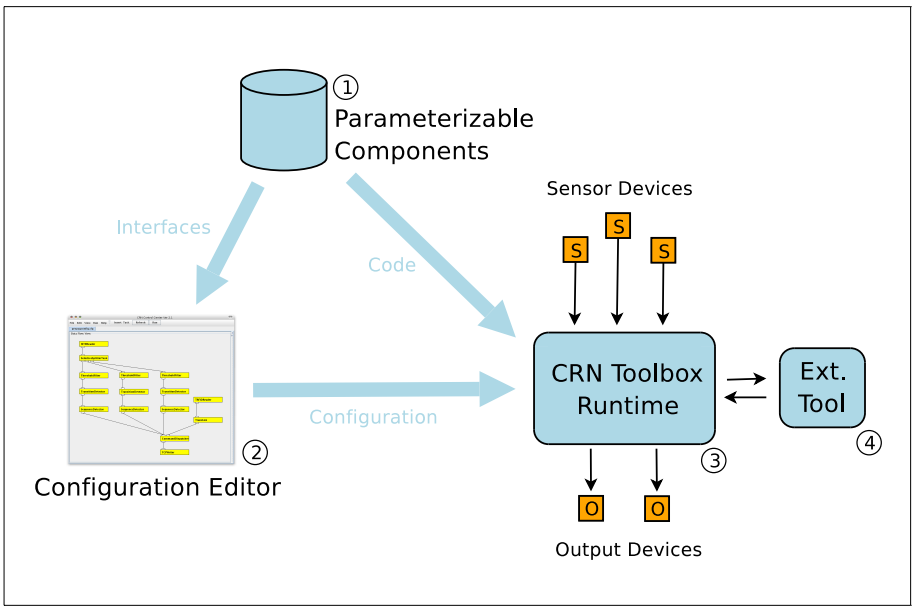


Figure 2.1: Concept of the CRN Toolbox: (1) a repository of parameterizable software components including I/O device readers and writers, filtering- and classification algorithms, components for splitting, merging, and synchronizing data streams, (2) a graphical editor for specifying data flow and configuring components, (3) the CRN Toolbox runtime environment for online executing the configured software components, (4) arbitrary external tools, communicating with the Toolbox runtime, e.g., live data stream plotting or another Toolbox (local or remote).

Each task is implemented as its own (POSIX) thread to completely decouple the scheduling from the network design and to exploit the potential of multi-core processors. Those threads are often idle while waiting for a new data packet to be available from their in-port queue and they only consume processing power when actually handling the data packet. As a consequence of the multi-threaded approach, a task must immediately delete any reference to the data packet after handing it over to one of its out-ports, because from then on it will be handled by another task which must have exclusive access. If necessary, new tasks can easily be added to the Toolbox by implementing a concrete subclass of the abstract *StreamTask* class, adding it to the build system, and rebuilding the Toolbox. They will be available for instantiation and configuration from then on.

The encapsulation in active objects and the parameterization proved essential for the reusability of the actual code. Hence, for most applications the fact that the Toolbox is implemented in C++ is insignificant, and yet they benefit from the efficient runtime.

Table 2.1: Summary of components available in the CRN Toolbox that evolved during this thesis. Detailed task descriptions are available as Doxygen web pages from the code repository. The list is constantly growing as more and more users are contributing to the project.

Generic reader tasks using decoder plug-in:

`DirectInputReader`	Passing data through function call (API)
`FileReader`	Reading from file
`KeyboardReader`	Reading keystrokes
`SerialReader`	Reading from serial device (including Bluetooth, ZigBee)
`TCPReader`	Reading from TCP sockets (client mode)
`TCPServerReader`	Reading from TCP sockets (server mode)
`UDPReader`	Reading from UDP sockets

Specific reader tasks:

`AliveEcgReader`	Alive Heart Monitor
`AntGenericReader`	ANT protocol
`ARSBReader`	ARSB (walking sensing)
`AudioFileReader`	Audio file
`AudioReader`	Audio device (soundcard)
`BAccReader`	BAcc (Bluetooth acceleration sensor)
`BTnodeReader`	BTnode
`ContextDBReader`	ContextDB (Apache CouchDB)
`CricketReader`	MIT Cricket
`DAMReader`	DAM (Data Acquisition Module)
`EmoBoardReader`	Emotion Board (ETH)
`ESLTmoteReader`	Tmote touch-less touchpad
`FBReader`	Serial port, frame-based devices
`HexamiteReader`	Hexamite (ultrasonic positioning)
`HttpInput`	HTTP (web interface)
`JennicDataReader`	Jennic sensors
`LukotronicReader`	Lukotronic (motion capturing)
`MagneticAccelReaderByte`	Magnetic sensor (location & orientation)
`MT9Reader`	Xsens MT9/MTi

continued on next page

continued from previous page

`NeoAccelReader`	Openmoko Neo accelerometer
`NMEAReader`	NMEA protocol (GPS)
`OmTouchPadReader`	Openmoko Freerunner accelerometer
`ParticleReader`	TecO Particles
`PhilipsReader`	MyHeart protocol
`SCIPIOReader`	SCIPIO (sensor glove)
`SensFloorReader`	Future Shape SensFloor
`SinGenerator`	Sinus generator
`SuuntoReader`	Suunto wireless sensors (HRM, FootPod, GPSPod, etc.)
`TFSRReader`	Tmote force sensing resistors
`TMagReader`	Tmote magnetic distance
`TMSIFReader`	TMSI fiber protocol
`TRFIDReader`	Tmote RFID
`UTicker`	Ticker (packet generator)
`WiiReader`	Wiimote
`XbusReader2`	Xsens Xbus (up to 12 Xsens MTi units)

Channel reordering tasks:

`ChannelSelect`	Dynamic selection of a channel
`ChannelSelection`	Select channels by name
`ConstMerger`	Merging multiple streams at constant data rate
`Demultiplexer`	Demultiplexing interleaved streams
`DownsamplingMerger`	Merging two streams, downsampling
`ResamplingSyncMerger`	Merging two streams, choice of sampling methods
`SelectiveSplitterTask`	Select channels by index
`SimpleMerger`	Merging multiple streams, up-sampling (last)
`SyncMerger`	Merging two streams by timestamp

Filtering tasks:

`Distance2Position`	Simple position calculation
`Distance2Position3D`	3D position calculation from distance measurements

continued on next page

continued from previous page

`Downsample`	Let only fraction of data packets pass
`Einsnorm`	Sum of absolute values
`FilterTask`	Sliding-window filter task using filter plug-ins (see below)
`FindBlob`	Find contiguous blobs
`FindMax`	Find maximal channel
`FreqScanner`	Determine rate of incoming data
`NeoDynamicCalibrationFilter`	Calibrating Openmoko acceleration data
`OffSet`	Add offset vector
`PeakFinder`	Find peaks
`ProximityEstimator`	Discrete proximity from Particle distance data
`RFIDTranslator`	Translate RFIDs from ByteBufValue to IntValue
`Rotate3D`	Rotate 3D vectors
`SegmentFilter`	Apply filter to whole segment
`SensFloorCamTrigger`	Transform SensFloor data into heartbeat stream
`Slider`	Convert touch-less touchpad data into "slider" value $[0, 1]$
`StrMatcher`	String matching
`SwiftSeg`	Approximate data stream using polynomials
`TrailingEdge`	Detect trailing edges
`TransitionDetector`	Only let data packets pass if not identical to previous
`VecLen`	Calculate vector length

Filter plug-ins:

`AbsFilter`	Absolute value
`ASEFilter`*	ASE (average signal energy)
`BERFilter`*	BER (band energy ratio)
`BWFilter`*	BW (bandwidth)
`CGFilter`*	CG (center of gravity)
`Cutoff`	Cut-off
`EntropyFilter`*	Entropy
`FFT`	FFT (fast Fourier transform)
`FFTW`	FFT (using libfftw)
`FluctuationFilter`*	fluctuation

continued on next page

continued from previous page

`IIRFilter`	IIR (infinite impulse response)
`MaxFilter`	Max
`MeanFilter`	Mean
`MedianFilter`	Median
`MinFilter`	Min
`Quat2rotmatFilter`	Convert quaternions to rotation matrix
`ScaleFilter`	Scale
`SimpleHighpassFilter`	high-pass
`SimpleLowpassFilter`	low-pass
`SlopeFilter`	slope
`SumFilter`	Sum
`SFRFilter`*	SFR (spectral rolloff frequency)
`ThresholdFilter`	Threshold
`VarFilter`	Variance

Classification tasks:

`CarPositionClassifier`	Simple position classification
`DecisionTree`	Decision tree classifier
`ESParser`	Earley-Stolcke parser (PCFG)
`Hexamite2D`	Simple location-based classification
`HMM`	Hidden Markov models
`KNN`	K-Nearest Neighbor
`ParserTask`	PCFG (probabilistic context-free grammars) baseclass
`RangeChecker`	Very simple range checker
`SequenceDetector`	Spotting known sequences
`ShakeDetector`	Detecting shake gesture
`SlidingGestures`	Detecting “sliding” gestures

Miscellaneous tasks:

`CmdDispatcher`	Specific command dispatcher
`ContextLogic`	Specific data conversions for hospital scenario

continued on next page

continued from previous page

`ContextMonitor`	Monitoring context information
`Correlator`	Empirical correlation coefficient of two streams
`Heartbeat`	Audible Heartbeat
`LabelBasedSegmenter`	Segmenting data based on label
`MagneticDataRefiner`	Approx. distance & orientation from magnetic sensor
`Mapping`	Map vector to scalar
`Nothing`	Pass-through (debugging)
`ProximityWatcher`	Translating output from `ProximityEstimator`
`RFIDWatcher`	Translating output from `M1miniReader`
`ScrollCounter`	Specific gesture repeater
`SetStreamDescription`	Set stream description on passed-through packets
`Similarity`	Similarity search
`SliderCommandsTmoteTouch`	Send discrete slider/scroll commands
`StoryTracker`	Extracting meta-data for story file
`Superpacket2SyncEvent`	Convert superpackets to SyncEvents
`SWAB`	Explicit time series segmentation
`SyncBench`	Stream synchronization tool
`Synchronizer`	Event-based synchronizer
`SyncManager`	Synchronization event manager
`Tapper`	Extracts seqNr fields from superpackets
`UDPHeartbeat`	Broadcasting stream metadata via UDP
`Valve`	Valve-like control of data flow

Writer tasks:

`AudioFileWriter`	Audio file
`AudioWriter`	Audio device (soundcard)
`ConsoleWriter`	Console
`CouchWriter`	ContextDB (Apache CouchDB)
`DirectOutputWriter`	Passing data to callback/polling function (API)
`DisplayGraph`	X11 desktop graph
`DisplayImage`	X11 desktop image
`FileWriter`	File

continued on next page

continued from previous page

`GizmodWriter`	Gizmo daemon
`JMSWriter`	ActiveMQ messaging service
`Nirvana`	Quietly discard data
`SerialWriter`	Serial port
`Sink`	Sink
`TCPClientWriter`	TCP (as client)
`TCPWriter`	TCP (as server)
`UDPWriter`	UDP broadcast/multicast

Encoder plug-ins:

`ARFFEncoder`	ARFF format (WEKA)
`BinaryEncoder`	Binary to Int encoding
`CmdEncoder`	Command strings encoding (printf-like)
`CouchEncoder`	JSON encoder for Apache CouchDB
`IntLinesEncoder`	Integer lines encoding
`IntToTextEncoder`	Map EnumValues to text strings
`JsonEncoder`	JSON encoder
`PlottingEncoder`	ASCII-art plotting
`SuperPacketEncoder`	Timestamped lines encoder for superpackets
`TextLabelEncoder`	Output EnumValues as text strings
`TimestampedLinesEncoder`	Timestamped lines encoder

Decoder plug-ins:

`ARFFDecoder`	ARFF format (WEKA)
`ASCIIDecoder`	ASCII lines
`CxdbDecoder`	ContextDB (JSON)
`FloatLinesDecoder`	Lines of floating-point values
`IntLinesDecoder`	Lines of integer values
`M1miniDecoder`	ZigBee-M1mini-RFID format
`NMEADecoder`	NMEA (GPS)
`StringLinesDecoder`	Words to StringValues

2.2.2 Runtime Environment and Data Flow Control

The Toolbox runtime provides the vital environment for tasks to operate. It handles dynamic creation and configuration of tasks as well as configuration of the data flow.

For parameter handling the Toolbox utilizes the JavaScript Object Notation (JSON) format [Cro06] with an object loader in the "get instance by name" style. Thus, the Toolbox can be configured at runtime by text-based configuration files that define the settings for tasks and the data flow needed by the application.

The data flow between tasks is specified through directed *connections* from out-ports to in-ports. Each data packet transmitted along these connections contains data entities belonging to one time instant. The payload of a packet is organized as vector of values from an abstract *Value* data type. Moreover, the packets contain a timestamp and sequence number. For combining multiple streams the Toolbox provides *merger* tasks. Mergers combine the payload of packets from separate in-ports and synchronize data streams with different sampling rate.

Data packets are passed by pointer reference along the internal connections through the task network. Packets are cloned only if more than one receiver is connected to the same out-port. This implementation of the runtime core ensures high packet-processing performance. Moreover, we preserve processing performance by providing operations to the task-developer that inherently modify data objects, such as the operator "+=" does, instead of allocating new result objects for each operation (e.g., like $v = v_0 + v_1$ would).

By implementing POSIX threads, the scheduling of individual components is subject to the operating system on which the Toolbox is running. The Toolbox runtime itself only needs to ensure that no deadlocks can occur. Accidental loops in the data flow, introduced by malicious configurations, could force the runtime to crash. But multiple connections to the same in-port are allowed only in special cases and checks for such configurations could be aded if necessary. Feedback loops for controlling the operation of tasks are intended to be fed into separate in-ports which are designated by receiving tasks.

With the CRN Toolbox, the same runtime is used during development of new activity recognition systems (e.g., during recording of sensor data for training new classifiers) and for actually running the new systems on the target device(s). Usually the configuration only need to be slightly modified between the two phases (e.g., exchanging the recording task by a classifier, see Section 2.3). The actual training of the classifier is depending on the individual implementation of the classification task. The decision tree classifier, as an example, is trained and evaluated outside the Toolbox using WEKA and the resulting tree structure is imported into the Toolbox configuration. Other tasks could adopt training and evaluation modes to completely cover this process within the Toolbox runtime.

2.2.3 Synchronizing Independent Data Streams

Synchronization of the data streams coming from different sensors is a major issue in multimodal activity recognition. When several independent sensors are used, their data streams should be synchronized to a common starting point.

Ideally, every sensor would provide samples with an exact clock frequency,

global timestamp, or would support an external clock. However, in reality many simple devices send data with an internally generated rate. Consequently, when several independent sensors are used, their data streams should be synchronized to a common starting point. Moreover, synchronization should be repeated during runtime as the sampling rates may jitter, e.g., due to communication delays.

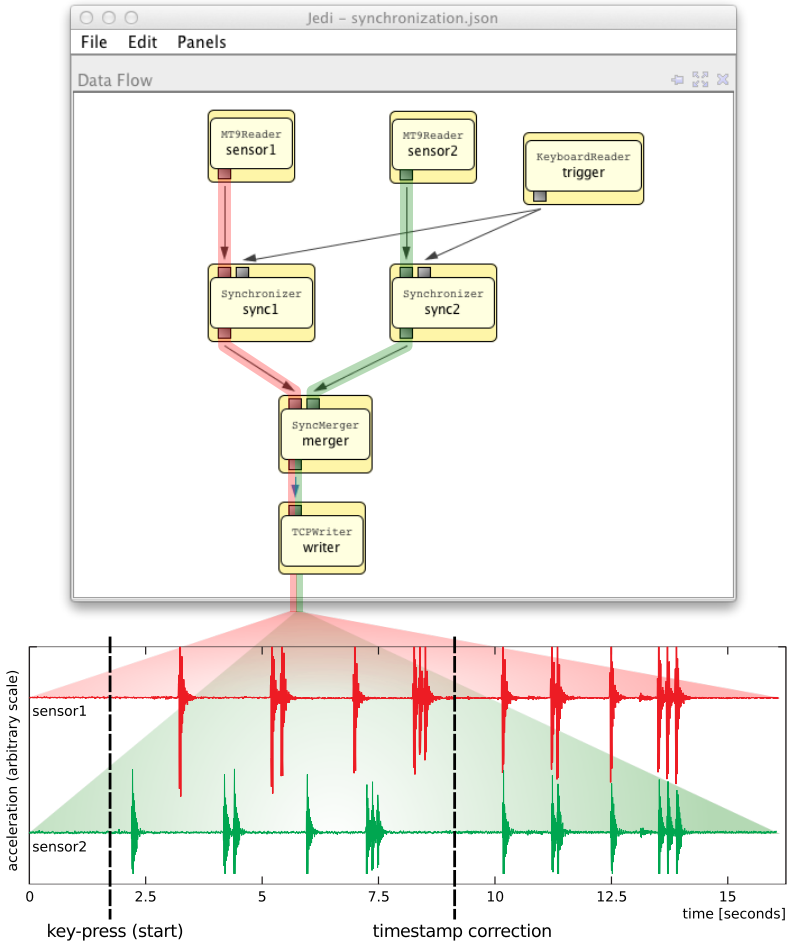


Figure 2.2: CRN Toolbox graphical configuration editor with a synchronization setup for two acceleration sensors. The waveforms demonstrates the data alignment achieved at an event detected by the `Synchronizer`s.

A feasible concept for this type of synchronization is aligning streams on events occurring simultaneously in the data from all involved sensors, e.g., a user's jumping up with a set of on-body acceleration sensors. The details of this synchronization method are presented in Chapter 3. It is implemented in the `Synchronizer` and `SyncMerger` tasks. Figure 2.2 depicts the solution for the example of two acceleration sensors (Xsens MT9). A characteristic high acceleration amplitude was inserted by the jump. The `Synchronizer` tasks detect the peaks caused by these events and adjust data packet timestamps accordingly.

The `SyncMerger` combines the data streams by aligning the timestamps. The `Synchronizer` tasks are manually activated, e.g., by the user's input through a `KeyboardReader`, to limit the alignment phases to controlled time frames. The accelerometer signals plotted in Figure 2.2 are clearly out of sync during the first ten seconds, yet after the alignment phase ended (timestamp correction) the signals are aligned correctly. Our analysis of the method shows that an alignment of 0.3 seconds and better can be achieved in a real-life scenario (see evaluation section in Chapter 3).

2.2.4 Readers: Sensor Hardware Encapsulation

In the CRN Toolbox sensor interfaces are implemented as tasks without in-ports, called *reader* tasks. They instantiate new data packets for data samples acquired from sensors (or other sources) and provide them on their out-port. Our architecture supports various reader implementations that can capture different sensors or other sources, such as databases, web pages, application outputs, or data files.

For activity annotation, we implemented a keyboard-reader to perform online labeling of data. This reader proved very helpful, since the labeling can be stored aligned with the raw data for later evaluation.

2.2.5 Writers: Communication for Distributed Processing

The key to distributed execution and use of external tools, are *writer* tasks. They forward data received at their in-port to external interfaces, e.g., files, displays, or network connections. For the latter we use `TCPWriter` and `TCPReader` tasks to communicate via TCP/IP sockets. Data packets are transmitted on the channel in a serialized form. The serialization is obtained from an `Encoder` plug-in in the `TCPWriter` task. Similarly, the `TCPReader` uses a `Decoder` plug-in for de-serialization. Thus, two CRN Toolboxes running independently, e.g., on different hosts, can collaborate using the writer-reader communication.

Using this mechanism the Toolbox can link to arbitrary programs based on compatible interfaces. Currently, such interfaces exist for Matlab and WEKA. Both are used for data visualization and pattern recognition in experiments and demonstrators.

2.2.6 Graphical Tools for Configuration

The rapid prototyping capabilities of the Toolbox raised our needs for an easy and quick configuration editor. The Toolbox configuration files are simple to create and modify as the JSON format is supported by many editors. Still, an integrated editor that is aware of the Toolbox specific schemes and the available components can be convenient.

We provide a Java based, modular, graphical JSON editor for the CRN Toolbox (JEDI). Users can simply drag tasks from a library into the JSON document, set their parameters, and interconnect them with just a few mouse clicks in the data-flow view. The configuration is validated on the fly and possible problems are highlighted. Configurations are saved to JSON files which the CRN Toolbox may load. Future modules might directly interface running

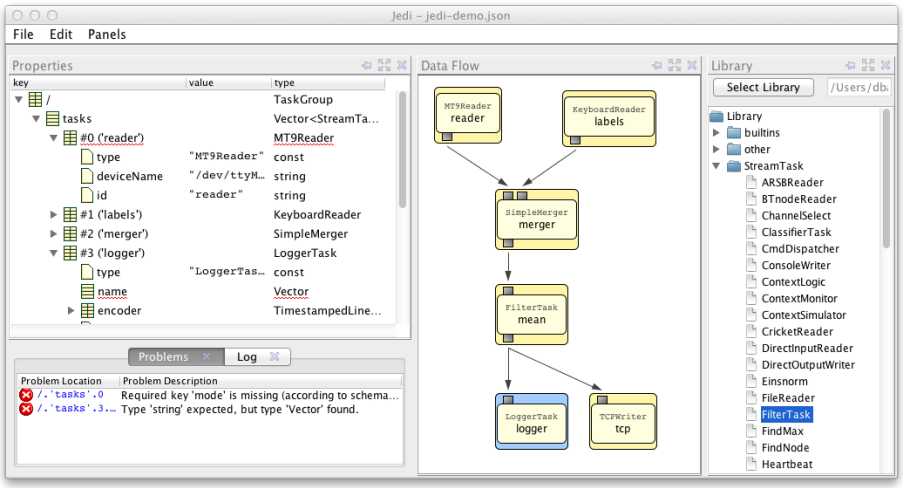


Figure 2.3: Screenshot of the graphical Toolbox configuration editor JEDI.

Toolbox instances. Figure 2.3 shows a screenshot of the graphical editor with two problems spotted in the configuration of the reader and logger tasks.

2.3 Step-By-Step Guide: How to Cook

With the CRN Toolbox building of activity recognition applications becomes really easy. For example, it takes only five steps to implement your own kitchen activity recognition including the classifier training. You do not have to write additional code. This section briefly explains all steps that are necessary to build the gesture recognition system for kitchen guide applications using the CRN Toolbox.

The ingredients are: a motion sensor mounted on a glove, a wearable computer or "kitchen PC", and, of course, the CRN Toolbox. In this guide we use the MT9 sensor from Xsens featuring accelerometers, gyroscopes, and magnetic field sensors, all on three axis. From the Toolbox we utilize a reader task for acquiring data from the sensor, a couple of filtering tasks for computing features, the KNN classifier task, and a task for displaying classification results on a screen. Typical activities that can be recognized and discriminated by setup include stirring, whisking, cutting bread, slicing onions, and wiping with a cloth.

1. Using the graphical configuration editor, **create a configuration for recording training data** as shown in Figure 2.4a. Begin by adding the `MT9Reader`, to acquire data from the MT9 sensor at 100 Hz and provide all nine channels on its out-port. With the `SelectiveSplitterTask` pick out the channels of interest and send them through a set of filters: `MeanFilter` and `VarFilter` (variance), both operating on sliding windows. Set the window size to 100 (1 sec). With the `SimpleMerger` combine the two data streams again and add the output of a `KeyboardReader` task. This task is used to annotate the recording by keystrokes. Finally, add a `LoggerTask` for writing the resultant data streams into a file.

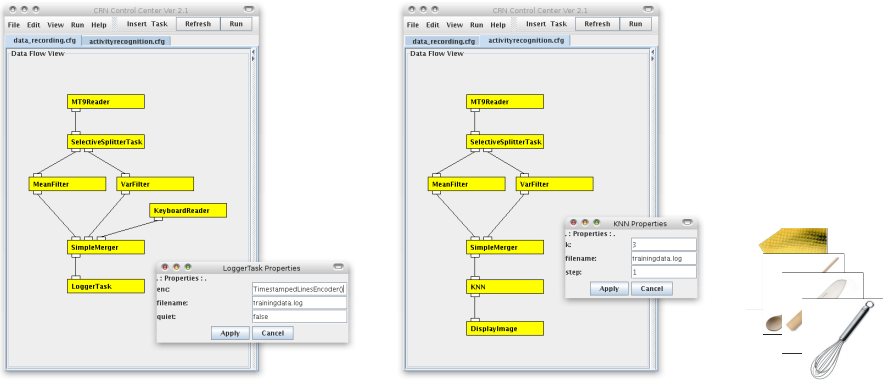
2. Select an annotation key for training each activity. Connect the sensor and start the Toolbox with the created configuration. Then, wearing the sensor glove, **perform each activity for about 30 seconds**. At the beginning of each activity press its selected annotation key.

3. **Review the recorded training data** in the log file and reduce it to about 100 samples per activity class. The class label is indicated by the number in the last column.

4. Now, modify the first configuration to **include the classifier and the output task** (see Figure 2.4b). You may remove the `KeyboardReader`, because from now on the classifier will do the "annotation". Specify the filename of the training data in the properties of the `KNN` task. Attach the `DisplayImage` task to the `KNN` and specify a pictures that should be displayed on the screen for each recognized activity category.

5. Start the Toolbox with the new configuration. Now you can work in the kitchen as you wish and **let the Toolbox track your activities** or, even better, feed the results into a context-aware cookbook. Bon appétit!

To improve the system you may add more sensing modalities such as location, select useful features, and use more sophisticated recognition algorithms.



(a) (b) (c)

Figure 2.4: Configurations for the kitchen activity recognition. (a) recording of training data; (b) on-line classification and display of result; (c) example output of the classification using `DisplayImage`.

2.4 Case Studies

The vitality of a framework such as the CRN Toolbox stems from its continuous development and deployment in various projects. The showcase of applications in industrial projects, student classes, and demonstrators (see Table 2.2) not only highlights its maturity and widespread use but also serves as evaluation of the fundamental concepts. During such projects the Toolbox has been successfully deployed on different platforms, including

- Linux (arm32, i386, amd64, x86-64),
- Mac OS X (i386, x86-64),
- iOS (iPhone, iPod touch, iPad),
- Android (smartphones), and
- Cygwin (i386).

Here, we depict three case studies from different areas, outlining the successful utilization of the CRN Toolbox.

Table 2.2: Summary of major projects using the CRN Toolbox. For student class projects approximate lines of code (LoC) and components count are shown.

Project description	**Utilization of the CRN Toolbox**
Industrial projects	
WearIT@Work supporting hospital information flow [Ada08, Ban06]: gesture controlled access to patient's document using wrist worn motion sensor. Using RFID for patient identification.	Data capturing, gesture recognition, control of hospital's document browser, running on QBIC (Linux/arm32). Several demonstrators and test system built. A hospital trial was conducted.
WearIT@Work production support: activity recognition of car assembly and maintenance [Sti06a]. The project utilizes inertial motion and indoor location sensors.	Recording multimodal sensor data: Xsens, Hexamite, Ultrasound, muscle force; various demonstrators. Platform: Linux/i386.
MonAMI dynamic monitoring services	Dynamic re-configuration of the Toolbox depending on available sensors and registered services. Platforms: Linux/i386 and Linux/arm32.
MyHeart walking habits: online classification of walking activities and intensities to support active lifestyle and improve fitness.	Acquisition of heart rate, acceleration, and air pressure; classification; streaming results to a mobile phone and professional coaching center, running on QBIC (Linux/arm32).
NESD location tracking: GPS-based local map visualization	GPS position logging (NMEA protocol) and conversion for dynamic map display, forwarding to central mission server, running on QBIC (Linux/arm32).

continued on next page

continued from previous page

Student classes and -projects	
ISWC 2006 tutorial "Hands on Activity Context Recognition": building a gesture recognition system for controlling a simulated car parking game with real waving gestures, 12 participants.	Testing of algorithms and gesture types using simulated data streams from a motion sensor glove. Components: 9, LoC: 7000. Platform: Linux/amd64.
Number entering game: entering binary digits using a motion sensor only, practical exercise and competition for ambient intelligence lecture, 15 students in 5th semester.	Understanding of algorithms, modularization and interfacing to sensor data. Components: 7, LoC: 2000. Platform: Linux/i386.
Location estimation and activity recognition: ultrasonic- and motion sensors practical exercise for ambient intelligence lecture, 12 students in 5th semester.	Understanding of algorithms and challenges of location tracking. Components: 7, LoC: 2000. Platform: Linux/i386.
Interactive World: software project to implement gesture control for games (Pong, Tetris), 3 days, 19 students in 4th semester.	Implementation of a TCP reader task, utilization of KNN classifier (gesture recognition). Components: 5, LoC: 1200. Platform: Linux/i386.
Activity monitoring: training a classifier to recognize human activities (sitting, standing, walking, running) and visualizing results, 10 students in 4th semester.	Learn the concepts and operation of a classifier, implement and testing. Components: 7, LoC: 2000. Platform: Linux/i386.
Demonstrators	
Parking Game: controlling a virtual driver and car with real hand gestures in a parking game [Ban07].	Capturing glove-based inertial sensor data, gesture spotting using explicit segmentation, gesture event search, and fusion steps; controlling the game visualization engine. Platform: Linux/amd64.
Parsing of dietary activity events using probabilistic context-free grammars (PCFGs): inference of food intake cycles from activities [Amf07].	Simulation of activity event input, PCFG parsing, reporting of results. Platform: Linux/amd64.
Hammering-screwdriving demo: recognizing assembly activities (hammering, screwdriving, sanding, vising) with a motion sensor in a glove.	Xsens motion sensor capturing, classification of activities, display of recognition result on screen and wireless connection, running on QBIC (Linux/arm32).

2.4.1 Supporting Information Flow in Hospitals

Together with clinical partners in the EU-sponsored WearIT@Work [Luk07] project we developed a solution to improve information flow for the doctor's ward [Ada08, Ban06]. At the ward round, doctors decide on further treatment of patient under tight time limitations. Access to patient documents right at the bedside would allow the doctor to make decisions based on all available patient information. Notebooks or PCs are impractical for this task, since their operation is time-consuming, distracting, and involves touching non-sterilized devices while in contact with patients.

Our wearable solution simplified the document access. When the doctor comes to the patient's bed, the bedside monitor automatically shows up a document list for that patient. The doctor could then browse these documents by pointing at the monitor and swivel the forearm. The system worn by the doctor consists of the Q-Belt Integrated Computer (QBIC) [Amf04], running the CRN Toolbox, an Xsens motion sensor, and an RFID reader. The QBIC is worn as a belt, the latter two are integrated into an armband which is worn at the wrist. The patient's name tag is equipped with an RFID tag. At each patient's bed there is a bedside monitor to display documents from the hospital's information system during the ward and to serve as an entertainment system otherwise.

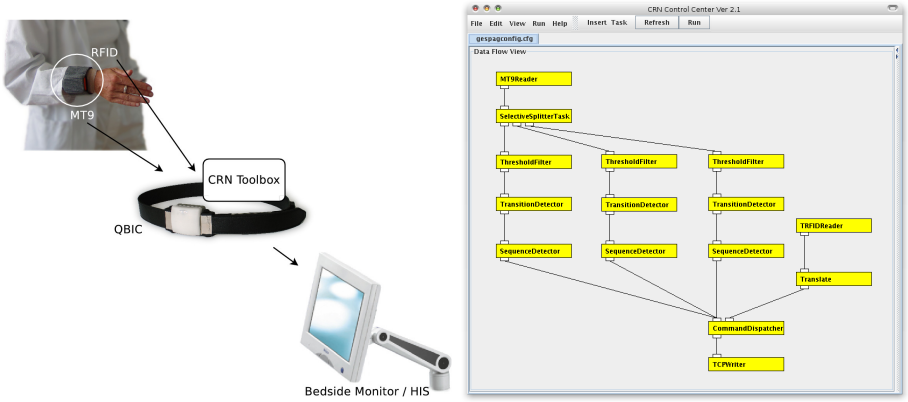


Figure 2.5: Hospital information support system setup and CRN Toolbox configuration.

In the implementation we used a set of three tasks to process each gyroscope axis of the motion sensor. The set contained threshold detection and sequence matching tasks (`ThresholdFilter`, `TransitionDetector`, and `SequenceDetector`). This algorithm can detect forearm gesture sequences such as swivel left, then right (open document command). The `CommandDispatcher` acts as gateway, forwarding the commands only in active state (controlled by an activation gesture). This task also consumes the patient identification from RFID. Finally, the commands are transmitted (`TCPClientWriter`) to the document browser of the hospital's information system. The physical setup and the Toolbox configuration are shown in Fig. 2.5.

The complete setup was tested in a two weeks real-life trial with doctors in an Austrian hospital. While the system performed well for the staff that was involved in the early prototyping phase, doctors which were new to the

system had issues in using it, mainly because the required gestures were not intuitive to them. Another reason were stability issues of the Bluetooth and WIFI connections. In a further iteration of the project, we therefore replaced the wrist mounted motion sensor with a strip of capacitive sensors mounted in the doctor's coat [Che08b, Che08a] which would allow for simpler sliding and tapping gestures. Also, the belt computer was replaced with an infrastructure PC. The switch to new, custom sensors and gestures only involved coding of a new reader- and a gesture recognition task for the Toolbox. Aside from compiling the Toolbox for the new platform, only the configuration script (JSON) had to be adapted to replace the reader- and recognition components.

2.4.2 Monitoring Walking Habits

Together with industrial partners in the EU-sponsored MyHeart project (IST-2002-507816) we investigated new approaches for preventing cardiovascular diseases and maintaining low disease risks. Since many daily activities involve walking, we developed a walking habits monitor that supports active walking and could track activity intensity.

In our implementation, the QBIC served as a central data acquisition and processing hub, running the CRN Toolbox. The user's activity was monitored with a custom sensing unit containing acceleration and air pressure sensors attached to the belt. Additionally, a heart rate chest belt was connected via Bluetooth. Based on features from the belt sensor unit we classified walking straight, -up, -down, and idle as well as using the elevator up or down. The result along with the heart rate was forwarded to a mobile phone.

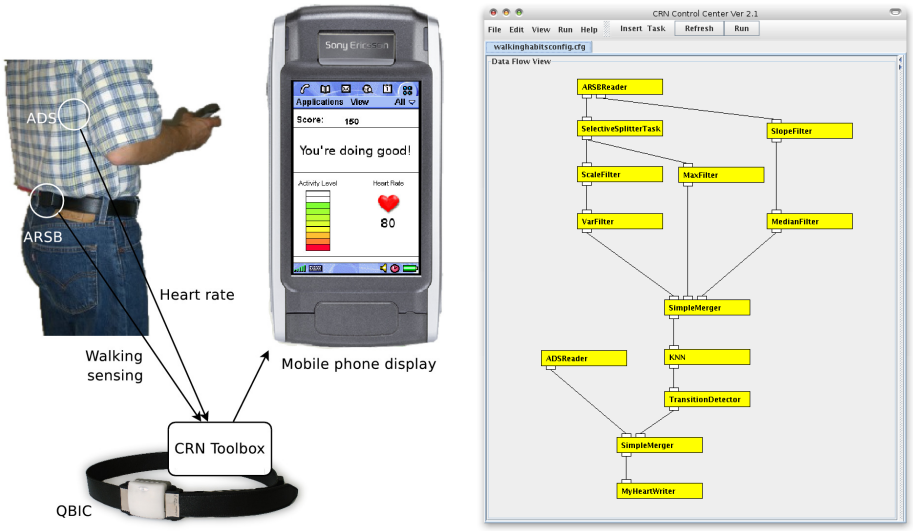


Figure 2.6: Monitoring walking habits phone visualization and CRN Toolbox configuration.

Figure 2.6 shows the final Toolbox configuration. This project required readers to capture data from the belt sensor unit (`ARSBReader`) and heart rate belt (`ADSReader`), several features filters, a classifier (`KNN`), and a writer to communicate to the mobile phone application (`MyHeartWriter`).

The visualizations on the phone showed activity level, heart rate and provided recommendations based on detected activities. In our ongoing work we use further sensors at the limbs to capture diverse activities.

2.4.3 A Mixed-Reality Car Parking Game

We designed a car parking game to explore the use of wearable systems in computer games [Ban07].

The game plot features the player helping a virtual driver to fit the latter's virtual car into a parking lot. The player does so by weaving hand and arm gestures while facing a virtual scene at the roadside where a parking spot is available between other vehicles. Figure 2.7 shows a screenshot of the scene. The driver and car's behavior are simulated and follow the gesture commands. The goal is to perform this guiding task as fast and safely as possible, in particular avoiding collisions with other cars and obstacles.

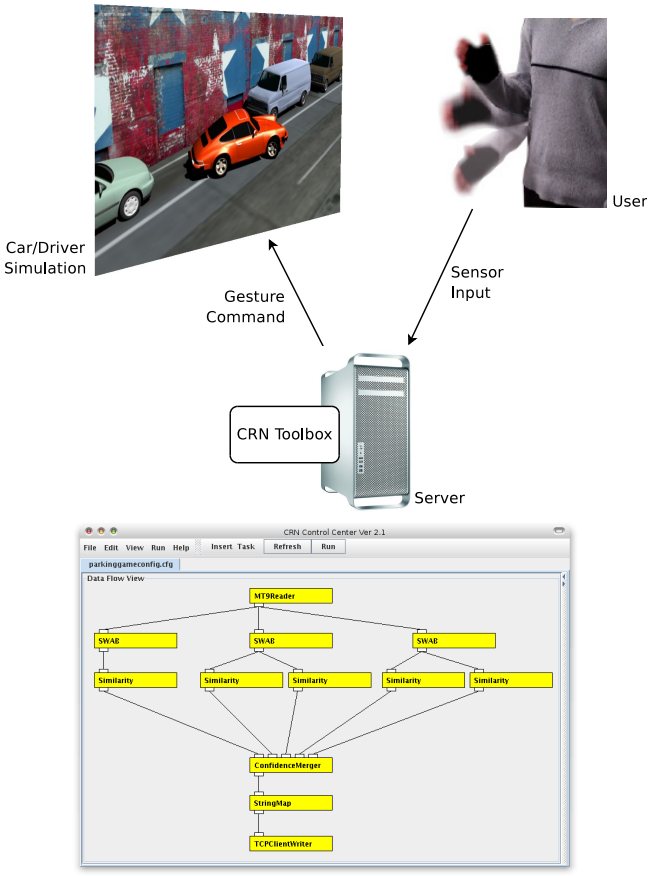


Figure 2.7: Scene of the parking game and CRN Toolbox configuration.

In this application the CRN Toolbox performs the recognition of gestures from the player's glove. Its task is to the detect five gesture commands in the continuous data stream from the glove: forwards, backwards, turn left, turn right, and stop. We used acceleration and gyroscope sensors in three axes from

an Xsens MT9 unit attached to the glove. The gesture spotting procedure utilizes an explicit time series segmentation algorithm (`SWAB`), followed by a class-specific feature similarity search (`Similarity`). Finally, the individual gestures are fused (`ConfidenceMerger`). The retrieved gestures are mapped (`StringMap`) to game commands and transmitted (`TCPWriter`) to the game simulation and graphics engine.

The game was used as demonstrator for tutorials and student courses. We built recognition models for 16 different gestures. Hence, every player could customize the system, by selecting five gestures according to individual preferences. This was easily implemented by exchanging configuration files for the recognition tasks.

2.5 User Evaluation

Right from its very first days, the CRN Toolbox has been a community project. The very positive user feedback and the growing number of tasks indicate that our approach is well perceived. However, a thorough quantitative evaluation of middleware and programming tools such as the CRN Toolbox is hard [Edw03].

While we have not yet performed a controlled assessment, we do have some empirical, in some cases even quantitative results that support our view on its usefulness.

2.5.1 Experience with Students

As presented in Table 2.2 the Toolbox has been widely used in student classes with over 60 students having worked with it.

A class of 19 fourth semester computer science students implemented an application with the Toolbox to control the computer games Pong and Tetris by shaking a motion sensor in different directions. A typical solution for recognizing these gestures consists of five Toolbox components (approx. ∼1200 lines of code used, but just ∼40 lines of configuration):

- acquire data from sensor
- apply filters (mean, variance)
- classify gestures, using KNN algorithm
- send result via TCP
- manage data flow

The exercise also included the implementation of a new Toolbox task for reading from TCP sockets. The students were just Java beginners and never programmed C++ before, yet with the Toolbox all students were able to solve the recognition problem within 20 hours. Four of them also completed the Java game in that time.

2.5.2 Evaluation of researchers

The CRN Toolbox was used in a activity recognition tutorial at ISWC[1] 2006. The 12 participants were asked to rate their impressions after having worked with it for 4 hours. Ten completed the tutorial feedback form. On average they rated themselves as advanced software programmers with some knowledge of C++, but little experience in context recognition. They reported average durations of 10 minutes (max. 30 minutes) to understand the four tasks of the tutorial, 15 minutes (max. 30 minutes) to implement and run solutions with the Toolbox, and 20 minutes to debug their configuration if needed.

We received many positive comments, such as "good and fast platform for application development", "one can click filters together", "easy to understand, easy to use" and "you don't have to reinvent the wheel". Critics addressed missing documentation materials. Our current work addresses this issue by using automatic documentation tools (Doxygen) and web platforms more intensively.

2.6 Conclusion

The CRN Toolbox was developed to ease the process of building activity recognition systems. We believe that its quick adoption by researchers is due to intuitive design of reusable components and data flow mechanism. The spectrum of implemented solutions indicates that our approach is viable in the diverse environments of wearable and server-based applications. Even students which were new to the theme managed to implement recognition solutions during class times.

As a framework, the CRN Toolbox introduces processing overhead. A prominent aspect in our design is the between-task communication, required in most useful configurations. It relies on a common packet format to exchange all media types. Besides the payload, a packet contains timestamp, sequence number, and a payload pointer, totally 16 Bytes, independent of the payload size. For a typical scenario, as the hospital support system, raw sensor-data packets have the highest transmission rate. In this example, an `MT9Reader` acquired a 9-channel Xsens MT9, requiring 36 Bytes for one sample. Each sample was sent separately, which amounts to 44% overhead. For packet rates above 100 Hz, such as audio, the effective overhead is reduced by transferring multiple samples in one packet.

Concerning the Toolbox's implementation, two areas remain open for future work: One is the addition of other classification methods and filtering tasks to support an even larger spectrum of applications. Clearly, this is what the Toolbox is designed for – new components may be added any time and enable the community re-used them in their projects. Another area is the implementation of an efficient memory management for allocation of data packet and value objects to optimize system performance for packet rates well beyond 100 Hz as required, e.g., for audio signals. The default memory allocation algorithms can be a bottleneck there. Yet, the Toolbox's concept encourages a pool based memory management.

The CRN Toolbox, as presented in this chapter, provides all means necessary to build context- and activity recognition systems which rely on a statically

[1] http://www.iswc.net/iswc06/

defined set of sensors and tasks. For the vision of opportunistic activity recognition systems, however, which cope with the dynamic availability of sensor systems and computing platforms, and which opportunistically take benefit of sensors that were not available during design time, the Toolbox can only serve for their implementation at a base level. Beyond the level of configurable tasks, such opportunistic systems require a higher level of abstraction, e.g., "services" with dynamic binding. We introduce one such approach that serves the needs of opportunistic activity recognition systems in Chapter 4 when presenting an integrated tool chain built around the CRN Toolbox.

A further point for future work is the back propagation of messages through the processing network for the purpose of resource management. Whenever a resource is scarce or costly, e.g., memory on a mobile device or a power consuming sensor, it would be favorable to release it as soon as it is no longer needed such that other processes can use it or the system can save power. This decision is usually made at a higher abstraction level, and thus raises the demand to propagate messages back to the components that are in control of the resources.

When combining data streams from different sources, the Toolbox relies on the timestamps within each stream in order to align them properly. It is therefore necessary to synchronize the "clocks" of the data streams before merging them. In the next chapter we discuss an approach for automatic synchronization of data streams based on events detected in both signals.

Chapter 3

Event Based Synchronization

A major challenge in using multi-modal, distributed sensor systems for activity recognition is to maintain a temporal synchronization between individually recorded data streams. A common approach is to use well defined 'synchronization actions' performed by the user to generate, easily identifiable pattern events in all recorded data streams. The events are then used to manually align data streams. This chapter proposes an automatic method for this synchronization.

We demonstrate that synchronization actions can be automatically identified and used for stream synchronization across widely different sensors such as acceleration, sound, force, and a motion tracking system. We describe fundamental properties and bounds of our event-based synchronization approach. In particular, we show that the event timing relation is transitive for sensor groups with shared members. We analyzed our synchronization approach in three studies. For a large dataset of 5 users and totally 308 data stream minutes we achieved a synchronization error of maximal 0.3 s for more than 80% of the stream.

David Bannach, Oliver Amft, and Paul Lukowicz. Automatic event-based synchronization of multimodal data streams from wearable and ambient sensors. In *Proceedings of the 4th European conference on Smart sensing and context*, EuroSSC'09, pages 135–148, Berlin, Heidelberg, 2009. Springer-Verlag.

Multi-modal, distributed sensor systems have been proposed for a variety of wearable and pervasive computing applications. A major practical concern to use such systems is how to temporally synchronize data streams between individual sensors. In particular, methods that rely on sensor fusion (such as computing joint features from several sensors or jointly feeding a classifier) require that sensor signals are well synchronized.

The synchronization problem is a well known and widely studied distributed systems issue [Cou01]. Without precautions, clocks of physically separate processors will run asynchronously [Sun05]. A receiving node observes this as skew

and drift in the incoming sensor data stream. Furthermore, networked nodes may startup or resume operation independently. This can causes offsets at stream receivers, similar to not handled data interrupts in wireless links.

Much effort has been devoted to lightweight clock synchronization methods for wireless sensor networks (see related work). In general these methods rely on elaborate message passing and time distribution protocols among the network nodes. The approach presented in this chapter does not aim to compete with, or improve upon this work. While in theory, it may be assumed that a sensor node could receive messages and run such synchronization protocols, in practice many on-body and pervasive sensing setups do not have appropriate capabilities. For one, leaving out the receiving capability reduces power consumption and makes node design simpler. Secondly, systems are often built using nodes from different manufacturers with insufficient built-in synchronization support. It is not common that commercial nodes provide access to node-internal communication routines.

Consequently, we look at a specific problem variation that occurs frequently in wearable and pervasive systems: dedicated, low-level sensor nodes merely stream data in a hierarchical topology and are not able to communicate with each other (e.g., because of incompatibility or just lack of resources). This means that no message passing protocols can be run between the nodes. Instead, the synchronization can rely solely on the content of the data stream.

In theory, it may be assumed that a sensor node could receive messages and run synchronization protocols. In practice, many on-body and pervasive sensing setups do not have such capabilities. For one, leaving out the receiving capability reduces power consumption and makes the node design simpler. In addition, systems are often built using custom nodes from different manufacturers with insufficient synchronization support. Often such nodes do not provide access to the internal communication routines.

It is common practice in wearable and pervasive application studies to rely on event-based synchronization of data streams. To this end, 'synchronization actions' are defined, which are periodically performed by a subject during the recording (at least once at the start of a data collection, or more often to detect drift). Such actions are designed to simultaneously activate multiple sensors and provide an easily identifiable data signature. Often encountered examples are clapping, jumping, and hitting a surface. The characteristic signature of a synchronization action can be used to align signal segments from different sensors. Currently, this is typically done through manual inspection of the data streams. In this chapter we investigates a method to automate this process.

It is intuiting to rely on natural activity patterns to synchronize sensor data streams that are performed by users in a pervasive application. However, it is essential to develop a robust synchronization framework in the first place. Hence, we chose validation scenarios for this initial work, in which particularly selected synchronization actions had been inserted into sensor data streams. We subsequently discuss how our concept could extend on natural activities.

3.1 Related Work

Time synchronization is a well known and widely studied problem in wireless sensor networks, in particular for subsequent sensor data fusion. Surveys can

be found in [Siv04] and [Sun05]. Typically, the solutions target a network-wide synchronization by aligning the clocks of all physically distributed nodes [Sun05, Su05]. For this purpose specific messages are exchanged among the nodes.

As detailed above, we target systems where a synchronization based on exchanging messages is not feasible due to unidirectional communication channels. Our approach relies on data signatures (events) embedded in data streams, hence it does not modify the source clocks. However, it ensures a relative synchronization between data sources at the fusion unit.

Instead of establishing a network-wide clock, various communication systems use a source synchronization approach for medium access. These techniques target to align message recipient(s) on a link with the clock provided by the sender, through monitoring packet reception [Mac01] or protocol-specific features, such as a preamble. Our approach is similar to source synchronization, as the fusion unit observes the relative offset in the incoming data streams.

More relevant research looks at correlation of different sensor data streams for various applications. This includes the use of correlations to determine that a set of devices are carried by the same person [Les04], as well as attempts to use correlation between sounds received by different sensors for indoor location [Bia05].

Many researchers experienced clock synchronization failures in deployed sensor networks due to software bugs, communication failures, and frequent reboots, rendering large parts of data unusable [WA06, Gup09, Luk09]. Common solutions try to recover corrupted timestamps in a postmortem process. Werner-Allen [WA06] corrects errors caused by the time synchronization protocol on behalf of a base station that adds its own timestamps to the collected data. They build piecewise linear models to map local node time to global time without utilizing the actual sensor data collected by the nodes. Lukac [Luk09] proposes data driven time synchronization by utilizing background noise in seismic sensing systems. By building a model of the propagation of micorseisms (seismic waves that travel through continents, originated by oceans) they can reconstruct timestamps in large datasets gathered by seismic networks. The Sundial system [Gup09] uses light sensors to detect length of day and noon time and compares them offline with the astronomical model to estimate the correct global timestamps. While this approach is based on a single sensing modality they also apply a method to correlate rain events with soil humidity events for finding the correct day. This is very similar to our approach but we do not rely on additional information about the events to find the correct mapping and we explicitly allow event spotting errors.

3.2 Challenges of event-based stream synchronization

The conceptual solution for event-based synchronization is straightforward. We assume that each data item is time-stamped with a local clock (or stream sequence number). Once a first synchronization action is localized in all data streams, offsets between the local clocks or sequence numbers are computed. Upon locating a second event, differences between sensor clock frequencies can be computed. Subsequently, the data streams are aligned according to these

differences. Additional events may help keeping streams synchronized over time and could increase synchronization accuracy.

In reality, there are a number of problems related to reliable, automatic spotting of synchronization actions. It is well known that recognizing short actions embedded in a continuous stream of arbitrary sensor data is a hard problem. To synchronize data streams based on events, actions must be spotted separately in each data stream. This is even more difficult than under a multi-modal spotting scheme, where signals from several sensors can be combined. Hence actions could be confused with other arbitrary actions. As a consequence, the synchronization algorithm has to cope with deleted (missed) and inserted (wrongly recognized) events in individual streams. Hence it is not a-priori clear which events should be aligned with each other.

Another particular spotting property is the timing variations among retrieved events. For an event-based synchronization approach, this variation can have two origins. On one hand, a spotting procedure can introduce a temporal jitter due to an assumed segmentation. Typically, this jitter is a fraction of the expected event size [Amf08]. Secondly, spotting on independent sensors can impose different event patterns. For example, hitting a table with a fist will generate a short temporal acceleration at the wrist, while the table will vibrate longer with a delayed onset. Thus, even if the synchronization actions are correctly spotted, it is not always clear how to relate events in time.

In this chapter we investigate solutions to the problems described above, facilitating a practical implementation of automatic, event-based synchronization. We demonstrate that synchronization actions can be identified across widely different sensors, such as accelerometers and optical motion tracking systems. We show how using repetitive events can improve synchronization results. Moreover, we show that the timing relation is transitive for sensor groups with shared members. The latter can be exploited for further reliability improvements and to synchronize larger multi-modal sensor networks.

3.3 Spotting and synchronization approach

At first, we define two terms on which we rely within this chapter:

action An action describes the physical movement that is conducted by a subject. Sensors which are able to perceive the action will show a distinct pattern in their data stream at (and around) the time of perception.

event An event denotes the individual detection of a pattern instance caused by an action. It has a certain extent in time (duration) and it marks a single, distinct point in time as its "synchronization point".

Our approach consists of two steps. In the first step appropriately defined synchronization actions are spotted in each sensor data stream. Specifically, we spotted hand 'clap', 'push-release' of a button, and arm 'shake' actions as described below. In a second step our online synchronization algorithm is used to establish, which events in two different streams correspond to the same physical action. This step deals with missing and inserted events, as well as temporal jitter. The result of this step is a continuous alignment estimation corresponding to events derived from the participating data streams.

The required synchronization performance is application dependent. For a typical example of motion-related activities in daily life, such as considered in this work, stream alignment should be well below 1 s. Nevertheless, an alignment performance below 0.1 s is typically not needed. In our evaluation we selected a performance of 0.3 s as target alignment.

3.3.1 Event spotting

In our approach, synchronization actions are associated with a single timestamp from each data stream. Nevertheless, even short physical actions such as a hand clap exhibit a temporal signal pattern. Figure 3.1a illustrates such patterns for acceleration and audio streams. Hence, we specifically modelled a synchronization point in the event patterns using signal features. Two sliding window algorithms were used to spot 'clap', 'push-release', and 'shake' gestures on individual modalities. Figures 3.1a-3.1c show these events on four different sensing modalities (acceleration, force sensitive resistors (FSR), positioning sensors, and audio). Other event types or sensing modalities may require further spotting algorithms, which can be easily designed.

For both algorithms we obtained model parameters manually by analyzing pattern examples. The parameters were chosen to minimize event miss rate. Recall is more important than precision for the purpose of this work, as false positives are usually discarded at the subsequent stage.

Trailing edge detection

The patterns produced by 'clap' and 'push-release' events have similar characteristics. Their signal pattern have a stable phase of "silence" followed by a fast rise or fall (see Fig. 3.1a and 3.1b). The start of a rise/fall phase was considered as the synchronization point. The two events were distinguished by the features: minimal length of silence phase, signal range of silence phase, length of rise/fall phase, and signal range of rise/fall phase.

Shake detection

The detection of shakes was implemented by tracking peaks (alternating local minima and maxima) over time. The features, minimal peak height, min/max of time between peaks, and number of peaks had been used. The begin and end of shake events may vary between sensing modalities (shown in Fig. 3.1c). Consequently, we defined the center point (midpoint between first and last peak timestamp) as synchronization point for shake events.

3.3.2 Event-based synchronization

While the event pattern structure (such as signal peaks for 'shake' events) could be used to analyze the synchronization between two sensor streams, we found these patterns to be too variable. Moreover, the patterns may differ between modalities, which would hamper their alignment. Instead, our approach relies on a sequence of spotted events from both streams to estimate alignment. In particular, we consider the temporal distribution of the retrieved events as relevant synchronization property. Thus, as long as events do not occur with

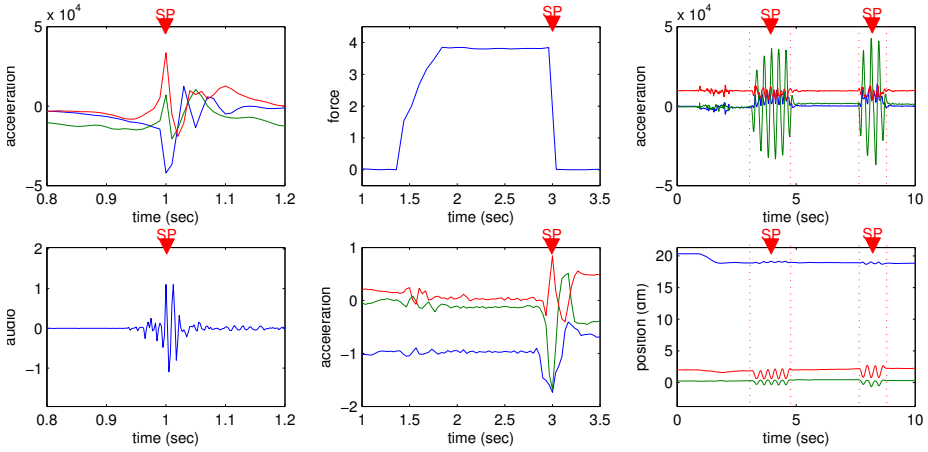


Figure 3.1: Aligned sensor signals: (a) hand 'clap' recorded from wrist mounted accelerometers and a microphone, (b) button push followed by sudden release recorded from an FSR under thumb and wrist mounted accelerometers ('push-release'), (c) two 'shake' motions recorded from accelerometers and a camera-based positioning system. "SP" indicates synchronization points returned by our spotting procedure and subsequently used for alignment.

constant frequency, we expect that a match between event sequences of both streams can be found. Clearly, the probability for a correct event match will increase with the sequence length of matching events.

Algorithms with similarities to the one proposed here have been used in different fields, but to the best of our knowledge, such an approach for synchronizing data streams has not been explored before in the field of context recognition.

Algorithm: The synchronization algorithm compares two event sequences $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ to estimate a sequence match. A sliding window with a size of 30 s was applied on the event sequence for online operation and retain multiple events for alignment analysis at any considered stream position. The procedure can be understood as shifting sequence B in time until an optimal match is found with sequence A.
Let

$$d(i,j) = t(b_j) - t(a_i),\ i \in [1, \ldots n],\ j \in [1, \ldots m] \tag{3.1}$$

be the n-by-m matrix of timestamp differences between all possible events of sequence A and B, and let

$$P = \{(i_1, j_1), \ldots, (i_k, j_k)\},\ k > 1, i_l < i_{l+1}, j_l < j_{l+1}\ \forall l \in [1, \ldots k-1] \tag{3.2}$$

be a matching path of length k between A and B. Each cell (i,j) in path P denotes a match between event a_i and b_j and therefore indicates an offset of $d(i,j)$ between A and B. Matches in P are unique ($i_l \neq i_{l'},\ j_l \neq j_{l'}\ \forall l \neq l'$)

but not necessarily include all events ($i_{l+1} - i_l \geq 1,\ j_{l+1} - j_l \geq 1$). The latter allows coping with event spotting errors.

We analyze all potential paths (candidate paths) and select the best by applying a weighting function $w(P) = w_K(P) \cdot w_V(P)$ on each candidate. Weights w_K and w_V were defined with an intend to maximize path length k and minimize variance of path distances $v = var(\{d(i_1, j_1), \ldots, d(i_k, j_k)\})$. The synchronization algorithm provides a continuous estimation of relative stream offsets between two sources by selecting $P_{best} = max[w(P)]$.

The path search can be performed incrementally. For a given cell (i_l, j_l) a subsequent cell (i_{l+1}, j_{l+1}) in the path is determined by searching in the submatrix of (i_l, j_l):

$$j_{l+1} = \arg \min_{j' \in [j_l+1 \ldots m]} |d_1 - d(i_l + 1, j')| \text{ and} \tag{3.3}$$

$$i_{l+1} = \arg \min_{i' \in [i_l+1 \ldots n]} |d_1 - d(i', j_{l+1})|, \tag{3.4}$$

where $d_1 = d(i_1, j_1)$ denotes a path's starting point. If $|d_1 - d(i_{l+1}, j_{l+1})|$ does not fall below a tolerance threshold this cell is discarded and searching is repeated in submatrix $(i_{l+1}, j_{l+1} + 1)$. In our evaluations a tolerance of 0.3 s was used to prune non-relevant paths. Multiple candidates of matching paths are found by starting a new search from each top and left border cell (see Fig. 3.2). This corresponds to shifting sequence B against sequence A.

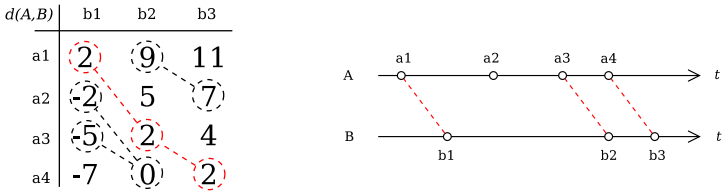


Figure 3.2: Two event sequences with the corresponding matching paths highlighted in the timestamp differences matrix (a), and the best path displayed on the timeline (b).

For path weight $w_K = 1 - e^{-((k-1)/(2-k_E))^2}$ was used where k is the actual path length and k_E denotes the expected path length. As variance weight we used $w_V = 1 - e^{-0.5(v/R)^2}$, where regularizer R serves to control variance weight contribution. Weighting behavior is designed to maximize weight for particular parameter sets. For w_K a maximum is obtained at $k = k_E$, for w_V, variance should be minimized.

Parameters k_E and R can be used to adapt our synchronization algorithm. Expected sequence length k_E depends on the targeted synchronization performance, where larger k_E, hence longer sequences, will result in reduced synchronization errors. Parameter R controls the influence of temporal event jitter in w_V. For this work, we determined $R = 0.05$ in empirical tests and confirmed it in all evaluations discussed in Section 3.4.

It should be noted that our heuristic algorithm can miss a globally optimal path as Eqs. 3.3, 3.4 are sequentially evaluated. We consider our approach as a tradeoff limiting computational complexity compared to a full search which

would involve evaluating all potential paths within a submatrix of cell (i_l, j_l). Our evaluation results confirm that this approach is feasible.

3.3.3 Specific synchronization properties

Transitivity

Synchronization alignments, as determined with our approach, are transitive. If relative offsets between data streams A and B and between stream B and C are known, then the offset between stream A and C can be deduced. This transitive property allows that sensors or sensor subnets, which do not share common events with other sensors, can still be synchronized on behalf of mediating sensors. We demonstrate how this property can be exploited by analyzing synchronization pairs in Section 3.4. Moreover, such dependencies could be used to refine synchronization result, as potentially multiple synchronization sources become available.

Performance bounds

We analyzed theoretical performance bounds for our synchronization algorithm with regard to event input. In particular, we considered the following essential properties:

- effect of event sequence timing,
- impact of the number of synchronization actions, and
- temporal event jitter introduced through the spotting method.

A sequence of events $a = \{a_1, \ldots, a_n\}$ can be described in terms of temporal distances between subsequent events

$$s = \{s_1, \ldots, s_{n-1}\} = \{t(a_2) - t(a_1), \ldots, t(a_n) - t(a_{n-1})\}, \tag{3.5}$$

where $t(a_i)$ denotes the timestamp of event a_i. The worst-case scenario for our synchronization algorithm is a monotonous sequence of constant event distances, hence $s_1 = s_2 = \ldots = s_n$. In this specific case, an alignment can not be uniquely identified using our distance approach. However, in practice this situation will be rare and difficult to construct.

If we consider some degree of temporal variance in an input event sequence, we may describe s as a random variable with Gaussian probability distribution $p(s) = \mathcal{N}(\mu, \sigma)$. We denote $\hat{s}$ as an actual distance between two spotted events and $\pm\omega$ as the interval of jitter caused by the spotting method. Consequently, the probability for obtaining a specific event distance can be derived by:

$$P(s = \hat{s} \pm \omega) = \int_{\hat{s}-\omega}^{\hat{s}+\omega} \mathcal{N}(\mu, \sigma), \tag{3.6}$$

which has its maximum at $s^* = \mu$. The lower this probability, the lower the risk that the distance $\hat{s}$ randomly appears in a sequence, and the more likely that a certain interval $\hat{s} \pm \omega$ is unique and thus can uniquely be matched between sensors. Hence, if σ increases, the probability for a unique event match is raised, as the maximum of $P(s)$ decreases. In contrast, if the temporal event jitter ω

increases, $P(s)$ increases as well. This means, that it is more likely to randomly observe a specific event distance.

In the analysis above we considered a distance between two events only, hence $k = 1$. In a more practical setting, however, several synchronization actions may be considered for estimating alignment and we may safely assume that the probabilities $P(\hat{s}_i \pm \omega)$ are independent. Thus,

$$P_{\mathsf{path}}(\omega, k) = P(s_1 = \hat{s}_1 \pm \omega, \ldots, s_k = \hat{s}_k \pm \omega) = \prod_{i=1}^{k} P(\hat{s}_i \pm \omega) \tag{3.7}$$

indicates the probability for a matching path of length k. Hence, the risk of a random match decreases exponentially with increasing event sequence length k.

3.4 Evaluation

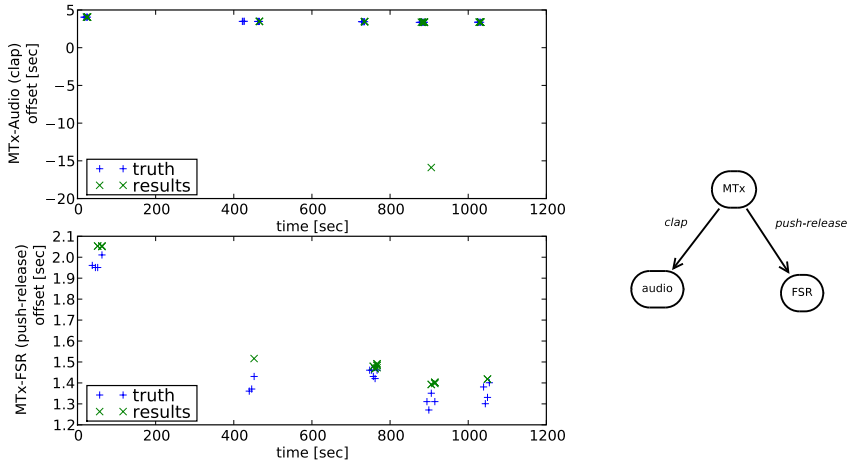
We performed three experiments to analyze the synchronization performance for different events and sensor modalities. All evaluations use sensor modalities that are relevant in ubiquitous applications and have been used in previous studies. Initially, two experiments where performed to investigate the synchronization approach with different sensor modalities and combinations (force-acceleration-audio, acceleration-positioning). In a subsequent evaluation, we investigate the performance of the synchronization algorithm in a large data set (totally 308 min. of 5 users) that was recorded for activity recognition in daily living scenarios [Amf10]. All synchronization actions were annotated and refined in a post-recording step by visual inspections of the data streams. For all experiments the correct alignment was manually determined from signal inspections. Annotation and alignment information was used as ground truth.

3.4.1 Evaluation 1: force-acceleration-audio

Data recording

The dataset was recorded with a single wrist worn motion sensor (Xsens MTx), one FSR on the desk, and an ambient sound microphone. The MTx comprises 3D sensors for acceleration, rate of turn, and earth-magnetic field. In this investigation only 3D acceleration was considered. MTx were sampled with a nominal data rate of 100 Hz, FSR at 16 Hz, and audio at 8 kHz. For FSR and MTx a wireless connection (Bluetooth) was used to transmit data to a computer. Audio was acquired and timestamped on a second computer and streamed to the first computer for archival.

The total recording time was ∼20 minutes. During this time five sessions of 'push-release' gestures and six sessions of 'clap' gestures were performed by one person. Each gesture was repeated several times during a session resulting in a total of 20 'push-release' and 24 'clap' gestures. Between these synchronization sessions normal office activities were conducted and several times the wireless connection is intentionally broken by increasing the distance to the recording computer. These link interrupts served to simulate both, connection errors and temporary sensor failures.



(a) (b)

Figure 3.3: Evaluation 1: (a): Results for the force-acceleration-audio dataset. The offsets indicate the actual (truth) and estimated (result) alignments for the entire dataset. (b): Synchronization pairing scheme.

Synchronization procedure

Using the transitive property of our approach, two synchronization pairs were established in this analysis: (1) MTx-FSR using 'push-release' and (2) MTx-audio using 'clap' gestures. The synchronization pairs are illustrated in Figure 3.3b. The streams of each synchronization domain were independently aligned using our synchronization algorithm. We used the trailing edge spotting algorithm to recognize both event types in all streams.

Results

The event spotting returned 95.5% of the gesture instances correctly. Only 4 instances from the MTx sensor were missed. In total 23 false positives were returned (26.1%), 19 of them by the clap detection on the accelerometer signals, which classified most 'push-release' events as 'clap' events. The remaining false detections were caused by the FSR spotting, which generated insertions at rising signal edges.

Figure 3.3a shows the synchronization algorithm results as a time plot. Blue '+' signs indicate true offsets, manually determined at each event instance. Green crosses mark the offsets estimated by our synchronization algorithm. At least one result was found for each session of 'push-release' gestures with an error of $\sim$0.1 s. One session of clap gestures did not lead to a result because of missed events. This session contained only three annotated event instances, which was also used for k_E. Consequently, if one event was not retrieved by the spotting algorithm, a correct match was no longer possible. The MTx-audio pair incurred one wrong result with an error of -10 s due to event insertions. Such outliers would be eliminated before actually aligning the streams.

3.4.2 Evaluation 2: acceleration-positioning

Data recording

We attached infrared markers of an optical motion capturing system (Lukotronic) onto two accelerometer-based motion sensors (Xsens MTx). The position data from the Lukotronic system was streamed over a wired network connection (TCP) to the recording computer. The acceleration data from the two MTx sensors was transferred over two independent wireless connections (Bluetooth). All sensing systems sampled with a nominal data rate of 100Hz. A dataset of ∼60 minutes was recorded during which individual units and both units together were shaken periodically.

Synchronization procedure

Three synchronization pairs were established: (1) MTx0-Lukotronic, (2) MTx1-Lukotronic, and (3) MTx0-MTx1. The synchronization pairs are illustrated in Figure 3.4b. The streams of each synchronization domain were independently aligned using our synchronization algorithm.

We used the shake detection algorithm to spot 'shake' events for both sensor modalities. As described before, we considered the midpoint of a shake event as synchronization point. This approach compensated varying detections of peak begin and ends on both modalities.

Results

The manual data stream alignment revealed that the camera system had an offset of 8 seconds relative to both accelerometers. This can be explained by different starting times of both system types and by differences in transmission delays. In addition to the stream offset, skew and drift seemed to be marginal for this recording.

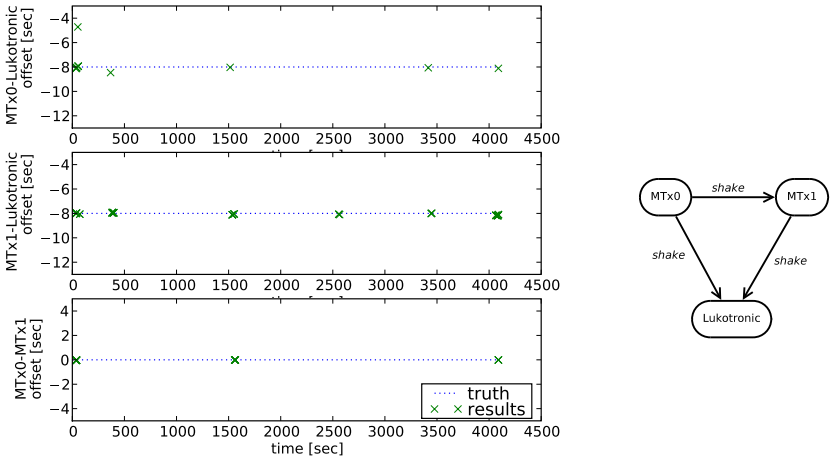
The event spotting correctly returned all synchronization actions and did not incur insertion errors. Figure 3.4a shows the synchronization algorithm results as a time plot for all synchronization pairs. The plot can be interpreted as discussed for evaluation 1 above. The algorithm correctly detected that no offset existed at multiple time points on each synchronization domain. The algorithm incurred one error due to a similar event sequence found in one stream.

This experiment demonstrates that repetitive gestures (with an extent in time) can be used as synchronization actions. Moreover, it confirms the feasibility to use the transitive property for independent synchronization pairs. Sensors that are members of two independent synchronization pairs could use both to refine alignment results.

3.4.3 Evaluation 3: Office scenario

Data recording

A setup consisting of a motion sensor at the right wrist (Xsens MTx), left wrist (Xsens MTx), and ambient sound microphone was worn by the users. In addition, four FSR sensors were mounted under pads that could sense if household utensils are placed on them were used. The right wrist motion sensor was attached to an Xbus, sampled at 30 Hz and interfaced to a laptop computer



(a) (b)

Figure 3.4: Evaluation 2: (a): Results for the acceleration-positioning dataset. The offsets indicate the actual (truth) and estimated (result) alignments for the entire dataset. (b): Synchronization pairing scheme.

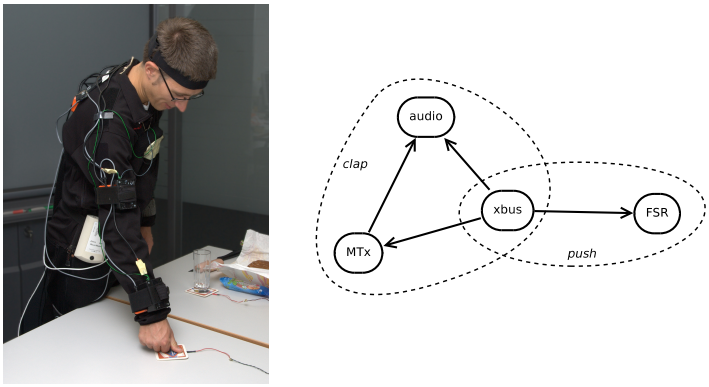
using a wired connection. For the microphone a wired connection at 8 kHz sampling rate was used. The left wrist Xsens was connected through a wireless link (Bluetooth) at 30 Hz. The FSRs were synchronously sampled at 12.5 Hz by another (stationary computer). During the actual study the laptop that interfaced all wearable sensors, was carried by an experiment observer who followed the test person. In total, this setup had four unsynchronized data streams recorded by two computers.

Five test person were individually recorded for sessions of ∼60 minutes. During these recording sessions the test persons were asked to perform various activities in four different scenarios: office (desktop work, printing, using fax), eating (preparing sandwich, making coffee, eating), gaming (installing a Wii console, gaming using the Wii remote), and leisure (reading magazines, speaking at the phoning).

Synchronization procedure

Three times during a recording session (beginning, mid-time, end) the test persons were asked to perform synchronization actions consisting of 'push-release' of one FSR pad on the table using the right arm (activation of right MTx and FSR), and 'clap' with both hands (activation of both MTx and audio). Figure 3.5a illustrates the synchronization action for 'push-release'. The synchronization actions were repeated three to four times in each synchronization session, resulting in a total of at least 45 relevant synchronization actions per test person (when counted on each data stream separately).

Four synchronization pairs were established: (1) right MTx-left MTx using 'clap', (2) right MTx-audio using 'clap', (3) left MTx-audio using 'clap', and (4) right MTx-FSR using 'push-release'. The synchronization pairs are



(a) (b)

Figure 3.5: (a): Test person during push-release gesture. (b): Synchronization pairing scheme for evaluation 3.

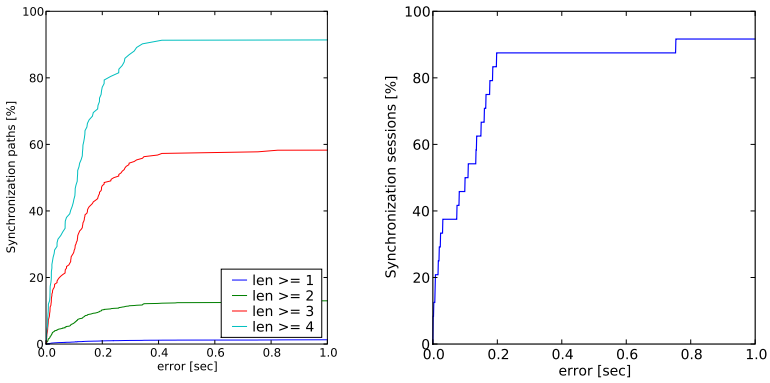
illustrated in Figure 3.5b. The streams of each synchronization domain were independently aligned using our synchronization algorithm. We used the trailing edge spotting algorithm to recognize both event types in all streams.

Results

The event spotting algorithm was used with the same parameter set for all test persons. In total for all persons, 83% of the synchronization actions were correctly spotted. The procedure incurred 654 insertion errors (284%) with low inter-person variances. This result should be considered in relation to the recording time (total: 300 minutes), resulting in an average of two insertions per minute. This insertion rate was expected due to the simple spotting procedure and the limited model training. Elaborate spotting procedures, automated training, and person adaptation could improve the result, but the focus of this evaluation lies on the synchronization method in the subsequent step. And for this purpose such event spotting results offer real life test conditions.

Figure 3.6a shows the cumulative error distribution for individual synchronization path lengths. This result was obtained by analyzing the synchronization error for all candidate synchronization paths. The result confirms that with increasing path length k the probability for large alignment errors decreases. When considering the targeted alignment error of 0.3 s, ∼55% of all cases for $k \geq 3$, and ∼90% of all cases for $k \geq 4$ are below this error bar. In contrast, for path length of $k \geq 2$ no useful synchronization performance is obtained. This means that it is very common to find two events in one stream that incidentally have the same distance (time difference) as two events in the paired stream. This observation is related to the relatively high insertion rate for this dataset and the deployed event spotter. However, path lengths of three or four events are sufficient to reach a reasonable synchronization performance. With the simple event spotters considered in this evaluation, we observed that $k = 4$ is needed in many cases.

Furthermore, we analyzed the error distribution with regard to the synchronization sessions that were identified by the event spotting. Figure 3.6b confirms



(a) (b)

Figure 3.6: Evaluation 3: Quantitative results of the synchronization. (a): Cumulative distribution results for all candidate synchronization paths and different path length k. (b): Cumulative distribution of synchronization sessions that have at least one alignment estimation result with regard to the synchronization error. The synchronization paths length k was not restricted for this representation.

that for more than 80% of the synchronization sessions a synchronization error of less than 0.3 s was achieved.

3.5 Conclusion

Our experiments clearly show the potential of an automatic, event-based approach to synchronize distributed multi-modal sensor systems. We concluded from the results that our synchronization algorithm can achieve the initially set target of 0.3 s synchronization error. The effort for finding suitable values for the two parameters k_E and R turned out to be negligible, as we achieved the best results with a single setting (k_E=3, R=0.05) for all experiments and subjects. If synchronization actions are repeated more often per session, k_E may be increased to achieve more reliable results. However, the evaluations also showed that in realistic scenarios achieving high synchronization performance is not trivial. In evaluation 3, a synchronization path length of four was required to achieve a performance greater than 80% for the targeted error level. This result indicates that an appropriate event spotting and subsequent reasoning across different synchronization pairs is the key to good performance. In addition, more investigation is needed on suitable synchronization actions for different sensor combinations. We expect that including natural activities in the event-based synchronization approach will help to further improve the synchronization performance as such activities will be present throughout the recordings instead of just within artificially inserted synchronization sessions.

Automatic synchronization is a valuable element for online context recognition systems as it further relieves the developers and users from manual actions.

The method allows for constantly monitoring the temporal alignment of data streams and to initiate actions to re-synchronize the streams if necessary. In offline tools operating with recorded data streams, e.g., the annotation tool presented as part of an integrated tool chain in the next chapter, the method can automatically suggest corrected alignment of the streams which the user may refine if desired.

In the context of opportunistic activity recognition systems, automatic synchronization is a key element to guarantee temporal "compatibility" of new sensors that just appear, or sensors that have previously been used by the system but disappeared for a while an then suddenly reconnect with unknown clock state. Yet, synchronizing an unknown sensor poses new problems to be addressed in future work. How can synchronization events reliably be detected on an unknown sensor, and how do such events relate to events seen on the known sensors?

Chapter 4

Integrated Tool Chain

Building activity recognition systems always requires data sets for training of the involved recognition methods, and retrieving good training sets can be expensive. In Chapter 2 we have introduced the CRN Toolbox which already provides means to define a data flow for recording live sensor data from heterogenous sources. While it facilitates the setup of recording- and recognition chains, it does not cover challenges beyond. In this chapter we describe an integrated tool chain, built around the CRN Toolbox, for the development, testing, and deployment of multimodal activity- and context recognition systems. The tool chain aims to support data collection with additional tools, allow structured storage and retrieval of annotated sensor signals and supporting material, simplify the labeling and associated post processing, facilitate flexible usage and combination of different parts of a recording, and allow easy implementation of distributed, adaptive recognition systems. The tool chain has been implemented around the established CRN Toolbox rapid prototyping platform (see Chapter 2), has been successfully used in a variety of experiments, and is available to the community as an open source project.

David Bannach and Paul Lukowicz. Integrated tool chain for recording, handling, and utilizing large, multimodal context data sets for context recognition systems. In *ARCS 2011*, 2011.

David Bannach, Kai Kunze, Jens Weppner, and Paul Lukowicz. Integrated tool chain for recording and handling large, multimodal context recognition data sets. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Ubicomp '10, 2010. ACM.

The development of context aware systems involves a set of complex steps that may differ in detail from case to case but in essence reoccur in most applications. In general the process starts with the collection of a sample data set. Issues that typically need to be addressed include reliable streaming of data into appropriately organized storage, detecting faulty sensors and missing data points, synchronization of sensors and initial data labeling. In most cases the data collection needs to be followed by a post processing enrichment step in which the synchronization points and the labels are refined and missing

data points and signals dealt with. This requires different sensor channels to be reviewed, compared with each other and possibly synchronized with a video record of the experiment. For non trivial data sets such post processing can be extremely time consuming. In previous work (e.g. [Luk10, Rog10]) we have found around 10 hours of work to be needed for each hour of recorded data. Once the data has been enriched the core of system development can begin. It is usually an iterative process consisting of feature selection, classifier training and tuning, and performance testing. Often the so developed classifier then needs to be ported to a final platform such as a mobile device or a set of such devices.

The sum of those steps make it practically impossible for non-experts to succeed in building an activity recognition system and also for experts it is a tedious and error-prone process. We want to find out if those steps can be structured and supported by an easy-to-use toolchain such that less expert knowledge is needed. Would it be possible to simplify and structure the processes enough such that higher-level algorithms could control them? We try to answer the questions with user-centered design and by applying known techniques from different fields, integrating them into an extensive tool chain, making it available for everybody, and using that toolchain in various projects and experiments.

In this chapter we describe the integrated tool chain that evolved over the course of several years and a number of experiments [Sti06b, Sti08a, Luk10] to support the above described process and make it more efficient. The tool chain is built around the CRN Toolbox (see Chapter 2) which is already widely used for collecting and processing sensor data. It allows predefined, parameterizable algorithms and sensor interfaces to be put together into a complete application using a simple GUI. The work described in this chapter goes beyond the original Toolbox presented in Chapter 2 in the following ways:

1. We have built CRN Toolbox compatible context data loggers for common mobile devices (iPhone, Android).

2. We have implemented a GUI based tool for dynamic monitoring of data flowing through a toolbox application. This is crucial to ensure data integrity during long term recordings with a large number of sensors.

3. We have interfaced the CRN Toolbox to a database system (Apache CouchDB) in order to allow sensor data, annotations and accompanying videos to be stored and accessed in an organized way. The database uses the map-reduce paradigm to efficiently down-sample data streams for presentation in a web browser.

4. We have developed a GUI based tool (interfaced to the CRN Toolbox) for the inspection, annotation, and manual resynchronization of context data stored in the above database. The tool allows flexible views of selected sensor streams to be compared to a video recording, moved around in time, and given ground truth annotations at different concurrent levels of granularity.

5. We have implemented a trace generation tool that allows combinations of signals from a subset of sensors referring to certain ground truth events to be easily retrieved from the database and streamed for system training, testing or demonstration.

6. We have demonstrated how multiple instances of the CRN Toolbox and context loggers can be combined into a distributed, service oriented system that includes dynamic resource discovery allowing for easy construction of distributed, adaptive context recognition systems.

All above functionalities have been implemented and most have been used in "production mode" in different experiments [Ban10, Luk10, Rog10]. They are available to the community as an open source project and are used for the dissemination of different public data sets[1].

After discussing related work the rest of the chapter describes each of the above components and its applications.

4.1 Related Work

The number of publicly available activity recognition datasets has grown in recent time [Int06, Ten09, Kas08, cmu]. Each of them is available for download from their own website. Most of them comprise multiple files per recording session, one for each sensor system, and are shared together with their own visualization and annotation tools. Contributing refined annotations, comparing, or combining datasets can be difficult.

One such tool is the BoxLab Visualizer [box]. It supports playback of several parallel video and audio streams and the timeline shows color-coded activation times from binary sensors. Time series data from accelerometers are also visualized. The tool shows two kinds of annotations, listed in separate windows: annotations with start- and end time, and partial annotations having a start time only. Annotations can have certainty ratings. A color code shows which annotations are active for the current playback time. The software runs on Windows systems only. Source code is not available.

Anvil [Kip01] is a tool for annotating videos with different levels of label granularity. It was created for investigating human gestures based on video recordings. The labels are visualized on a timeline in multiple tracks. The number and type of label attributes can be defined separately for each track. It does not support loading, visualizing, and synchronizing time series data from sensor recordings. The tool is based on Java and therefore runs on many platforms. It has a rich plug-in interface which would allow for adding support of time series data to a certain extent but source code for the core framework is not available.

The MARKER [mar] labeling tool is a Matlab toolkit which allows for visualizing, synchronizing, and annotating of time series data from sensor recordings. Videos are not supported. The toolkit is limited by the performance and requirements of the Matlab system.

TU-Munich provides a custom, simple labeling tool [kit] for each version of the kitchen datasets [Ten09, cmu]. The tools show multiple videos in parallel, a timeline for scrolling forth and back in the videos, and provide several tracks below the timeline where markers can be added. Time series data of sensor recordings is not displayed. Configurations are made directly in the available source code.

[1] http://www.contextdb.org

In our work we try to integrate good features of existing tools into a single tool chain that is suitable for most projects handling multimodal context recognition data sets, and that allows for easier sharing of such datasets.

There are several toolkits available that support the development of context sensitive systems in general. However, most of them contribute only parts of the complete tool chain necessary for creating context recognition systems and are often not flexible enough to be integrated. The Context Toolkit [Dey01], for example, focuses mainly the application level. The core qualities of the CommonSense ToolKit [Lae] are its real-time facilities and embedded systems-friendly implementation, but not its flexibility for integration.

A much broader coverage of the complete tool chain is presented in [Gün10] with a lightweight, easy to use tool chain which provides approaches for all phases from recording, storing, annotating, and processing of context data to the final classifier for context recognition. The focus of that tool chain lies on simplicity and robustness and therefore it is limited to a single classification method and single sensor input. Our approach differs in that we aim to provide a general and modular solution which may be applied to a broad range of problems and execution environments. Specifically, our approach covers simultaneous data recording for a broad range of sensors, progress monitoring and quality control during recording sessions, synchronization and video-based labeling of multimodal data streams, storing, browsing, and sharing of context data sets, and utilizing context data sets in distributed, multimodal context recognition systems.

4.2 Concept

Our tool chain consists of three integrated phases (see Fig. 4.1). The first phase is the *data collection* phase where training data is recorded from real sensors and video cameras during various experiments. During this process the status of all sensors can be monitored to ensure the quality of the recordings. The recorded data sets are stored in a central database which is maintained in the *data management and annotation enrichment* phase. Data sets can be reviewed, organized, and re-labeled using graphical user interfaces. The web-enabled database also allows for sharing the data sets and annotation efforts with the community. Furthermore, specific reproducible data traces can be generated from the database which are necessary for training and evaluating methods for context recognition. In the *classifier training and online context recognition* phase the training data is used to parameterize the context recognition methods. These methods are connected to sources of live sensor data for online context recognition on target devices. Results as well as lower level data can be shared dynamically among such recognition components to allow for adaptations to changing environments. In the following we discuss each of the tree phases in detail.

4.2.1 Data Collection

Sometimes it is inevitable to record new sensor datasets, e.g., when there is no public dataset available that suits the requirements of a new context recognition method. Collecting good sensor data for training and evaluation of context

recognition methods is a tedious work, especially when multiple sensors and sensors with different modalities are involved. In this section we present three tools that help recording multimodal sensor data sets for context recognition.

A Toolbox for Rapid Prototyping of Context Recognition Systems

For data recording in general, e.g., with wearable or stationary sensors, we use the CRN Toolbox presented in Capter 2. Its modular, data-driven concept and its implementations of reader tasks for various sensors make it a valuable tool for the recording of multi modal data sets. In its simplest configuration, signals from sensors may be directly streamed into individual files on the recording computer(s). On-line annotations may be merged directly to those data streams

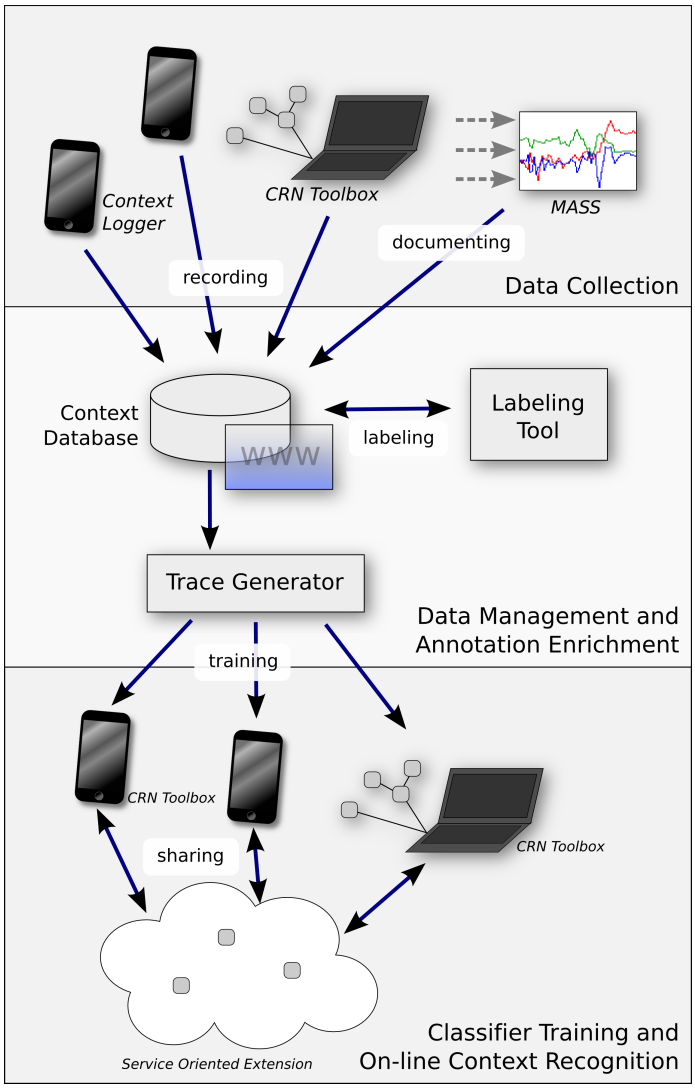


Figure 4.1: Overview of the context recognition tool chain.

or recorded into separate files for later merging via timestamps. Such a base configuration is easily extended, using writer tasks, to support streaming of live data for monitoring purposes (see below) or to upload the recordings directly to the context database (see Section 4.2.2).

This Toolbox configuration, especially the reader components, can be re-used later in the third phase (see Figure 4.1) when live sensor signals are needed for on-line context recognition.

Context Logger for Smartphones

One kind of largely pervasive sensor nodes are today's smartphones, most of them equipped with inertial measurement sensors, location services, and means to record ambient sound and identify nearby wireless nodes. Yet, they are often worn in pockets close to the body in differing positions and orientations, making them an exemplary source for opportunistic sensing. Diverse context sensitive applications have been developed already for mobile devices, e.g., [Yan09, Sap08, Sie03, Gel02].

In order to facilitate the recording of sensor data from smartphones we implemented dedicated data logging applications for the Apple IOS and the Google Android platforms. The application allows for an easy recording and labeling of all sensors on the device. These include accelerometers, gyroscopes, compass, GPS, sound, WiFi information (name, MAC address, signal strength, protection, visibility), light and proximity sensors, and phone events. The apps record sensor data into local files on the device to ensure a lossless logging of the signals and in addition can stream live data to a monitoring device. The recordings can be uploaded to the Context Database described in section 4.2.2.

Live Monitoring of Recordings

During recordings with larger number of sensors it is crucial to monitor the state of each sensor and also to document the recording sessions. Sensor failures can occur often (for various reasons) and would make whole multi-hours of recordings unusable if not detected early enough. For this purpose we developed *MASS* (Monitoring Application for Sensor Systems). MASS is a software tool that helps monitoring and documenting experiments with multimodal sensor setups. It features graphical and tabular views for visualizing sensor up-times and dynamic plots of live sensor signals for quick checking of signal quality (see Fig. 4.2). Users may visually place sensor status icons on images and maps to document sensor positioning and to get an intuitive, location-based overview of sensor states. All modifications and events during a recording are logged for documentation purposes.

MASS is implemented in Java and integrates seamlessly with the CRN Toolbox. The recording tools advertise their service through Multicast-DNS such that MASS can find them on the local network and connect to them. MASS receives sensor status updates via UDP multicast messages from the recording tools and can monitor the sample rate to detect anomalies. For displaying live sensor signals MASS dynamically opens a TCP connection to the recording tool.

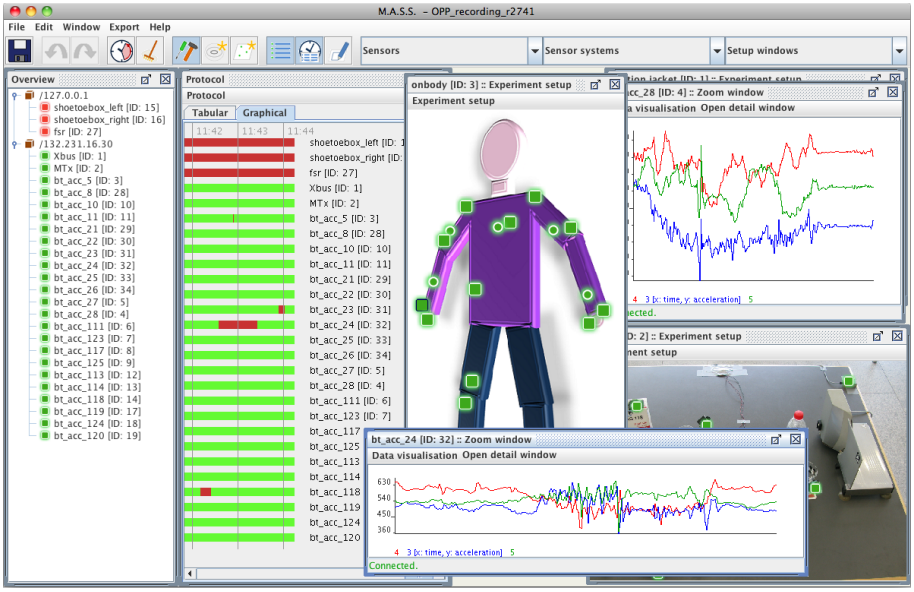


Figure 4.2: A screenshot of the monitoring application MASS, showing (from left to right) the list of monitored sensors, a timeline view visualizing sensor uptimes, on-body placement of sensors, live data stream plots, and placement of sensors on a table.

4.2.2 Data Management and Annotation Enrichment

Once sensor data sets are recorded they need to be maintained and prepared for use in the context recognition methods. Sharing of such data sets with the community may provide a common base for researchers to compare their activity recognition methods and it reliefs from the effort to record own data sets. A crucial point of an activity recognition data set is its ground truth annotation. Often it is not possible to do an exact online labeling during recording, e.g., when actions or gestures are short and suddenly occurring, when many actions are overlapping in time, or when unforeseen actions occur. In such cases careful re-labeling is inevitable. In this section we present three tools that help annotating, sharing, and preparing for utilization of sensor data recordings.

Context Database

Research on context recognition methods tends to require large amounts of real-life sensor data recordings. Yet, collecting complex data sets is time-consuming and needs a lot of effort. One way of reducing effort is sharing those datasets among research groups, which also brings the much desired advantage of having a common base for comparing different approaches.

For this we store the recordings in a database with web interface. We use the Apache CouchDB to implement the *Context Database* because of its scalability and its support for replication. Users may easily replicate parts of the database locally, e.g., for performance reasons, and still continue to work on exactly the same interface.

With the experience from past projects and experiments we structured the Context Database to store documents for the following entities:

Experiment A description and reference for the conducted experiment, hence the entity for a "data set".

Session A recording session within an experiment, aggregating a list of concurrent video streams, data streams, and label tracks together with their individual synchronization data. Many sessions may belong to the same experiment.

Data Stream An entity to describe the data stream originating from a single sensor or from a more complex sensor system with synchronized data channels. Channels inside a data stream all have the same sampling rate and are synchronized. The actual data is provided as data packets (see below) or in case of high sampling rate (e.g. audio) as an attached file.

Data Packet The data vector belonging to a timestamp for a data stream. It stores one value for each channel plus a timestamp, and it is referring the data stream it belongs to. It is analog to the data packets of the CRN Toolbox.

Video Stream An entity to describe a video track consisting of a video file. The video file is stored as attachment to this document and may be available in multiple encodings.

Label Track A description of a label track, defining the number and type of attributes per label.

Label A label instance in a label track, defining start- and end time, and the label's attribute value(s).

Sensor Type Description of a specific type of sensors used in the experiment.

Sensor Instance An entity to represent a specific instance of a sensor type. The instance may point to a list of "sub-sensor" instances in case the entity consists of multiple physical or logical components.

Sensor Placement Image Stores an image (attachment) and positions of sensor instances in the coordinate system of that image.

Using this database structure the data streams for a given recording session can easily be requested, in total or partially for a given time range, which might be desired for long recordings. The above set of entities proved to be sufficient for many data sets. However, future experiments and data sets may require a more refined structure, but with the underlying CouchDB, which is designed to allow for adaptations at any time, this is not an issue.

Web-based Database Browser

One of the key requirements for sharing large, multimodal data sets is an accurate description of the recordings, such that the community is able to perceive the full dimensionality of the recordings. Text documents and images can describe the recordings at different levels of detail, but obviously, the most detailed description is the data itself. This led us to the opinion that large, multimodal datasets should be browsable with little effort when shared with the community.

The support of HTML5 web standard in current web browsers allow for such browsing interfaces without the need of installing additional software.

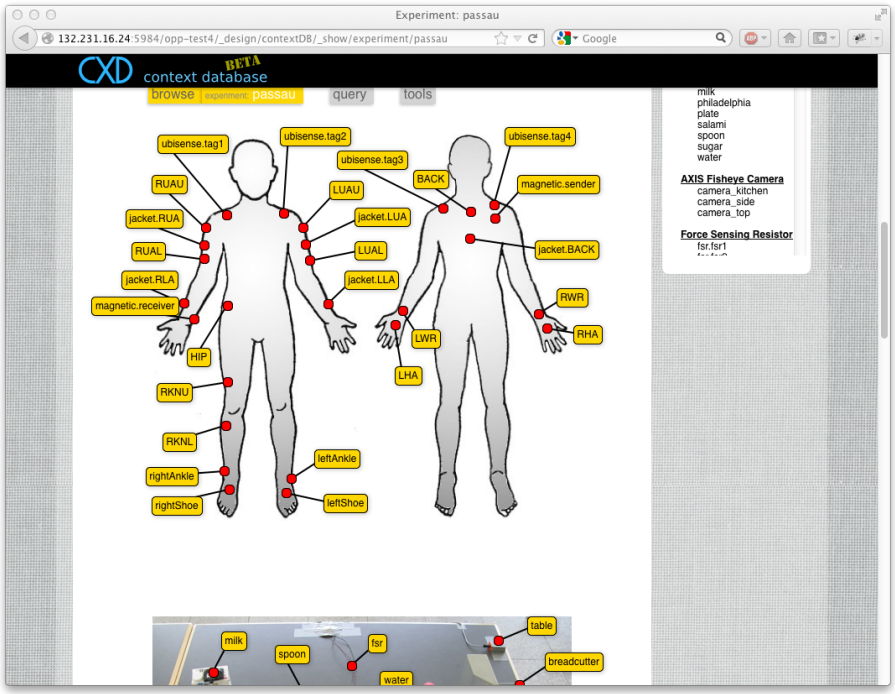


Figure 4.3: Screenshots of the Context Database web interface showing the placement of individual sensor instances within images.

We have implemented a web-based front end to the Context Database which gives access to:

- description of the stored data sets (experiments)
- description of the utilized sensor types in an experiment
- visual placement of sensor instances on images
- listing and summary of available recording sessions per experiment
- browsing of synchronized videos, label tracks, and data streams
- download of visualized data streams (export to ARFF format)
- querying for labeled segments and export to ARFF format

For each data set the user can find a textual description with images added. On each image the visible sensor instances are tagged to mark their placement in the specific setup (see Figure 4.3). Each sensor instance is linked to the respective sensor type, which is described on a separate page. On the sesor type description page all images with tagged instances of that type are automatically added. The experiment page also lists all available recording sessions with a brief

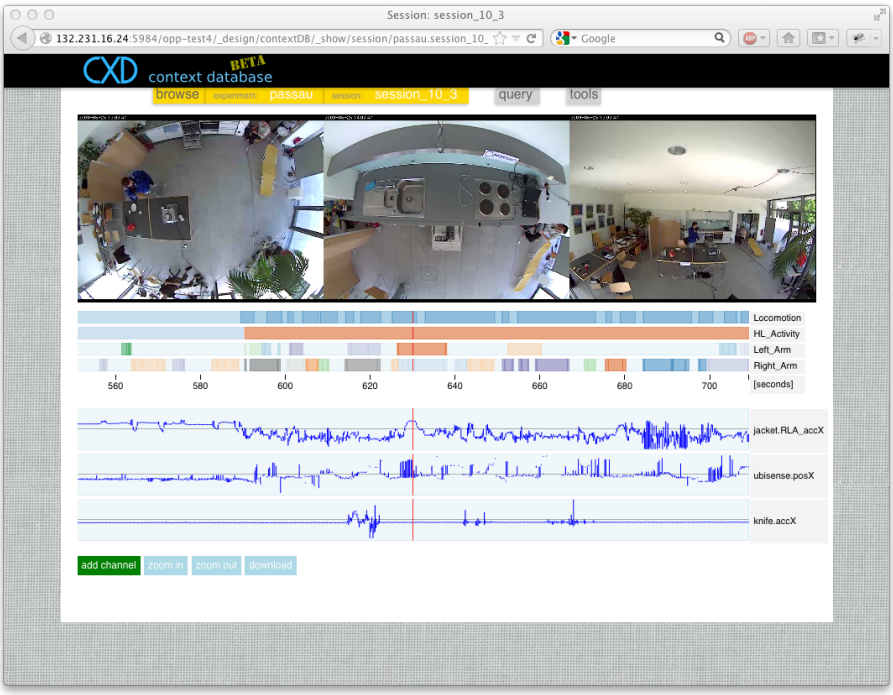


Figure 4.4: Screenshots of the Context Database web interface showing the data browser. Three video streams, color-coded annotations on four label tracks, and sensor signals from three selected channels are played back synchronously.

summary statistic of the respective video-, label-, and data tracks. Clicking on a listed session leads to the session view for browsing the data set.

The session view (Figure 4.4) by default shows all video streams synchronized with a timeline visualization of the label tracks (similar to the Labeling Tool described below). Multiple channels from the available data streams can be added by selecting them from the "add channel" menu. The menu will first list all available data streams, and after selecting one, a small preview of each channel of the selected stream will be shown. Zoom buttons allow to change the zoom level and the view can be panned with mouse drags (or swipes on a touchscreen device). Down-sampling of the data streams on the database side is implemented using the map-reduce paradigm [Dea08].

In the query view (Figure 4.5) the user can query the database for labeled segments and export them to an ARFF formatted file. The view allows to select

- all or a selection of experiments,
- all or a selection of sessions, and
- a label track

After selecting a label track the list of available labels loads. In case of two attributes (e.g. left- or right arm track with "action" + "object" attributes) a Sankey diagram is shown which visualizes the different compositions and allows

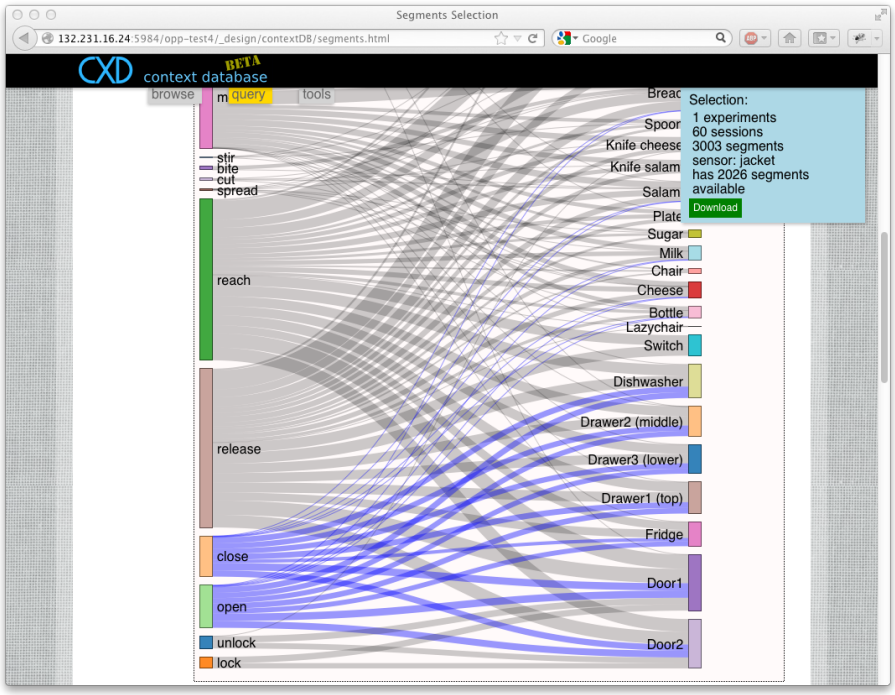


Figure 4.5: Screenshots of the Context Database web interface showing the data querying view. The Sankey diagram visualizes the label attribute combinations available for the chosen data set. Combinations may be selected individually.

for more intuitive selection. These segments can then be downloaded for each of the listed sensor data streams separately in an ARFF formatted file.

Reviewing and Labeling Recorded Sensor Data

Once a dataset is recorded it has to be prepared for utilization in training and evaluation of context recognition algorithms. For that purpose all sensor streams must be synchronized properly and the segments of interest must be determined and labeled. We developed the *Labeling Tool* for exactly this purpose. It synchronously displays several video- and sensor data streams in parallel alongside with several label tracks (see Fig. 4.6). The number and type of attribute in labels can be specified for each label track separately, fostering different granularity levels of the annotations. The Labeling Tool allows the user to

- synchronize video- and sensor data streams by aligning them in the timeline view,
- playback and seek all video-, label-, and data streams in parallel,
- assign labels with one or more attributes to selected sections,
- configure number and type of label tracks, and
- export data sets into flat, synchronized files for further exploration in tools like WEKA, Scipy, and Matlab.

It integrates with the tool chain by connecting to the Context Database for loading and saving the data sets, and by supporting, among others, the format of data files recorded with the CRN Toolbox. The Labeling Tool is implemented in Java.

We have used the Labeling Tool recently for labeling the data set described in [Luk10, Rog10] with more than 20 hours of recordings and over 70 sensors installed.

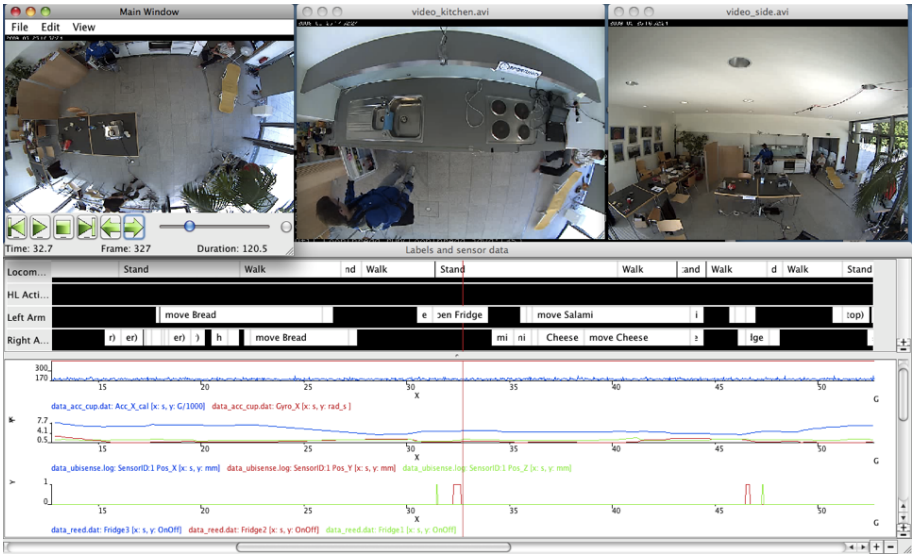


Figure 4.6: A screenshot of the Labeling Tool showing tree parallel video streams (top), four label tracks at different granularity (middle), and plots of several data streams (bottom) around the current master video frame.

Generating Data Traces from Recorded Data Sets

In order to reduce information loss inherent when transforming sensor data into the form needed by machine learning methods, we store only raw, unmodified sensor data in the Context Database. As many machine learning methods have very specific needs for the format and properties of their input data we developed the *Trace Generator* which transforms raw sensor data from the database into specific data traces for each method.

The Trace Generator is designed to select the necessary portions of data from the Context Database and to transform them into the format suitable for machine learning methods. Beyond the static training sets for training of context recognition methods it is capable of generating dynamic data traces with a story line of appearing, disappearing, rotated, or distorted sensors for training and evaluating opportunistic context recognition methods. It provides means to

- select specific recordings among all recording sessions stored in the database,
- select the desired sensors and channels,

- specify resampling options for each channel,
- merge (and synchronize) selected channels from different data streams,
- specify filters to apply on single channels,
- select and, if necessary, rename the labeled segments,
- specify transformations for some channels and time ranges, and to
- specify the output file format.

The traces are defined by a configuration script which allows for re-generating and adapting of the trace. Researchers can share Trace Generator configurations together with their methods to ensure reproducibility of their results.

The Trace Generator is implemented within the CRN Toolbox. In fact, the Trace Generator configuration is a Toolbox configuration file utilizing the specific reader tasks for accessing the Context Database and the desired filter tasks that can be timed according to the story line. Any other task that supports the desired transformation of data may be added as long as it does not depend on sources other than the data stream from the database.

4.2.3 Classifier Training and Online Context Recognition

The tools presented above provide all means necessary to prepare sensor data recordings for use in context recognition methods. At this stage (see Figure 4.1), the recorded sensor data is ready to be fed into scripts and applications for training and evaluating machine learning methods. There is a wide spread range of programming libraries and tools available that help processing and analyzing numerical data, and training and evaluating classifiers (e.g., WEKA, scikit-learn (Python), Matlab, just to name a few). For the purpose of online context- and activity recognition, we provide the CRN Toolbox presented in Chapter 2 which offers an efficient and flexible runtime environment for this purpose. The reader tasks involved in the recording of the data set (Section 4.2.1) and the transformation tasks used in the Trace Generator (Section 4.2.2) may simply be re-used in the configuration at this stage, providing the online recognition system with exactly the same data format and processing steps as were applied to the training data. In the simplest case, only the configured classifier and an appropriate writer task need to be added to the configuration.

In the remainder of this section we present an extension to the Toolbox which allows for dynamic sharing of sensor data and services.

A Service-Oriented Extension for the CRN Toolbox

Regarding context recognition in well-known and stable surroundings, the CRN Toolbox is capable of setting up such a scenario and delivering data back to an application instantly. However, despite the fact that Toolbox instances can be flexibly distributed amongst numerous devices (this includes mobile devices such as small wearable computers, phones, PDAs, laptops, as well as desktop computers, workstations, and servers) and include other applications into their data flow, a distributed setup is error-prone, as each Toolbox instance relies on a known and stable network topology due to its static configuration.

It is quite common in environments of mobile, networked devices that devices leave the current network permanently or are temporarily unavailable due to lost connections. Once they rejoin the system, they might have another IP

address assigned to its network interfaces. Furthermore, mobile devices often disable their wireless connection to preserve battery power, go into low-power modes or even shut down automatically once the battery is just about to run out. Moreover, new sensors or devices – possibly valuable to the application's mission – occasionally appear, such as an inertial sensor in a new shoe or a new edition of a smartphone offering better sensing and processing capabilities.

In these cases, the current Toolbox setup will either no longer function correctly, hence a manual reconfiguration of the setup is inevitable (i.e., changing the IP addresses in the configuration file), or the setup is just not optimal anymore and the configuration would need to be adapted. Consequently, many applications are hard to incorporate because of the static nature of the Toolbox configuration.

As a result of these observations, we enhanced the CRN Toolbox to support service-level interfacing of its instances and rise the system to a more abstract level of *services*. Consequently, we can now replace the static Toolbox configuration with dynamic ad-hoc recombination of Toolbox services. In order to achieve this, two major extensions need to be incorporated into the system, with service registration and service discovery being the first, and semantic annotations for these newly available services being the second.

We compared several options for the implementation of the service registration and service discovery layer: Universal Plug and Play (UPnP)[2], Service Location Protocol (SLP)[3], Jini[4] and DNS Service Discovery (DNS-SD)[5]. During our tests, DNS-SD outperformed the other protocols clearly. With every Toolbox implementing the DNS-SD protocol, it becomes easy for a third-party application to register and discover the available services. Basic service data like the name of the service's host or the port it is bound to, is provided automatically by the DNS-SD protocol itself. Service-specific data on the other hand, can be included during service registration. In our case, this service-specific data represents the current configuration of the Toolbox service, i.e., which tasks are loaded and how they are connected amongst each other. As the configuration of a Toolbox itself is semantically not meaningful enough and happens on a rather low abstraction level, we introduce a semantic layer on top of the Toolbox, that is capable of describing services.

Basically, two types of CRN Toolbox services can be distinguished: simple *sensing services*, that are recording data from sensors, and more complex *recognition services*, which are encapsulating a concrete recognition algorithm. These algorithms usually require a specific type of sensor, i.e., sensing service, since they rely on a certain data format, sampling rate and of course need appropriate data sets for them to work properly.

The description of sensing services is straightforward: it includes fields characterizing the sensor the service is working on, such as sampling rate, supported axes and the sensor's exact location, as well as the sensor service's output. The description of the recognition service contains the output of the service as well, but also defines a set of predefined taxonomies and rules that have to be fulfilled by any sensing service in order to be considered as a possible candidate, that could be linked to the recognition service.

[2]UPnP: `http://www.upnp.org/`
[3]SLP: `http://www.ietf.org/rfc/rfc2608.txt`
[4]Jini: `http://www.jini.org/wiki/Main_Page`
[5]DNS-SD: `http://www.zeroconf.org/`

With these extensions being made to the CRN Toolbox, dynamic ad-hoc recombination of CRN Toolbox services may take place. A context-aware application may for example browse for a specific recognition service which realizes exactly the task the application is looking for. After having found such a service, the service itself is able to find matching sensing services by reasoning over ontology rules. With a match being found, the sensing services are linked to the recognition service, which then initiates its algorithm and starts processing the input data and delivers the results back to the context-aware application for further processing steps. A comparable procedure has to be followed by the application in case it tries to find a suitable recognition service for a sensing service the application seeks to interact with.

Ultimately, the extensions described in this section allow for semi-automatic recombination of context-recognition tasks. Extending this work by making the step towards fully-automatic recombination might enable self-configuring and even self-healing sensor networks in the future.

4.2.4 Availability

The tool chain is freely available and the main components are being published as open-source projects. The Context Database is publicly accessible at `http://contextdb.org` and already contains a data set with more than 20 hours of recordings from over 70 sensor instances [Luk10, Rog10]. The site also links to all repositories of the tool chain's components.

4.3 Conclusion

We have presented an integrated tool chain to support all phases of the development of a context recognition system. The tool chain extends the basic concept of the CRN Toolbox presented in Chapter 2 to the entire process of developing context recognition systems:

- The recording of multi-modal data sets is supported by sensor reading tasks of the CRN Toolbox, dedicated logging apps for sensor-equipped smartphones, and a monitoring application to document the experiment progression and to early detect sensor failures.
- For storing, annotating, and sharing of recorded data sets the tool chain provides the browsable, web-based Context Database, a graphical annotation tool, and the concept of generating reproducible traces from selected parts of the database which can be used as base for classifier training and evaluation.
- Online context recognition from live sensor data streams is supported by the CRN Toolbox itself and a service oriented extension opens the way for dynamic sharing of sensor data and recognition services among multiple Toolbox instances.

The service oriented extension to the CRN Toolbox enables statically configured Toolbox instances to build dynamically composed systems at a higher abstraction level of sensing- and recognition services. This is an important foundation for building opportunistic activity recognition systems which dynamically

involve sensors and other resources as available. We focus on opportunistic activity recognition methods in Part II, where we investigate how a classifier can opportunistically take advantage of a sensor which was not available during training of the classifier.

The tool chain has been implemented, used in different projects, and is publicly available to the community. We hope that it will speed up the development of context aware applications, make it easier for new groups to deal with complex context recognition experiments, and foster the reuse and sharing of data sets.

The fact that structured and annotated context data is now accessible in an online database might enable future activity recognition systems to dynamically fetch training data for new classification problems, which were not known at design time. Other future work may address the integration of an algorithm- and results database into the system and continued improvements with respect to the stability and usability of the tools.

Part II

Opportunistic Methods for Activity Recognition

Chapter 5

Adding New Sensors to Activity Recognition Systems

The tools presented in Part I may facilitate the development of new online activity recognition systems. Still, such systems are tailored to a narrowly defined sensor setup and any change of that setup forces the system to be redesigned. Within this chapter we introduce a novel method to address the issue of long term system evolution by allowing new sensors to be integrated automatically without any user intervention (in Chapter 7 we consider efficient use of limited user feedback). The method bases on well established principles of semi-supervised learning, but instead of relying on a small set of user-labeled data we receive labeled instances from the initial classifier. More specifically, we apply unsupervised clustering to find structure in the new data, assume correlation of cluster membership with class membership, and use the initial classifier model to assign labels to clusters in the extended feature space. We study this approach on real world activity recognition data sets, discuss the factors that make a distribution difficult to handle, and elaborate heuristics that, in a majority of cases, are able to achieve an increase of recognition accuracy by the unsupervised integration of the new sensor. In the successive chapters the method is further refined.

David Bannach, Bernhard Sick, and Paul Lukowicz. Automatic adaptation of mobile activity recognition systems to new sensors. In *Workshop Mobile sensing challenges, opportunities and future directions at Ubicomp 2011.*

Today, state of the art approaches to activity- and context recognition systems mostly assume fixed, narrowly defined system configurations dedicated to often also narrowly defined tasks. Thus, for each application, the user needs to place specific sensors at certain well-defined locations in the environment and on his body. For a widespread use of such systems this approach is not realistic. As the user moves around, she/he is at some times in highly instrumented

environments, while at other times she/he stays in places with little or no intelligent infrastructure. Concerning on-body sensing, a user may carry a more or less random collection of sensor enabled devices (mobile phone, watch, headset, etc.) on different, dynamically varying body locations (different pockets, wrist, bag). Thus, systems are needed that can take advantage of devices that just "happen" to be in the environment rather than assuming specific well-defined configurations.

In previous work our group investigated how on-body position and orientation of on-body sensors can be inferred [Kun05, Kun09], how position shifts can be tolerated [Kun08], and how one sensor can replace another [Kun10]. In this thesis we address the question how an activity recognition system that has been trained on a given set of sensors can be enabled to use a new source of information that has appeared in its environment without any or with minimal user input.

5.1 Scope and Contribution

Clearly, a method that would *without any training* use information provided by an additional sensor and *always guarantee* to achieve a performance improvement is not feasible. Instead, we investigate an approach that sometimes provides an improvement and overall does "more good than harm". Thus, we claim that:

1. For sensors that provide useful information with respect to the classification problem at hand the method should achieve significant improvements in at least some cases.

2. The improvements should not come at the cost of performance degradation in other cases. This means that

   (a) averaged over all sensors the effect of the method should remain positive (we require at least a slight overall performance increase) and

   (b) no (or very few) single cases experience a significant (more than a few percent) performance decrease.

As we will discuss in Section 5.3.1, methods achieving the above goals can be fairly easily constructed for certain particularly simple and well-behaving problems. Thus, a classifier trained on one sensor ("old" sensor) can be extended to use a "new" sensor in four steps:

1. clustering within the joint feature space of the old and the new sensor,

2. identifying points within the clusters that can be labeled with high confidence without information from the new sensor,

3. transferring the labels from such points to entire clusters based on the assumption that cluster and class memberships are correlated, and

4. using the cluster labels to improve accuracy for points that can not be accurately classified with the old sensor alone (see Figure 5.1).

Obviously, the above method requires

1. a clear correspondence between class distribution and structure (as manifested through clustering),
2. a specific relationship between the old and the new feature space that ensures that at least some points within each cluster can be reliably labeled using the old classifier, and
3. the availability of a large amount of representative training data.

Unfortunately, in real life activity recognition systems (and many other problem domains) the above assumptions often do not hold. The core contribution of this and the two subsequent chapters is to demonstrate how the basic concept can be extended to be applicable even in such situations.

General Assumptions

In addition to the above assumptions we make the following general assumptions for the studies in this and the two subsequent chapters:

- We assume it is known which features to extract from a given sensor. It might be a set of features that are generally known to work well for the type of activities, a specific feature set provided with the sensor, or a set provided from a sophisticated, external feature selection method. If not stated otherwise we use the word "sensor" in this and subsequent chapters as synonym for "a specific feature extracted from a sensor".
- We assume that all classes which should be recognized by the system are known at design time of the first classifier and that at least some of those classes are not recognized perfectly with that classifier. I.e., we do not expect new classes to be discovered but to improve recognition accuracy of the known classes.
- We assume that the probability distribution of each class remains constant. More precisely, we assume that for each dimension of the initial feature space the class-specific probability distributions in the data recorded after a new sensor was added to the system are the same as within the initial training data. This means that the way a subject performs the activities and the setup of the initial sensors (e.g., position, orientation) must remain the same, even when a new sensor appears. In this sense we are not requiring more than traditional systems with fixed sensor setups do.
- We assume all sensors to be synchronized and all features to be available at the same rate. I.e., each data point has a known value for each dimension of the feature space. A method for synchronizing heterogenous sensor networks is presented in Chapter 3.

5.2 Related Work

The need for a large amount of annotated training data has widely been recognized as a major issue for the practical deployment of activity recognition systems. As a consequence, different approaches were studied to overcome this problem. One line of work looks at unsupervised discovery of structure in sensor

data (e.g., [Min07, Huy06]). Another attempts to develop activity models from online information such as a common sense database, WordNet, or general "how to" resources (e.g., [Wya05, Tap06, Che09, Sti08b]). Due to the nature of the data, such work has a strong focus on interaction with objects.

Beyond fully unsupervised approaches there has also been considerable interest in semi-supervised systems that limit the amount of data needed for training (e.g., [Sti08b, Gua07]).

The field of *transfer learning* covers the general problem of transferring knowledge learned in one domain to another domain where the task may also differ between the domains [Pan10]. Within this context, a domain comprises the feature space and a marginal probability distribution, and a task refers to the label space and the objective predictive function (the classifier). The special case – resembling the problem discussed in this article – where the source and target domains differ but the task remains the same is also referred to as *domain adaptation*. Daumé [Dau07] presented an approach for domain adaptation which takes labeled samples from both domains to construct an augmented feature space and use the result as input to a standard learning algorithm. Unlike to our approach, the paper requires both domains to have the same features and dimension. Duan et al. [Dua12] proposed a domain adaptation method for heterogenous feature spaces. They first project the source and target feature spaces into a common subspace on which they apply a standard learning algorithm. The authors achieved promising results on a computer vision data set with 800- and 600-dimensional feature spaces. In contrast, our work explicitly assumes the addition of additional sensors, i.e., the target space contains additional features and the common subspace is equal to the source feature space. Furthermore, the feature spaces in activity recognition scenarios usually contain much less dimensions as each dimension is related to a signal from a physical sensor and the features (derived from the signals) are often manually selected.

In [Dai07], Dai et al. applied co-clustering to learn a classifier for unlabeled documents of a target domain when only labels for documents in a (different) source domain are known. Co-clustering can reveal features that are similar and may help classifying items in different domains, yet it is not capable of using new features to improve classification.

Co-training [Blu98] is not applicable in our case because source and target feature spaces are not conditionally independent (target space contains all features of source space). Even for the additional features alone we can not assume conditional independence to the source features.

Closest to our work is [Cal10] which uses other sensors and behavioral assumptions to train a system to take advantage of a new sensor. Unlike our work it does not consider to add a new sensor to an existing system.

While related to our work at a methodological level, none of the articles mentioned above addresses the problem of integrating new sensors into an existing system.

5.3 Key Challenges and Ideas

We start by introducing the basic idea behind our approach, discuss the key challenges that arise from real world activity recognition data, and present an overview of the proposed method.

5.3.1 Basic Idea

In its core our method is a variation of well-established principles of semi-supervised learning:

1. Unsupervised clustering to discover structure in data.

2. Assumption that structure corresponds to class membership. In the simplest case this means that points in the same cluster belong to the same class.

3. Obtaining at least one correctly labeled data point for each cluster and using those data points to label the corresponding clusters.

In most semi-supervised approaches the labels are the ground truth provided by the user. Our approach uses a classifier trained on N features to provide labeled data points for a (N+1)-dimensional feature space. For the sake of simplicity and better understanding we consider the transition from 1D (a single sensor) to 2D (two sensors) space here.

As with most classification method the number of required training data points increases quickly with each additional dimension, a phenomenon known as *curse of dimensionality*. The clustering method we use in our approach is also affected by this problem, i.e., for taking the step from N to N+1 dimensions the clustering method needs to receive enough data points to produce reliable results in an (N+1)-dimensional feature space. For the rest of our method the absolute number of dimensions is less important as we only adopt the new dimension in those regions of the original feature space where the new dimension is more discriminative. For the purpose of human activity recognition the number of dimensions usually is limited by the number of physical sensors (and their features) available. In this work we concentrate on exploiting information from newly added sensors and we leave it open for future work to cope with the dimensionality problem, e.g., by discarding less discriminative dimensions or applying different clustering methods.

We start with a classifier trained on data from a single sensor (sensor 1). We assume that the classifier is performing reasonably well, but still has some regions in its feature space where the separation of classes is poor. For our method it is crucial that it is known in which regions of the feature space the classifier is confident and in which it has many misclassifications. In a tree classifier (which we are using for this work) this is trivially given if the purity of the leaves is retained from training.

At a certain stage a second sensor appears (sensor 2), that can potentially improve the overall class separation. We can now collect and cluster data points in the 2D space. The question is how to assign labels to the 2D clusters. The aim of our work is to do so without or, as discussed in Chapter 7, with minimal user input. Thus, we use the classifier trained on the 1D feature space to provide the labels. A naive method to do this proceeds as follows (see Fig. 5.1):

1. We project the 2D clusters onto the 1D space of sensor 1.

2. We consider regions of the 1D space onto which points from *only a single 2D cluster* are projected (single cluster regions). From the assumption about structure corresponding to class distribution we can conclude that

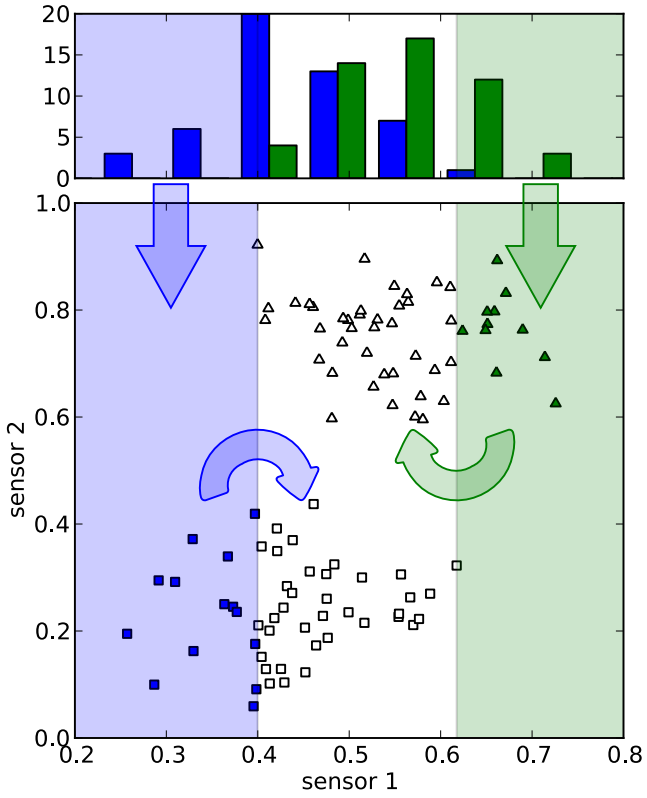


Figure 5.1: Naive method: histogram of a 1D feature space (top) and two clusters in a 2D space (bottom). Arrows indicate the propagation of labels.

the class distribution within those regions should be representative for the class distribution in the corresponding clusters.

3. We use the classifier trained on sensor 1 to determine the class distribution in the single cluster regions and use this distribution to label the corresponding clusters.

As can be seen in Fig. 5.1, the performance gain comes from clusters that are partially projected onto regions where the 1D classifier is confident and partially onto regions where it produces many errors. For the former, the 1D classifier provides labels for the 2D clusters. For the latter, the 2D classifier outperforms the 1D classifier.

Note that the basic method does not necessarily require a one-to-one correspondence between clusters and classes. As an example, consider a region that in a 1D space corresponds to a 50:50 mixture of two classes (see Figure 5.2). Now assume that in 2D that region corresponds to two clusters: one with a 25:75, and one with a 75:25 mixture of those two classes. It then follows that using the cluster information reduces the probability of misclassification from 50% to just 25% in that region.

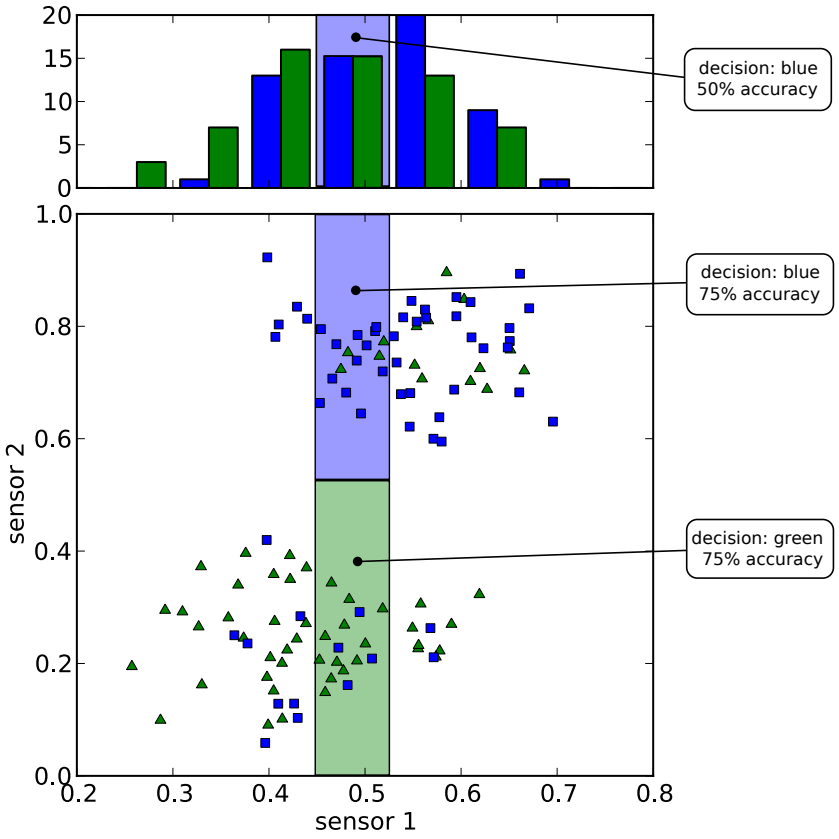


Figure 5.2: Example of accuracy increase by adding a decision boundary between mixed distribution clusters in 2D space.

5.3.2 Challenges

Clearly, the method works well on the sample clusters in Fig. 5.1 because the 1D and 2D distributions were carefully constructed to make it work. In most real data sets such ideal distributions are unlikely. When applied to these data the naive method described above is not able to achieve any improvement because the clusters that it can label using the "pure" regions of the classifier do not appear together with any "impure" region that could be improved. Furthermore, the variation of distributions between two samples (unlabeled 2D data, 2D test data) can be significant. This may even lead to a performance decrease.

In the remainder of this section we will systematically go through the challenges that a more sophisticated method has to face.

Distribution Issues

One distribution-related issue addresses the specific choice of a tree classifier. With the tree classifier we store class distributions at leaf level. That means that in order to be able to assign a class distribution to a cluster, that cluster must be the only cluster appearing in the leaf. Unfortunately, in real data we often have

the situation where two or more clusters are projected onto distinct areas of the 1D space, but these areas fall within the same leaf making class distribution assignment impossible. To address this problem, we create an additional tree in the 1D feature space using purity with respect to cluster membership rather than class membership as splitting criterion. We then use the original training data for the 1D classifier to assign class distributions to the new leaves which are in turn used to assign distributions to the clusters.

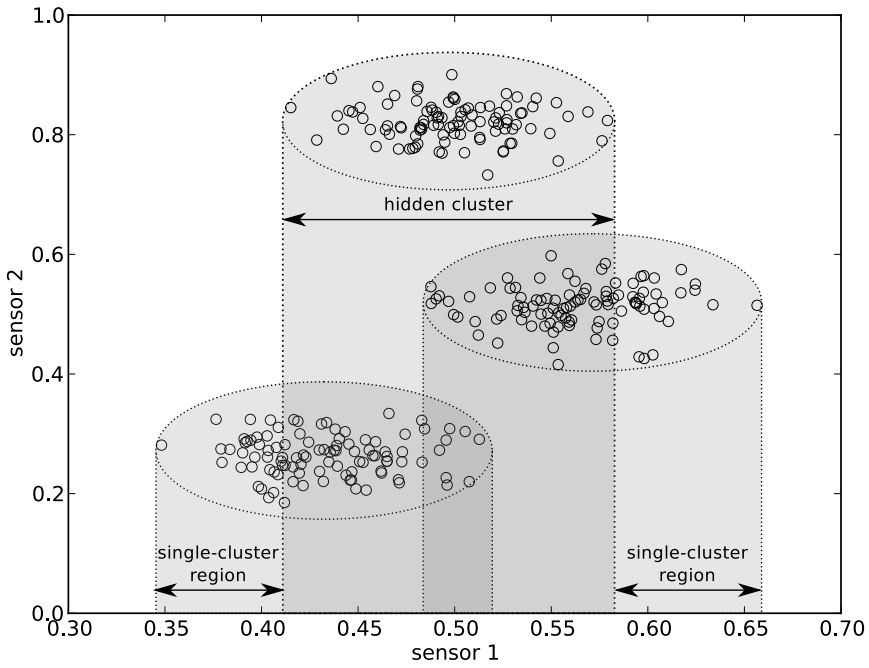


Figure 5.3: Effects of projecting 2D clusters onto 1D space.

The basic method requires that each cluster has a region where, when projected onto the 1D space, it does not overlap with any other cluster. Unfortunately, this is not always the case. Some clusters can be hidden, as shown in Figure 5.3. To assign labels to such hidden clusters we propose a "similarity search" in Chapter 6. To this end we first assign labels to all non hidden clusters. We then attempt to assign labels to the hidden clusters in such a way that, when the labels are propagated to the cluster's data points, the projection of all data points onto 1D space approximates the original 1D distribution (as trained for the 1D classifier) as well as possible.

Training Sample Quality and Size

All machine learning algorithms build on the assumption that the training data are reasonably representative and break down if that assumption is substantially violated. In activity recognition (as in many other fields) training data variance is often a problem because (1) it is costly to obtain a large amount of labeled data from users and (2) there are often large variations in the way

humans perform even fairly simple actions. At the same time our method is particularly sensitive to the quality of the training data, because it leverages small areas of the feature space (regions of the 1D space onto which only parts of a single cluster are projected) to label larger ones (entire clusters). Using small areas to derive labels for larger ones effectively means artificially reducing the sample size and thus making the estimator more sensitive to sample variance. In practice, we would often use 10% and less of the data for cluster labeling. If that 10% of the training data happens to have high variance and if they are not representative, then the addition of the new sensor will not only fail to improve performance, it may actually drastically reduce the classification accuracy over the entire feature space. As described later we have found such "erroneous" configurations to be a significant problem in real-world activity recognition data. In fact, if no additional measures are taken, the risk of ending up with erroneous configurations that worsen the recognition performance nearly neutralize the potential improvement from the integration of an additional sensor. To solve this problem we have developed three approaches for sorting out configurations that have a high probability of leading to performance degradation.

First, for every assignment of class distributions to clusters we compute *plausibility* and *gain* values. The plausibility is based on the projections of the labeled clusters onto the 1D feature space. It compares the class distribution that results from the projection to the class distribution in the original 1D training data, and thus offers a measure of how plausible the guessed labels are. The gain compares the "purity" of the labeled clusters to the "purity" of the corresponding leaves of the original classifier. When weighted with the number of data points in the respective region it is an estimation of the potential that the particular cluster labeling has for improving the system performance. The larger the discrepancy between the two measures, the more likely it is that the specific cluster labeling has been caused by sample variance and does not correspond to the true class distribution.

Overall, gain and plausibility proved very useful in deciding whether it makes sense to use a particular cluster labeling or not. However, they are just estimates and it is well possible that a solution with high plausibility and high gain is distorted by sample variance and will worsen rather than improve performance (as compared to the 1D classifier). In Chapter 6 we rely on bagging as an additional measure to address this problem. That is, we derive the cluster structure and their class distribution for different subsets of the training data created from the original set through randomly leaving out and replicating individual samples [Bre96]. We consider clusters and class distribution assignments that significantly vary between the subsets to be unstable with respect to training data variance and penalize their plausibility value.

As a final measure we consider user-provided additional labels to detect erroneous configurations in Chapter 7. We apply both, the original 1D classifier and the 2D classifier generated by our method, to new data points and compare the output. If the output differs, then we ask the user to provide a label to check if the new classifier would improve (if it was right) or worsen (if it was wrong) the performance (cf. selection strategies for stream-based active learning). We reject the new classifier as erroneous (and stay with the initial one) if it makes more than a pre-defined number of mistakes within a certain number of data points. When looking at points where the classifiers differ, even only two or four user-provided labels have a very high probability of spotting erroneous configurations

with little (but non-zero) probability of mistakenly rejecting ones that improve classification rate by a relevant factor.

Clustering Issues

In Fig. 5.1, the optimal clustering is obvious. However, even in such a simple case standard clustering algorithms would only return the "optimal" structure if they were provided with the correct number of clusters and managed to avoid suboptimal configurations related to local minima. In general neither can be assumed. We rely on a hierarchical clustering algorithm to provide a set of possible configurations. We generate cluster labels for each configuration and use the plausibility and gain metrics to select the most promising one.

5.3.3 Method Overview

We base the work described in this chapter on a tree classifier. In a tree classifier each region of the classifier is represented by a single leaf of the classifier tree and its ancestor nodes define the region's boundaries (hyperplanes, each perpendicular to one axis). Additional dimensions can easily be added by simply replacing a leaf with a new subtree which adds boundaries on the extra axis without affecting other regions.

From the description above it can be seen that our base method consists of two stages, as shown in Figure 5.4a:

1. Training and using a 1D tree classifier on sensor 1 in a standard supervised manner.

2. Collecting and clustering 2D data from sensor 1 and sensor 2 and using the classifier trained on sensor 1 to label the 2D clusters. This is the core part of our method.

It is only after all of the above has completed that the 2D classifier is ready for use. The core part (point two) consists of the following sub-stages, as illustrated in Figure 5.4b:

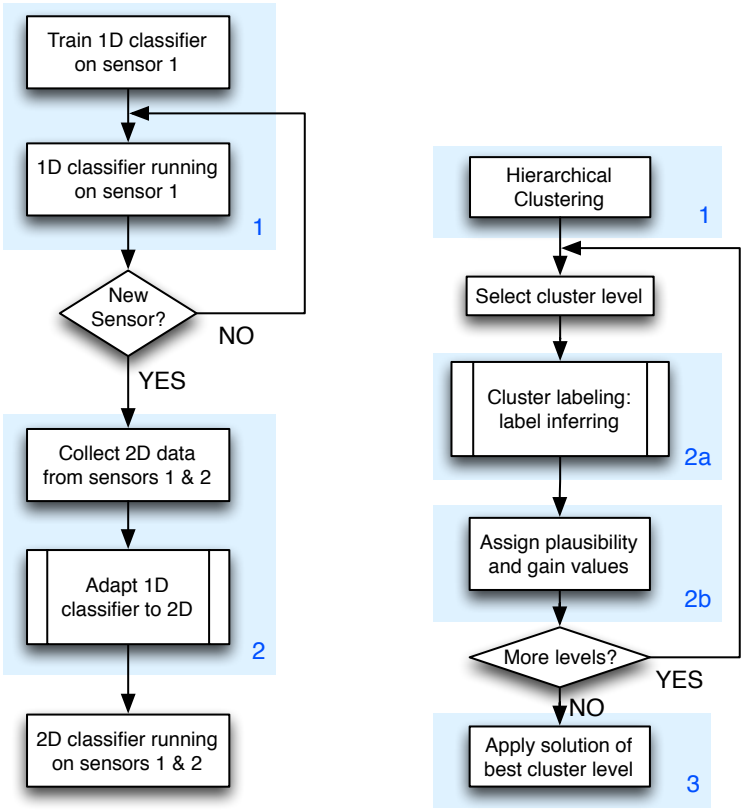
1. Running a hierarchical clustering algorithm on the 2D data.

2. For each plausible cluster configuration:

   (a) Label inferring: assigning class distributions to clusters using those areas of the 1D feature space onto which only a single cluster is projected (Section 5.5). This involves creating a new 1D tree trained according to cluster rather than class membership (Section 5.5.2).

   (b) Assigning plausibility and gain values for the resulting class distribution assignment for each region of the 1D classifier (Section 5.5.4).

3. Selecting the clustering and label assignment that have the plausibility and gain measures best matching a predefined policy.



(a) Overview (b) Adapt 1D to 2D

Figure 5.4: Overview of the overall procedure (a) and its most important step, the classifier adaptation (b).

5.4 Evaluation Strategy and Data Sets

We follow two different strategies for evaluating our method. First, we evaluate it on several synthetic data sets that allow us to systematically study the influence of different, specific characteristics of the data set on our method. We then proceed to investigate the performance on real world data sets previously used in different published activity recognition experiments.

5.4.1 Synthetic Data

The synthetic data sets contain four equally distributed classes (bivariate normal distribution with diagonal covariance matrix, i.e., the two dimensions are uncorrelated). To better assess the influence of the number of unlabeled instances we investigate the performance of our method on two different sizes of the synthetic data sets. In the first we generate 50 unlabeled 2D instances per class (the instances used for adapting the classifier), and in the second only 10 instances per class, which is closer to the real world data we use later. In both

cases we generate twice the amount of instances each for training and testing (for real data sets the testing set has the same size as the adapting set).

Gradually decreasing cluster overlap: In the first synthetic data set we initially place the four clusters vertically stacked in the 2D space such that they completely overlap in the first dimension but are completely separated in the second (vertical) dimension as displayed in Figure 5.5. This means that the second sensor provides perfect information for classification while the first sensor initially can not help discriminating any class. In subsequent iterations we gradually move the class centers apart along the first dimension such that they end up in a diagonal layout with no overlap in any dimension.

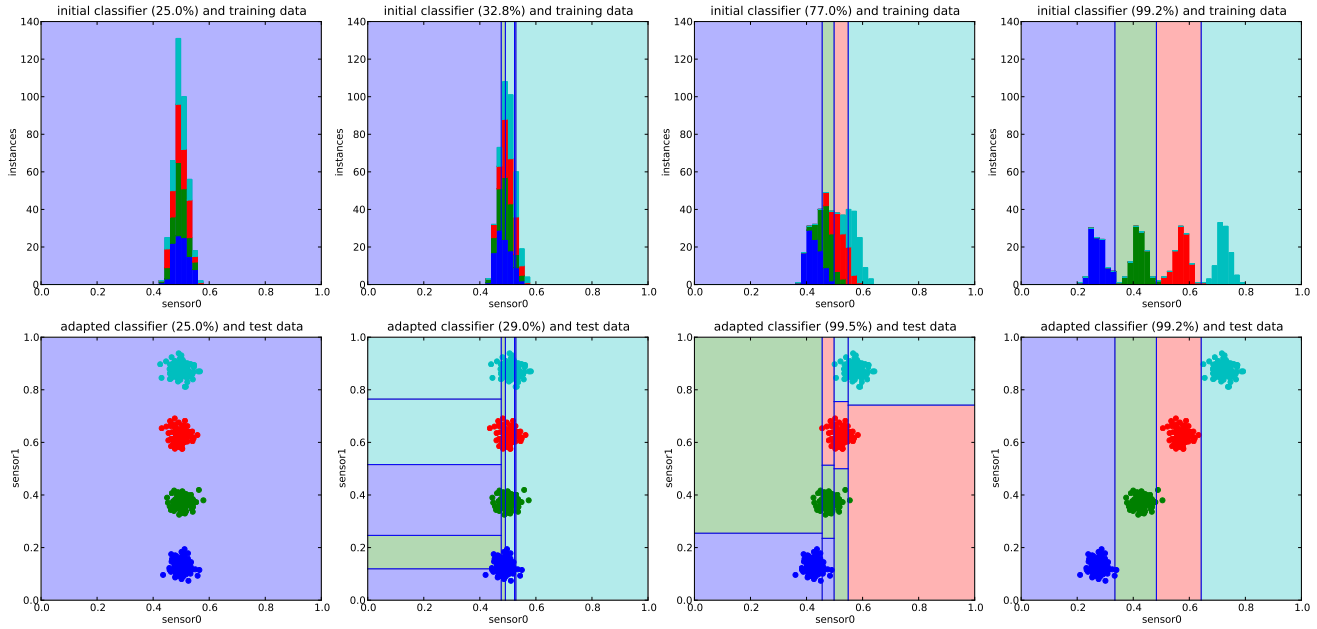
Moving a single cluster: In the second synthetic data set we position three clusters diagonally in the 2D space such that they slightly overlap in both dimensions (see Figure 5.6) and we move the fourth cluster vertically centered along the horizontal axis. While iterating through different positions of the fourth cluster, the degree of overlap and the number of clusters that overlap with the fourth cluster are varying from no overlap at all through partial overlap to complete overlap in both dimensions.

5.4.2 Real Sensor Data

We evaluate our method on data from previously published data sets: a bicycle repair data set [Ogr05], a car maintenance data set [Sti08a], and the OPPORTUNITY data set [Rog09]. Over the three data sets we study 768 different sensor combinations from 13 users executing 22 different activities (8 in the OPPORTUNITY, 9 in the bicycle, and 5 in the car maintenance data set) with over 3000 individual activity instances. The sensors in question have widely varying recognition rates: from 21.89% to 94.15% with a mean of 51.05% for an individual sensor and from 20.26% to 97.66% with a mean of 69.25% for a combination of two sensors trained in a fully supervised manner as baseline.

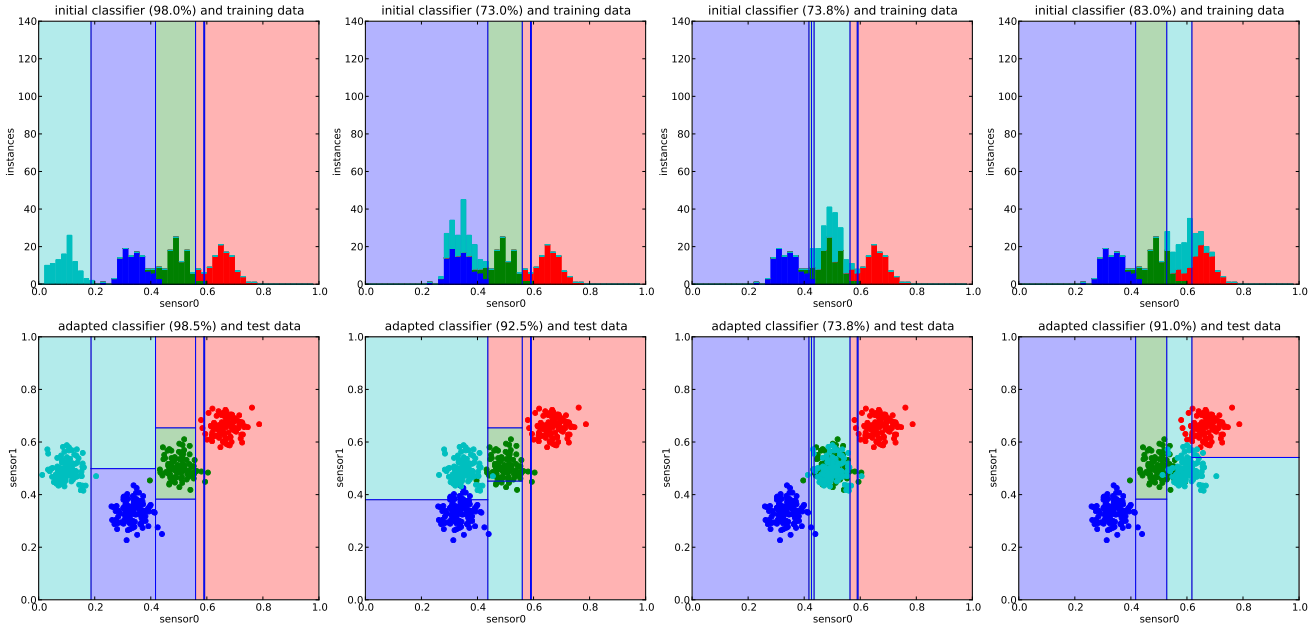
The evaluation focuses on basic manual actions and on body sensors. The choice of such actions rather than complex situations (e.g. "breakfast") is motivated by the fact that complex situations can be decomposed into such simple actions so that handling and understanding those is an obvious start. Within the above broad activity class, data sets were chosen to provide actions that cover different extremes. While the bicycle task has fairly complex, longer actions (e.g., taking out the seat, disassembling the wheel), the car maintenance and OPPORTUNITY data contains mostly different types of very short open/close gestures (see below).

As features on the sensors described below we allow only mean and variance. This way we decouple the problem investigated in this work from the issue of feature choice which can be complex and have big influence on the recognition. At the same time both features are commonly used in motion-related activity recognition. Looking at the ability to just add a sensor that appears in the environment to a recognition system using simple features also makes sense. Given an unknown sensor, the system would certainly not be able to identify sophisticated features and would have to work with what is known to work reasonably well across many sensors. In the evaluation we handle both features separately, as if they were separate sensors.



(a) dist $= 0$ (b) dist $= 0.4\sigma$ (c) dist $= 2\sigma$ (d) dist $= 6\sigma$

Figure 5.5: First Synthetic Data Set—Gradually Decreasing Cluster Overlap: Input data and example results for four different setups of synthetic data with decreasing amount of cluster overlap in the first dimension. Top: histogram of the labeled 1D training data overlaid on regions of the initial decision tree classifier. Bottom: test data overlaid on regions of the resulting adapted decision tree classifier.



(a) posX = 0.1 (b) posX = 0.35 (c) posX = 0.5 (d) posX = 0.6

Figure 5.6: Second Synthetic Data Set—Moving a Single Cluster: Input data and example results for four different setups of synthetic data with differing amount of cluster overlap in both dimensions by moving one cluster horizontally. Top: histogram of the labeled 1D training data overlaid on regions of the initial decision tree classifier. Bottom: test data overlaid on regions of the resulting adapted decision tree classifier.

The Bicycle Data Set

The data set is a simulated bicycle repair task where the subjects were given broad instructions what to do and a lot of freedom in performing the activities. We distinguish 9 classes which are

- pumping the front wheel,
- pumping the back wheel,
- screw/unscrew 3 different screws,
- testing the pedals,
- disassemble/and assemble the front wheel,
- test the back wheel, and
- disassemble/assemble the saddle.

The subjects have inertial sensors on different body parts and the position of the hands with respect to the bicycle is tracked with an ultrasonic system. For the evaluation we have used the mean of right hand (1) position x coordinate, (2) euler psi and (3) euler theta angles (4) and the y component of acceleration. For (5) the right hand gyro x component and (6) left hand gyro z component we have used the variance. Finally we have used the mean of left hand (7) y and (8) x position component. The sensor channels were chosen by training a tree classifier on mean and variance of all sensor channels from the left and right hand and then retaining those that the tree used.

The OPPORTUNITY Data Set

The data set is a simulated morning scenario where a person gets up prepares and has breakfast and then cleans up. It has been recorded with a large number of on body and ambient sensors (over 160 channels) to enable the comparative study of different sensors. From the data set we have chosen the following 8 classes:

- opening and closing gestures for
  - fridge,
  - dishwasher,
  - 3 different drawers,
  - 2 different doors, and
- the action of wiping the table.

These are much more subtle than the complex long bicycle actions and some are very specific to some sensors (e.g. position to disambiguate the different doors). Within the data set it can be shown that this type of basic actions is what more complex activities can be built of. We have used the mean of the following signals: x, y, z component of the right lower arm $(1, 2, 3)$ and back $(4, 5, 6)$ acceleration, as well as $(7, 8)$ x and y component of overall shoulder position (from a UBISENSE tag [Ste05]). In addition the variance of right lower arm

x, y, z acceleration $(9, 10, 11)$ and the vector of reed switches in different drawers (12) were used. Here, the sensor selection was based on "expert knowledge" and experience with the data set rather than on a systematic feature selection. We aimed to have sensors pairs that sometimes make sense together and sometimes not, as well as good and poor sensor for all actions.

The Car Maintenance Data Set

The data set is recorded in a real life scenario of car manufacturing industry. It covers tasks of an employee at the last checkpoint in the car manufacturing chain where all doors and the trunk are inspected for proper function and alignment. From this data set we select to distinguish 5 classes which are

- opening or closing the car's hood,
- opening, checking, or closing the trunk lid,
- opening or closing the front left door,
- opening or closing the rear right door, and
- opening or closing the fuel filler cap.

The subjects were wearing a "motion jacket" equipped with inertial sensors at each upper body limb segment and on the back, and UBISENSE tags to measure shoulder position within room coordinates. For our evaluation we selected mean of shoulder horizontal position x and y component $(1, 2)$, and from the back sensor mean of magnetic field y and z component $(3, 4)$ and variance of the magnetic field y component (5), and mean and variance of the right hand sensor acceleration x component $(6, 7)$. Similar to the above data set we selected the sensors based on "expert knowledge" to arrange combinations of sensors that are known to work well together and some that have poor contribution to the classification.

5.4.3 Evaluation Methodology

We evaluate our method on each combination of two different sensors; we train the base recognition system on the first and then integrate the second one as the "new sensor". As mentioned earlier, we treat each feature as a single sensor in this evaluation. Thus, we also consider improving a classifier with another feature of the same sensor or even with the same feature on a different axis of the sensor. Furthermore, this includes cross-modality situations where a system trained on one sensor (e.g., accelerometer) is improved with a sensor of a completely different modality (e.g., radio frequency positioning).

In order to avoid improving only the shortcomings of a badly trained recognition system we chose the parameters for the base classifier (minimum number of instances per leaf) individually for each combination such that the base system and the comparison system trained on both features perform best on the training set.

For evaluating our method we select the C4.5 decision tree classifier (WEKA J48 implementation [Wit05] with binary decisions) as base classifier. We normalize features from real data sets to the unit interval $[0, 1]$. For each evaluation

we need three separate data sets: for *training*, *adapting*, and *testing*. The training data set is needed for training the initial classifier. It uses only one feature and labels. The data set that we need for adapting the original classifier has two features but no labels, and the data set we use for testing the performance has both features plus the labels. If not stated otherwise, we set the size of the adapting and test data sets both to a half the size of the training data set. We shuffle the data set carefully before splitting to achieve similar class strengths per split and we repeat the evaluation 5 times with different seeds to avoid focusing on artifacts caused by a specific splitting choice.

5.4.4 Metrics

Obviously, the relevant performance measure is not the absolute recognition accuracy but the difference in recognition accuracy between the original system and a system with an additional sensor integrated according to our unsupervised method. Thus, for each sensor combination we compute the difference between the two accuracies (each expressed in percent) and refer to it as *accuracy change*. The accuracy change itself must be seen in the context of the amount of information that the additional sensor can provide. This amount is best quantified by looking at the classification accuracy when a dedicated classifier is trained in a fully supervised way on a feature space that includes the new sensor.

We consider the histogram of accuracy change that was achieved with each evaluated sensor pair (see Figure 5.9c for example) and look at those sensor pairs for which our method realized an improvement (accuracy change > 0) compared to those that ended up with less accuracy than with the first sensor alone (accuracy change < 0). The first measure to consider here is the **absolute number** of sensor pairs that ended up with positive or negative accuracy change. While this measure gives a good estimate for the chance of success it does not quantify the actual amount of improvement vs. "harm". The latter is better described by the overall **sum of positive accuracy change** and its counterpart, the **sum of negative accuracy change**. Additionally, we highlight the **median** and 10th- respectively 90th **percentiles** for positive and negative change.

5.5 The Label Inferring Method

We begin by focusing on the inference of cluster labels from single-cluster regions within the projection of 2D clusters onto the 1D space of the original sensor. In Fig. 5.4b this corresponds to steps 1 (clustering), 2a (cluster label inferring), 2b (plausibility and gain computation), and 3 (taking the solution with best gain/plausibility ratio).

5.5.1 Finding Structure in Extended Feature Space

We assess the structure of the extended feature space by applying an agglomerative, hierarchical clustering algorithm (Ward's minimum variance method [War63]). The resulting cluster tree gives us the possibility to explore the structure at different levels of detail without deciding on the number of clusters yet. We traverse the cluster tree top down, starting with two clusters and splitting one cluster at each step until a maximal number of clusters is reached. At each

step we attempt to (1) infer a label distribution for each cluster based on information from the original classifier model, and (2) build an adapted classifier model on the extended feature space, utilizing the label information of the clusters. All those adapted models are then rated based on the possible gain of classification accuracy and the best one is selected.

The method of rating different solutions not only allows for automatically finding an appropriate number of clusters but also makes it possible to choose the most helpful among different features available for a sensor, or even among different sensors.

5.5.2 Inferring Cluster Labels

As described in Section 5.3.1, the basic method for assigning labels to clusters is based on projecting the clustered data points back onto the original feature space. We then identify regions of the original feature space where (1) a single cluster is projected onto and (2) where multiple clusters are projected onto. We refer to them as *single-cluster* and *multi-cluster* regions (see Figure 5.3). In single-cluster regions the class distribution found in the original model (or in the training data) for that region is directly related to the cluster and may be used to infer the cluster label distribution. In multi-cluster regions, however, it is not clear how the class distribution in that region is correctly partitioned among the involved clusters, unless the distribution is pure, i.e., contains a single class only.

The problems that need to be solved are, thus,

1. delimiting the single- and multi-cluster regions, and

2. transferring the class distribution from the original classifier onto the single-cluster regions.

The first problem, distinguishing between single- and multi-cluster regions, is in essence a classification task by itself. We found the C4.5 decision tree classifier to be a good choice for solving this problem. The amount of pruning (B = minimal number of data points in a region/leaf) controls the granularity and stability against noise. Thus, for the actual process of cluster labeling we project the clustered data points onto the original feature space and train a C4.5 decision tree based on cluster membership. All regions (leaves) in this tree with a purity above a preset *purity threshold* are treated as single-cluster regions. 95% proved to be a good choice for the purity threshold in our data sets as it yielded a reasonable amount of single cluster regions while keeping labeling errors low.

For label assignment we then have to go back to the training data used for the original classifier. We use it to estimate the class density in those leaves of the tree trained on the clusters that correspond to single-cluster regions. In a final step, the label distribution is transferred to the corresponding clusters and normalized.

Note that the above means that we need to retain either the original training data or at least an adequate representation of it to be able to extend the system with a new sensor using our method.

5.5.3 Adapting the Classifier Model to the Extended Feature Space

Now, that the clusters have label distributions assigned to all clusters that appear in single-cluster regions, they can be used to extend the classification model with the new dimension. Obviously an extension only makes sense for the multi-cluster regions of the original feature space, since in the single-cluster regions the class distribution is identical in the new and in the old feature spaces. Furthermore, the extension is only possible for those multi-cluster regions, in which all clusters have been labeled. As explained in Section 5.3.2 and Fig. 5.3 some clusters may be hidden with no points in the cluster being projected onto a single cluster region (which is the prerequisite for the ability to assign a class distribution to a cluster).

Thus, multi-cluster regions in which all clusters have been labeled are split according to the cluster membership of the points. For a decision tree classifier this means that each leaf of the original classifier which falls into such a multi-cluster region is replaced by a subtree with each leaf of the subtree corresponding to a different cluster. Multi-cluster regions which contain a hidden cluster, i.e., a cluster that can not be labeled with this method, must be ignored and can not contribute to improving the classifier.

5.5.4 Estimating Plausibility and Gain

In essence, the above method amounts to generalizing the class distribution from a small subset of points (the points that project onto single-cluster regions) to the entire cluster from which the subset has been taken. For such a generalization becoming valid two conditions must be fulfilled.

1. The class distribution must be homogeneous throughout the cluster, i.e., when split into separate regions, the class distribution must be the same in each region. Note that it is not necessary that all points within the cluster belong to the same class (although it is certainly desirable). Instead the probability of a point belonging to a given class must be (approximately) the same everywhere within the cluster. The main questions are

   (a) does the new feature space contain enough correspondence between structure and class distribution to find clusters with homogeneous class distribution? and

   (b) is our clustering algorithm able to reveal those clusters in completely unlabeled data?

2. The subset of points that we use for labeling must be a good representation of the class distribution within the cluster. The main concerns are the sample size in relation to sample variance and the actual homogeneity of the distribution within the cluster. We may thus envisage a situation where, overall, the class distribution within a cluster is homogeneous enough to justify assigning a single distribution to the cluster, but between the small single-cluster regions, from which we collect the labels, the variations would be considerable. Clearly such regions should not be used for the derivation of the class distribution.

Obviously, if the sensor used to create the new feature space contains additional information about the classification problem at hand, it is well possible that the above requirements are (at least to a certain degree) fulfilled. However, this is by no means guaranteed. Thus, for example, there is no fundamental reason why an additional sensor should not result in a distribution that has two spatially well separated clusters, but where the separation boundary between classes is located within the clusters instead of between them. Furthermore, even if in the new feature space there exists a cluster structure that satisfies the above conditions, there is no way to guarantee that our clustering algorithm will find it.

Unfortunately, applying our method to a distribution that does not satisfy the above conditions in general leads to a significant performance decrease, not just lack of improvement! At the same time the proposed system should be able to deal with an arbitrary additional sensor without prior knowledge about the class distribution in the new feature space. The problem herewith is, that for a given clustering and label distribution in the new feature space there is no way to *exactly* determine (without additional user provided labels) the quality of the label assignment. As a consequence our system relies on the following heuristic:

1. A method for the estimation of the *plausibility* of label assignment.
2. A method for the estimation of the potential *gain* of accuracy that the new dimension may bring.
3. A risk-benefit analysis to decide a given label assignment is worth using.

Plausibility

The derivation of *plausibility* is based on the observation that, while the inclusion of the new sensor changes the classification of individual data points, the overall class distribution within each region of the old feature space should remain unchanged. Thus, we propose to take a set of data points from the extended feature space and

1. classify them using the new (extended feature space) decision tree,
2. project them onto the old feature space, and
3. for each region in the old feature space:
   (a) compute the resulting class distribution in the old feature space and
   (b) compare it to the class distribution in the original training data on the old feature space.

If the labels assigned to the clusters in the new feature space are correct, then the two distributions (point 3a and point 3b above) should be identical or at least very similar. Therefore, we derive the plausibility value for each region r from this similarity as

$$\mathsf{plausibility}_r = \frac{1}{2}\sum_{i=0}^{N}(\hat{p}_{r,i} - p_{r,i})^2,$$

where $p_{r,i}$ is the probability of class i (estimated using the training data) within region r and $\hat{p}_{r,i}$ the probability of class i in region r approximated with data in the extended feature space when classified with the adapted classifier. The plausibility is normalized by the maximal possible distance of the two class distributions.

Note that the plausibility value is just an estimation. If the value is high, then, within sample variations, we can be sure that the label assignment in the new data space is not useful. On the other hand, when it is low, then it is likely but by no means assured that label assignment is approximately correct.

Gain

As a second measure we estimate the potential classification accuracy *gain*. Since we consider a tree classifier this can be defined as an increase of purity that we achieve by extending the classifier with the new sensor. As described in Section 5.5.3 we create the extended classifier by splitting leaves of the old classification tree according to membership in (labeled) clusters of the new feature space. For each leaf of the old tree (i.e. for each region r in the initial feature space) we can compute its purity from the training data as

$$\mathsf{purity}_r = \max_i(p_{r,i}).$$

In the best case the new leaves in a region are all pure and hence the new purity would be 1. However, if a new leaf would still classify some points incorrectly (which we can not test without ground truth data) the relative size of the leaf (in terms of amount of data points falling in that leaf) would be bigger than the probability of the corresponding class in the original training data within that region.[1] Hence, the purity contribution of a new leaf can maximally amount to the probability of its corresponding class in the original training data. Thus, we obtain the purity resulting from the new leaves, in each region, from the histogram intersection of the new class distribution (point 3a above) with the initial class distribution (point 3b above)

$$\widehat{\mathsf{purity}}_r = \sum_{i=0}^{N} \min(\hat{p}_{r,i}, p_{r,i}),$$

and we finally obtain the gain value from the difference of both purities

$$\mathsf{gain}_r = \widehat{\mathsf{purity}}_r - \mathsf{purity}_r.$$

The gain value is computed separately for each region of the original classifier and is summed up to a weighted average for the whole solution. The weights are the relative amount of training data points in each region.

Note that the gain computation is based on the assumption that the label assignments in the new feature space are correct. In this sense it is, just like plausibility, an estimation that may or may not be correct.

[1] In case multiple new leaves contribute to the same class they are considered together.

Alternative Measures

We chose the above two measures by selecting the formula which complemented our method reasonably well to demonstrate its feasibility and which delivered promising results on test data. An alternative approach for computing the gain value, e.g., could be to rely on the information gain measure inherent with the C4.5 decision tree instead of purity, or computing the new purity from the cluster label distributions instead of the histogram intersection. We, however, preferred the selected variant because of the following properties: The computation of the plausibility and gain values using data points classified with the new decision tree (instead of using the cluster label distributions directly) allows to reflect the quality of the actual decision tree, i.e., it includes the effects caused by the actual added decision boundaries and by the leaf's decision for the single (most probable) class. The former manifests in case the clusters should not be perfectly separable along the new dimension within a region, and the latter allows for different cluster labeling methods such as presented in Chapter 6.

Deducing the new purity from the histogram intersection distance is an optimistic estimation. It benefits mixed clusters because they can cause a higher purity than actually possible. For the example displayed in Figure 5.2 we would obtain a new purity of 1 for the highlighted region (a gain of 0.5) instead of 0.75 (gain 0.25). This is, however, only a problem when another clustering and cluster labeling would result in a better (e.g., nearly perfect) classifier but would not be selected because of a lower gain.

Using the Plausibility and Gain Values

Given that a solution is plausible, the gain value allows us to quantify its utility for increasing the classification performance. Therefore, this measure is used for comparing solutions from different cluster levels, after discarding the ones that are not plausible enough. We simply choose the solution which has the best gain. This way we automatically select the best number of clusters. Furthermore, the gain measure may also help to quantify the utility of different sensors or features in case when multiple are available, which would help selecting the ones that can best contribute to the problem.

The parameter B denotes the minimal number of data instances allowed in a leaf of the cluster-based decision tree that is built by our method in order to find labels for clusters (see Section 5.5.2). Decreasing this parameter means to allow a more fine-grained search for single-cluster regions which are needed to reveal labels of clusters that are overlapped by others. While decreasing B often leads to more single-cluster regions it also rises the sensitivity to noise.

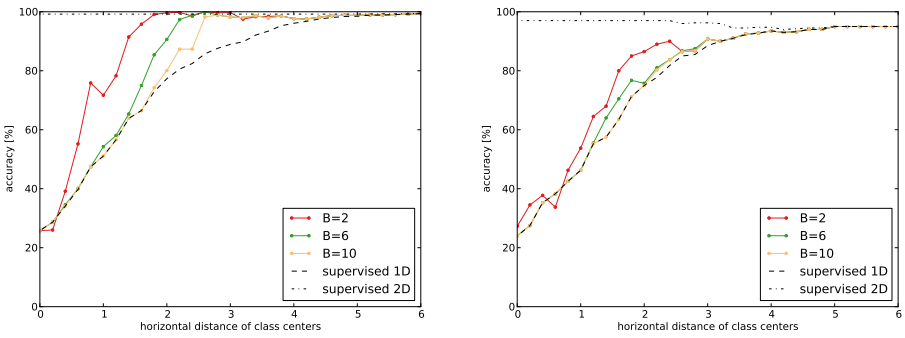
5.6 Evaluation on Synthetic Data

5.6.1 Gradually Decreasing Cluster Overlap

First, we investigate the technique with the first synthetic data set (gradually increasing cluster overlap). Figure 5.7 shows the average accuracy of the adapted classifier resulting from our unsupervised method in relation to initial feature space cluster distance and compared to fully supervised trained

classifiers. Results are averaged over five runs with different random seeds for increased robustness against particular effects of a single data set.

In this simulation, see Figure 5.7a, the result with the lowest value for B clearly outperforms others. It is up to 29.3% better than the initial classifier (supervised 1D) and often meets or even exceeds the performance of the fully supervised classifier (supervised 2D). Only at distance 0.2 does it make a very bad initial classifier slightly worse. The results for $B = 6$ and $B = 10$ behave more conservatively. They never fall below the accuracy of the initial classifier here but they also show fewer occasions with true performance increase.



(a) 50 instances per class

(b) 10 instances per class

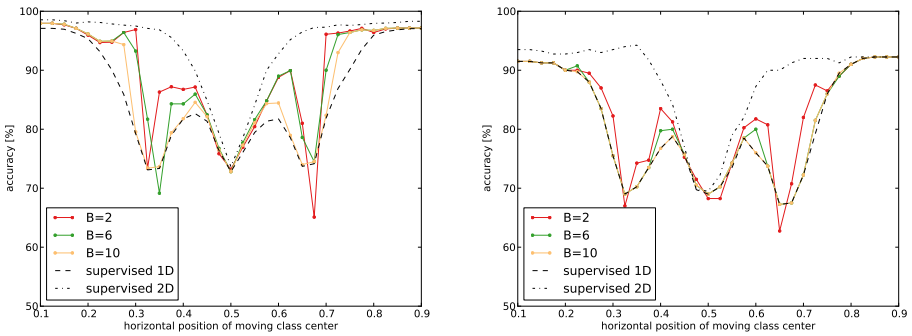
Figure 5.7: Gradually Decreasing Cluster Overlap: Effect of cluster overlap in first dimension shown for increasing horizontal distance between cluster centers (see Fig. 5.5) and for two different amounts of unlabeled 2D space instances (adaptation set). The accuracy of the adapted classifier is plotted for three different parametrizations of our unsupervised method. For comparison the results of the initial classifier (supervised 1D) and of a fully supervised classifier (supervised 2D), which is trained with an additional dimension for the ground truth instances, are given.

The results of this simulation indeed meet our expectations. We do not impair a good classifier ($\mathsf{dist} > 4$) and for initial classifiers with accuracies above 74% ($\mathsf{dist} > 1.6$) we can achieve perfect or nearly perfect improvement resulting at 97.5–100%. For initial classifiers below 74% accuracy we can still achieve a mean improvement of over 21%. Only at the lowest end there is one occasion where performance is decreased by 2.6%. These results also confirm the usefulness of the plausibility and gain measures to decide on the most promising cluster configuration.

The example results displayed in Figure 5.5 show the detailed regions of the initial classifier (top) and the adapted classifier (bottom). While in Figure 5.5a the four classes completely mix up in the histogram of the initial feature space and no true classification or improvement is possible at all, in Figure 5.5b there is already a slight difference between class centers in the initial dimension which the initial classifier is exploiting. Our method ($B = 2$) is able to find single-cluster regions (and thus labels) for all clusters, but in this example obviously fails to choose the correct number of clusters (the dark blue cluster is split) and

is in the mistaken belief that it found correct labels for all clusters as it splits up the leftmost region of the classifier. However, the classification accuracy in this region of the initial classifier is so low that even a wrong modification only does little harm to the accuracy because most instances are classified wrong anyway. Figure 5.5c shows an example of nearly perfect improvement (77% to 99.5%) and in Figure 5.5d the initial classifier is perfect enough that we cannot improve it in the second dimension alone.

Figure 5.7b shows results for the same simulation but with a much smaller data set. The set of 2D feature space data that is used for the unsupervised adaptation only contains 10 instances per class. These harder conditions tribute in a much lower gain of accuracy compared to Fig. 5.7a. Even the fully supervised variant, which is provided with the second dimension for the ground truth data, shows less accuracy on the (larger) test set. Still, our method does not harm a reasonable good classifier. Above 85% initial accuracy we hardly achieve any improvement but from there down to 42% initial accuracy ($\mathsf{dist} = 0.8$) we still realize a mean accuracy gain of 10.2%. With even more cluster overlap ($\mathsf{dist} < 0.8$) we eventually harm the initial classifier or can only achieve small improvements.



(a) 50 instances per class

(b) 10 instances per class

Figure 5.8: Moving a single cluster: Results for a similar situation as in Fig. 5.7 but with one single cluster that changes horizontal position and eventually is completely overlapping an other cluster as shown in Fig. 5.6c.

5.6.2 Moving a Single Cluster

We now evaluate the technique on the second synthetic data set (moving a single cluster). The accuracy of the base classifier in Figure 5.8a (supervised 1D) clearly reveals the positions of the fixed clusters where it reaches local minima at 0.33, 0.5, and 0.66. This is where the moving cluster completely overlaps with one of the fixed clusters in the initial feature space. At the center position those clusters also totally overlap in the 2D feature space which makes it even impossible for the supervised 2D variant to separate them. Notably, our solution does "no harm" here. At the outer two positions, however, our method eventually drops below the initial accuracy. This is once the case each for $B = 2$

and $B = 6$. With $B = 10$ our method does "no harm" at all but also shows only little accuracy overall. Between and aside of these three positions our method can achieve very good to nearly perfect improvement as it comes close to the results of the fully supervised variant (supervised 2D). Such an example is depicted in Fig. 5.6b.

In the case of only 10 instances per class (Fig. 5.8b) only the variant with $B = 2$ drops below the accuracy of the initial classifier at the three critical positions described above. Overall, the accuracy increase is much less than with 100 instances per class but there is still a fair increase of 2–10% between and besides these three positions. With the more conservative variants $B = 6$ and $B = 10$ we do not see any performance degradation, but also only very little improvement, if any.

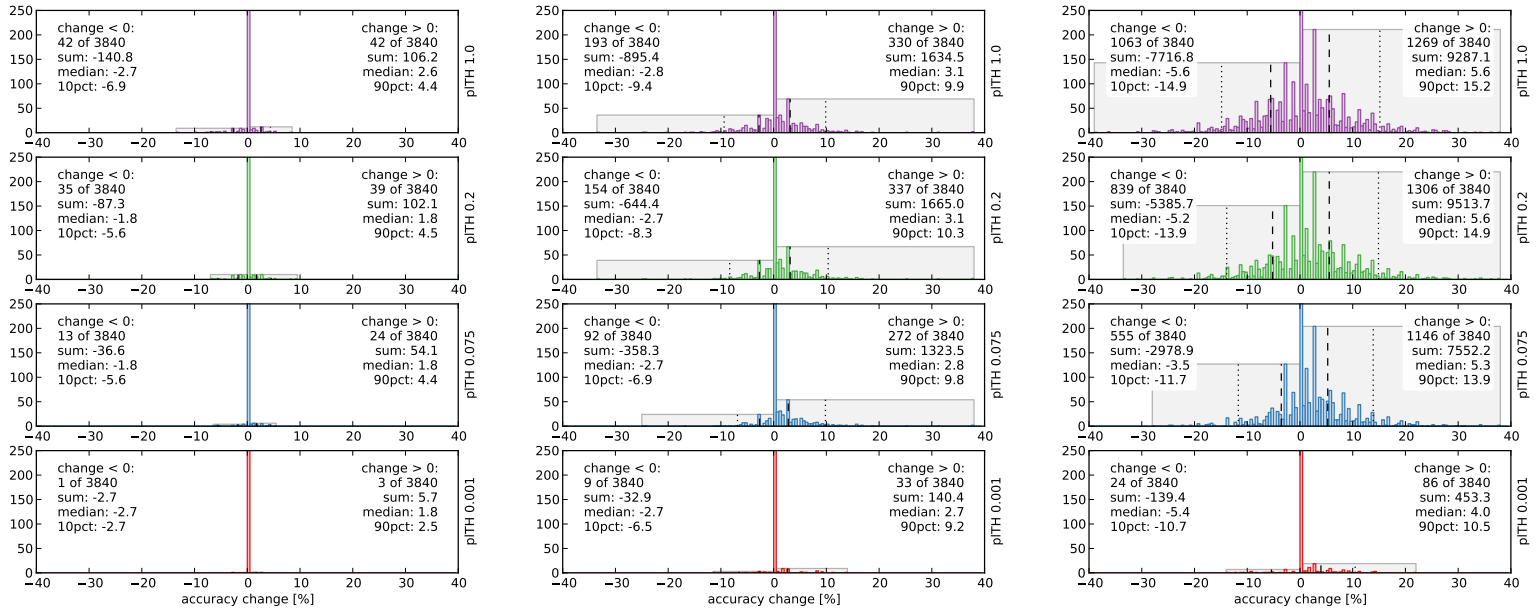
5.7 Evaluation on Real Sensor Data

The results on all three real data sets are summarized in Figure 5.9. The histograms display the number of sensor combinations per accuracy change. This is the unsupervised change of accuracy achieved by our method.

The histograms clearly reveal the number of combinations with desirable (positive accuracy change) and undesirable (negative accuracy change) results. Regarding Figure 5.9c, lowering the purity threshold (plTH) from 1.0 (unrestricted, top) down to 0.001 (strongly restricted, bottom) is clearly reducing the number and grade of undesirable results while keeping nearly all desirable results. Only at the lowest threshold (0.001) the number of desirable results is strongly reduced too. This effect is also visible in the results with less fine grained cluster based trees (Figures 5.9a and 5.9b), but in accordance with the results on synthetic data discussed above, rising the parameter B, i.e., increasing the minimal allowed leaf size for the cluster-based tree used for finding cluster labels, reduces not only undesirable results but also many of the desirable ones.

Judging from the histograms, the "blue" result in Figure 5.9c (plTH 0.075, B=2) seems the most reasonable. While it features many high grade desirable results (1146 of 3840, 29.8%) with a median of 5.3%, it only has 555 (14.5%) undesirable ones with a smaller median of -3.5% accuracy change. Figure 5.10 summarizes the results by opposing the sum of positive change to the sum of negative change. With B=2 the method clearly outperforms other parametrizations in terms of sum of positive change but with a similar increase in sum of negative change. With B=5 the method behaves much more conservatively, avoiding large negative change and still achieving some positive change. Notably, the sum of positive change always dominates, meaning that our method actually achieves an improvement over all tested sensor combinations by integrating the new sensor without any user input.

The effects of both parameters (plTH, B) are almost identical when compared between separate results from all three data sets, suggesting that they will occur on other data sets correspondingly without need for adapting the parameters.



(a) label inferring, $B = 10$ (b) label inferring, $B = 5$ (c) label inferring, $B = 2$

Figure 5.9: Results for our mthod with different parameterizations on the three real-world data sets (13 subjects, $768 \cdot 5$ tested sensor combinations in total). Each plot shows results for four plausibility thresholds (plTH). Bounding boxes for positive and negative results are highlighted, including 10/50/90 percentiles.

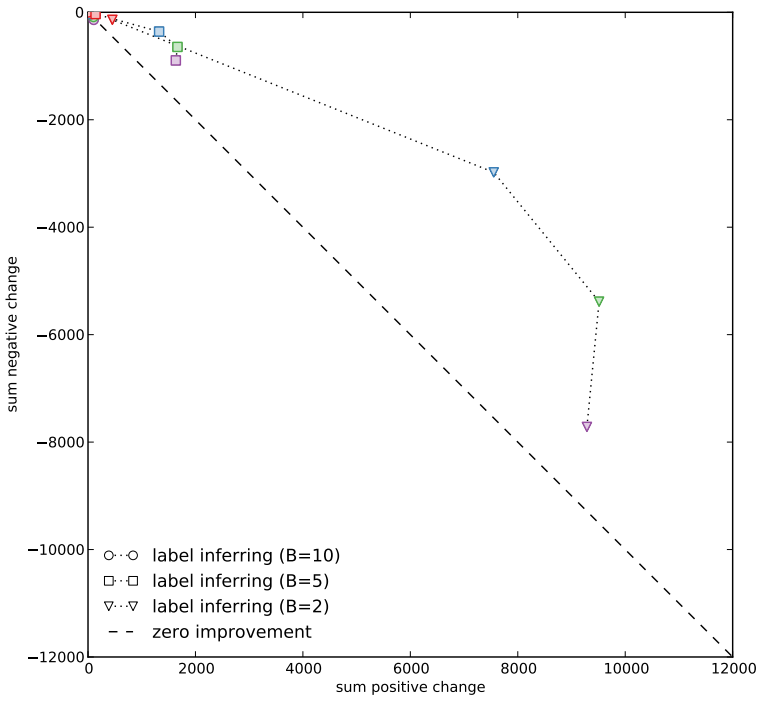


Figure 5.10: Comparison label inferring results with different parameters. Colors correspond to plausibility thresholds (plTH) as in Figure 5.9.

5.8 Conclusion

In this chapter we presented an approach to enable an activity recognition system based on a decision tree classifier to integrate a new source of information in a completely unsupervised way. We evaluated the system on two synthetic data sets and on three previously published real world data sets, one concerning bicycle repair, one about car maintenance, and one dealing with a home setting scenario. The method can achieve perfect results on synthetic data sets, which are comparable to systems trained in a fully supervised manner, even in difficult situations where multiple clusters overlap each other in the initial feature space. With the restrictions of real world data sets with limited sample size our label inferring method can still achieve more improvement than it does "harm" with 29.8% of the tested sensor combinations the accuracy being increased by a median of 5.3% and for the best 10% thereof realizing an accuracy increase of more than 13.9%.

One of the requirements stated in the introduction, however, is not met satisfactorily as we required the system to decrease performance in not more than a few occasions. When we choose parameters to achieve a reasonable amount of

improvement without getting large degradations, we still have nearly 14.5% of cases where the performance of the initial classifier is slightly decreased with a median of -3.5%. The reason for this can be sought at several stages. First, the variation between data samples seen in real world data sets is in conflict with our assumption made on equal class distributions in training-, adaptation-, and test set. Second, our label inferring method relies on the ground truth labels of a small subset of data samples (single-cluster regions) for inferring the class distributions to entire clusters. The parameter B directly influences the minimal size of those regions. If the single cluster regions are not representative for the entire clusters, then our method is promoting erroneous configurations which not just diminish or prevent an improvement of the classifier accuracy, but actually can cause it to perform worse than with the single sensor alone. The plausibility and gain measures used for assessing the quality of solution candidates are estimates that may not detect and sort out all such erroneous configurations.

A second observation from evaluation on real world data sets is, that many clusters are hidden, such that they do not appear alone in a region when projected onto the initial dimension, i.e., they are not part of any single-cluster region. And thus, there is no way for our label inferring method to assign labels to those clusters and taking advantage of them for improving the classifier. Furthermore, those hidden, unlabeled clusters actually prevent improvement in all the regions of the initial feature space they cover.

We address both of the above issues in Chapter 6 where we (1) investigate an advanced method, based on similarity search, to find labels even for hidden clusters, and (2) utilize bagging to reduce vulnerability to sample noise. In Chapter 7 we also consider an approach to use user provided labels for sorting out and discarding erroneous configurations.

Chapter 6

Labeling Clusters Based On Distribution Similarity

The unsupervised method to integrate a new sensor introduced in the previous chapter achieved promising results. Yet, the complex class distributions found in real world activity recognition data sets revealed limited opportunities for our method to achieve any improvement. Derived from the observations made in the previous chapter we present here an advanced approach for the unsupervised assignment of labels to clusters in the extended feature space. The approach is focused on finding the label assignment that – when projected back to the initial feature space – best resembles the label distribution found in the training data of the initial system. Additionally, in order to reduce the influence of sample noise to our method we integrate a bagging technique. The evaluation on the same synthetic and real world data sets as used in the previous chapter reveal a clear advantage in overall yield, i.e., the number of tested sensor pairs for which an improvement can be achieved. This however comes at cost of increased failure rate. With bagging we can efficiently reduce this failure rate with only moderately affecting the yield.

David Bannach, Bernhard Sick, and Paul Lukowicz. Automatic adaptation of mobile activity recognition systems to new sensors. In *Workshop Mobile sensing challenges, opportunities and future directions at Ubicomp 2011.*

The promising results of the unsupervised method for automatically integrating a new sensor into an existing activity recognition system, introduced in the previous chapter, encourage to further investigate its limits. While on synthetic data sets that method demonstrates its ability to compete in many cases with systems trained in a fully supervised manner, it also shows difficulties when, in real world data sets, the sample size is reduced or when clusters totally overlap each other in the initial feature space such that they are "hidden" to the method. Here, we explore the sources of the limitations and address the issues with an advanced approach.

6.1 Limitations of the Label Inferring Method

A key component of the label inferring method introduced in the previous chapter is that it attempts to infer labels for all clusters of unlabeled data in the extended feature space. The ground truth data in the initial feature space and the structure of the extended feature space clusters are the sources of information from which the labels are deduced. In fact, the clusters are projected onto the initial feature space where the method searches for regions where only data points of a single cluster fall into and based on the ground truth data of that region it infers labels to the whole involved cluster.

One obvious parameter which is influencing the performance of the method is the minimal size of single-cluster regions to look for (B, see Section 5.5.4). With large regions there is a good chance to capture a representative set of ground truth instances for the cluster, yet such large single-cluster regions are hard to find in real world data sets. This can be seen in Figure 5.9 where for plTH 1.0 and B=10 just 84 of 3840 sensor combinations resulted in any change (accuracy increase or decrease), while for B=5 and B=10 those numbers increased to 523 and 2332 respectively. Thus, smaller regions occur much more often, yet they rise the sensitivity to noise, both in the clustered data set and the ground truth data. The need for sufficient single-cluster regions is intensified by the fact, that for adapting the classifier in a region where multiple clusters fall into, all of the involved clusters need to be labeled. Otherwise, it would not be possible to assess the plausibility of the labeling. Thus, one unlabeled cluster prevents improvement in all regions it covers.

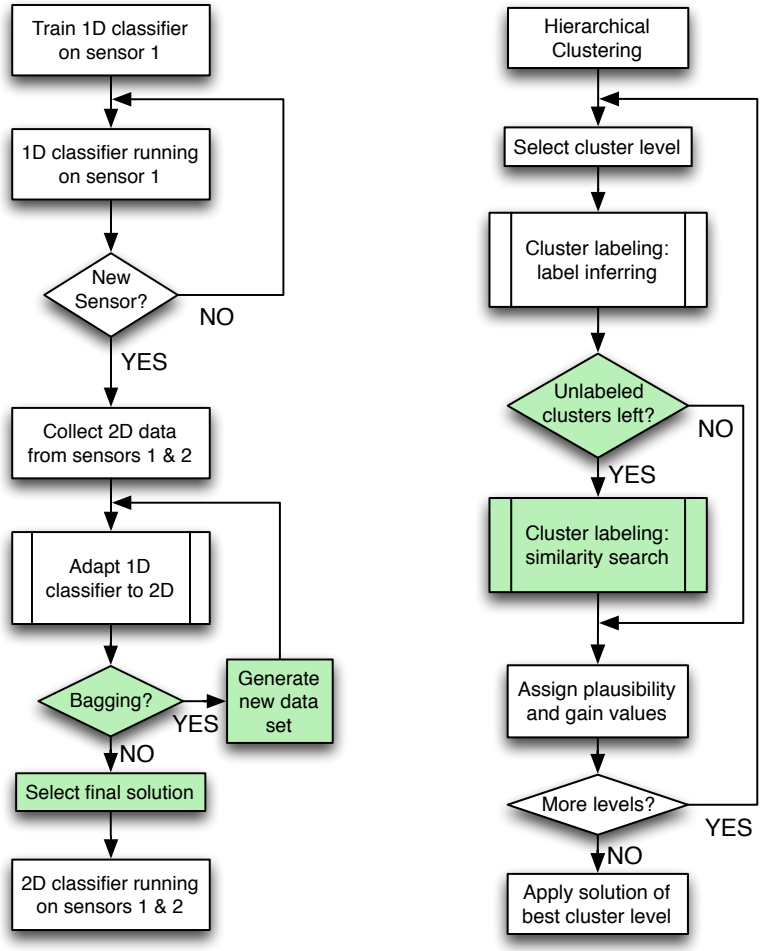
In Section 5.3.2 and Figure 5.3 we have discussed the problem of "hidden" clusters. These are clusters that have no points that are projected onto a single cluster region of the original feature space and, thus, can not be assigned a class distribution using the method described in the previous section. The effect of such hidden clusters can be seen in the performance graph in Figure 5.7a. Our label inferring method needs the cluster centers to be at least dist=1.6 apart to closely approach the accuracy of supervised 2D training and achieve a nearly perfect improvement. Below a value of dist=1 there is nearly no improvement compared to what supervised 2D achieves over the original classifier.

The noise that occurs in real data sets with small sample size also affects the gain estimate. The estimated gain of accuracy is the only way for our method to decide on the best number of clusters, optimizing the actual improvement. The gain estimate, based on the estimated overall purity of the modified classifier, is assuming identical class distributions in the clustered data set (when projected to the initial feature space) and the training set. Thus, when noise is affecting the class distributions the gain estimate gets inaccurate and, as a result, the wrong cluster level might be selected by our method.

6.2 Chapter Outline

With respect to the promising results on synthetic data sets we are encouraged to advance beyond the shortcomings of the label inferring method. Since one drawback of inferring labels via single-cluster regions is that it can not always find labels for all of the clusters, we like to answer the the question: Among all possible labelings of the given clusters, can we accurately find the correct one?

In this chapter we present an extension of the system introduced in the previous chapter. The extension attempts to assign class labels to clusters of unlabeled data, based on the similarity of the resulting class distribution to the class distribution found in the ground truth data. Furthermore, we integrate the bagging method as a way to avoid overfitting and to cope with noise in data sets in order to foster reliability of the gain estimate and the labeling results. We evaluate the advanced method on the same synthetic and real-world data sets as the label inferring method and compare the results.



(a) Overview (b) Adapt 1D to 2D

Figure 6.1: Overview of the overall procedure (a) and its most important step, the classifier adaptation (b). The new steps compared to Figure 5.4 are highlighted

6.3 The Similarity Search Method

The similarity search method integrates seamlessly into the system presented in the previous chapter. The main step of this method – the actual labeling of "hidden" clusters – is inserted directly after the label inferring step, as highlighted in Figure 6.1b.

The basic idea of the similarity search approach is to find class labels for the clusters, such that, when applied to the cluster data points and projected back onto the initial feature space, the resulting class distribution for each region of the original classifier most closely matches the original class distribution of that region (from the labeled data set). This means, among all possible combinations of cluster label assignments, we search for the labeling which maximizes plausibility as defined in Section 5.5.4. For this approach to work, we assume a single label per cluster instead of a label distribution as we used to do for the label inferring method. This is not a limitation for the final classifier as its leaves, which include the ones generated from cluster information, will each decide for a single class only. Clusters that have already been labeled in earlier steps are not touched and count with their most prominent label.

The procedure is as follows:

1. For each possible combination of labels assigned to clusters:
   (a) For each region of the original classifier:
      i. Retrieve the class distribution the region had in the original training data.
      ii. Compute the class distribution that results from the projection of the data from all the clusters (with their current label) onto the region.
      iii. Compute the plausibility value pl for the two above distributions as defined in Section 5.5.4.
   (b) Sum up the plausibility value retained from each region, weighted by the number of cluster data points in the respective region.
2. Return the label combination which yields the lowest sum (= the most plausible one).

Figure 6.2 illustrates the similarity search approach with an example setup. Three of the 2D clusters have been labeled with the blue, green, and red class already (by the label inferring method, for instance) and the fourth, gray cluster needs to be assigned a label still. From the 4 possible labels, the purple one obviously is the most plausible, as the projection to the sensor 1 feature space then best resembles the original class distribution for each region of the sensor 1 classifier. Note, that even if none of the clusters were labeled beforehand, our method would reveal the correct labels for all clusters in this example by evaluating the plausibility of all 256 possible label combinations on each region.

The initial classifier offers a natural way to break down the feature space into separate regions according to the structure of the data such that entropy is reduced within the regions. This is clearly the case for the C4.5 decision tree classifier but is valid for classifiers in general. In our method we utilize this natural partition of the feature space and evaluate the resulting class distributions

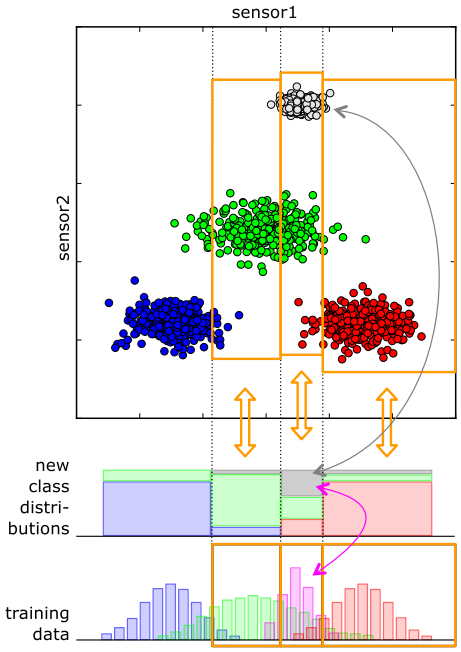


Figure 6.2: Illustration of the similarity search method for assigning labels to 2D clusters. Dashed lines show boundaries of the initial decision tree classifier.

individually for each region of the original classifier. The comparison of class distributions across multiple regions allows us to compensate for ambiguous cases where different labelings result in the same best local plausibility.

Such ambiguous cases can occur in two situations:

1. When in a class distribution of the labeled data set two labels l_1 and l_2 with identical contribution exist (i.e., same probability), then for each labeling of the clusters there exists an equally plausible labeling with the labels l_1 and l_2 permuted. Thus, there are two labelings with maximal plausibility.

2. When in the clustered data two disjoint groups of clusters with identical size (number of data points) exist, then for each labeling that consistently assigns label l_1 to all clusters in one group and label l_2 to all clusters of the second group, there exists an equally plausible labeling with the labels l_1 and l_2 permuted. If this is the case for the most plausible labeling the solution is ambiguous.

Even if the probability that an ambiguous situations occurs is relatively low, they still may occur with real-life data. However, if just one of the involved clusters appears in other regions too, which is often the case, then in sum the plausibility of all regions can still reveal a single best labeling. It is very unlikely that a cluster shows the same ambiguity in all regions it covers.

Thus, it seems feasible for our method to identify a single most plausible labeling among all candidates. Yet, a complete search over all possible labelings

leads to a combinatoric explosion: with N classes and K clusters there are N^K candidates to verify for each region. In order to reduce the number of candidates we apply two different stages of optimization.

Dependency Graph

First, we build the dependency graph with a node for each cluster and edges connecting each pair of clusters that appear together in the same region. The connected components of the dependency graph form disjoint groups of clusters. The sets of regions in which the groups of clusters appear are disjoint too. This means that the labeling of the clusters in one group does not affect the labeling of any other cluster and we may search the best labeling for each group of clusters individually. Thus, there are maximally $N^{\max(|g_i|)}$ candidates to verify, where $|g_i|$ denotes the number of clusters in the i-th group.

The maximal size of connected components $\max(|g_i|)$ is expected to be notably smaller than N in activity recognition scenarios. It is related to the maximal number of classes which the initial classifier confuses.

Avoid Evaluating "Impossible" Labelings

We can reduce the number of labeling candidates individually per group when limiting the search to the set of labels that actually appear in the training data of the regions affected by the group. Those discarded candidates would not be more plausible than any of the remaining candidates. This individually reduces N for each group.

6.4 The Bagging Method

With the similarity search method we now entirely rely on the plausibility estimate to determine the cluster labels, without any parameter to control the amount of "risk" we take. To prevent our method from being misled by noise and sample variance we include a bagging [Bre96] step: before starting, from the given unlabeled data set we form 10 replica of the same size, by drawing each instance at random, but with replacement. Each instance may appear repeated times or not at all in any of the replica. As highlighted in Figure 6.1a, we run our method individually on each replicated data set and aggregate the resulting classifiers. Where at least *bagging threshold* percent of the classifiers agree, the majority result is taken. Otherwise we fall back to the result from the initial classifier without any improvement.

When for a small change in the data set our method produces a different solution, i.e., because of a cluster getting labeled differently or the clustering deviates, this is a strong indication of either an ambiguous situation or overfitting (high model complexity). Both are cases that we want to avoid. Thus, filtering out those solutions is desirable, and it is exactly what can be achieved with bagging.

6.4.1 Merging Aggregated Classifiers

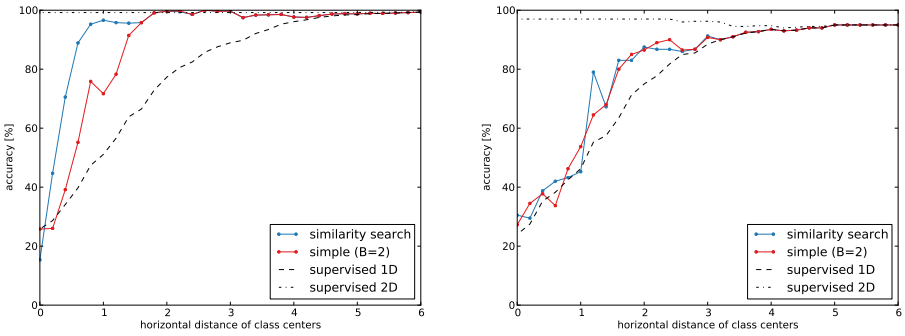
In case of decision tree classifiers, on which we base our experiments, we can merge the aggregated classifiers resulting from the bagging step into a single

decision tree. This simplifies classification afterwards, as only a single decision has to be traversed.

Our method of integrating a new sensor splits up the regions of the original classifier, which are the leaves in the decision tree, along the new dimension of the feature space. Thus, each leaf may get replaced by a subtree, providing new leaves for each split region. When merging the aggregated classifiers we first overlay the new subtrees in each region and successively (1) assign the majority label to all split regions (leaves of subtree) where at least *bagging threshold* percent of the subtrees agree, and (2) assign the initial label of that region (from the original classifier) to all other split regions. Adjacent split regions with identical label are joined.

6.5 Evaluation on Synthetic Data

We begin by evaluating our extended method on the synthetic data sets introduced in Section 5.4.1. Consequently, we can investigate the same situations that resemble real world conditions and compare the results to the label inferring method presented in the previous chapter.



(a) 50 instances per class

(b) 10 instances per class

Figure 6.3: Gradually Decreasing Cluster Overlap: Effect of cluster overlap in first dimension shown for increasing horizontal distance between cluster centers (see Fig. 5.5) and for two different amounts of unlabeled 2D space instances. The average accuracy is displayed for the similarity search variant of our unsupervised method presented in this chapter and for the simpler variant presented in Chapter 5 (see Fig. 5.7). For comparison the results of the initial classifier (supervised 1D) and of a fully supervised classifier (supervised 2D), which is trained with an additional dimension for the ground truth instances, are given.

Gradually Decreasing Cluster Overlap

Figure 6.3a clearly shows the advantage of our similarity search method over the simpler label inferring method of Chapter 5 (first synthetic data set). While the

simple method needs at least dist=1.6 to closely approach the accuracy of supervised 2D and achieve a nearly perfect improvement, the similarity search method achieves this already at dist=0.8. Thus, with less than one time the standard deviation of horizontal distance between centers of the Gaussian-distributed clusters the similarity search method can already achieve a nearly perfect result without any user input. Furthermore, there is still a reasonable amount of improvement over the initial classifier down to dist=0.2. Only with complete overlap of the clusters in the initial feature space (dist=0) the similarity search method does "some harm" and falls below the initial accuracy.

With the more ambitious restriction of only 10 instances per class (see Fig. 6.3b), the results are much more noisy and the similarity search method achieves similar results as the simple method with the difference that it does even less "harm" and in one case still reaches double the improvement of the simple method.

Moving a Single Cluster

With the second synthetic data set we observe a better overall performance of the similarity search method than the simple method achieves, as displayed in Figure 6.4a. As with the simple method there is also just one single occasion where accuracy is decreased at a minima of the 1D classifier's results.

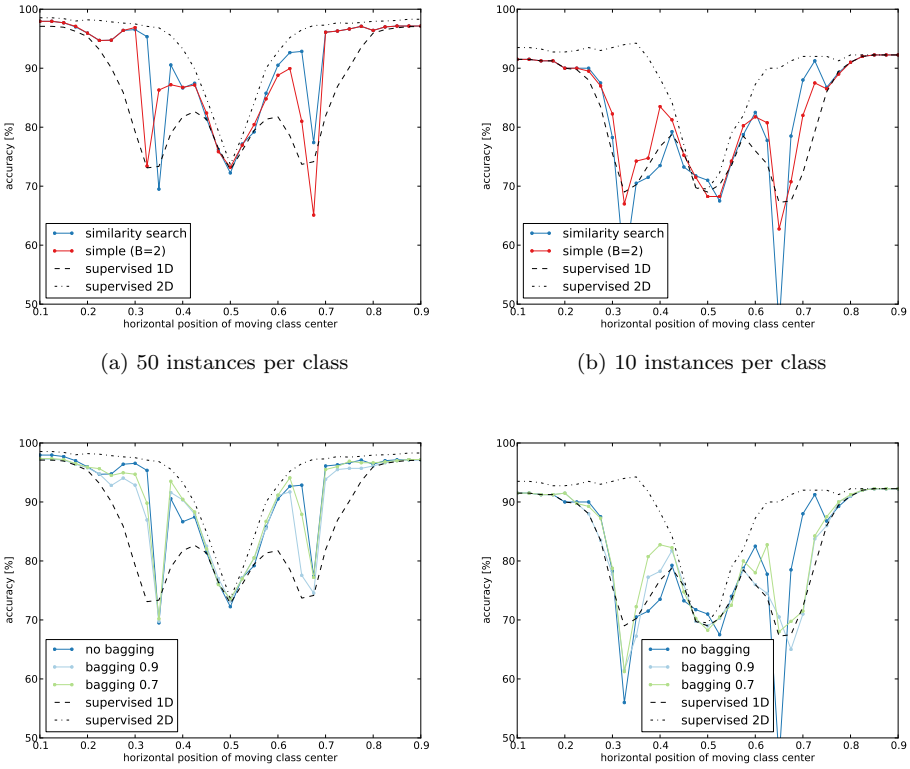
The price that has to be paid for the increased improvement can be best seen in Fig 6.4b when considering only 10 instances per class that can be used for labeling. In this data set there is a critical region (around shift of 0.3 and 0.65) when the shifted cluster is merged with some of the other clusters and the basic assumptions described in Section 5.3.1 are violated. In addition, because of a small sample size of 10 instances per class there is much more sample variance and the information on which the decisions of the similarity search are based is much less representative. As a result, in the critical area the similarity search method leads to a significant performance decrease (nearly -30%) over the 1D classifier.

With a bagging threshold of 0.7 (see Figure 6.4d), the performance decrease in the critical area is reduced to just under -10%. This is achieved at the price of slightly reducing the overall improvement in other areas as the system is "more conservative" about using information from the extended data set. In general, the results are more stable and better reflect the symmetry of the data sets when bagging is enabled (see Figure 6.4c).

6.5.1 Evaluation on Real Sensor Data

For the evaluation on real world data sets introduced in Section 5.4.2 we proceed the same way as described in Chapter 5 and compare the results.

We evaluate three different strategies in this chapter. First, we investigate the performance of both methods in sequence as depicted in Figure 6.1b, first the label inferring method from the previous chapter followed by the similarity search method described above. This way, the second method only cares about clusters which have not been labeled by the first method already. For the second strategy we completely disable the label inferring stage and solely rely on the similarity search method to find labels for all clusters. Lastly, we investigate



(a) 50 instances per class

(b) 10 instances per class

(c) 50 instances per class, bagging

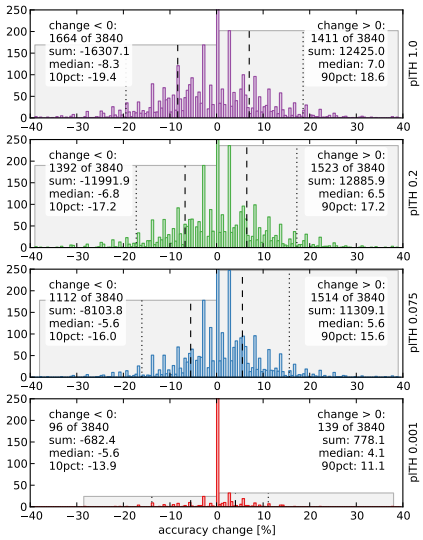
(d) 10 instances per class, bagging

Figure 6.4: Moving a single cluster: Averaged results for the same situation as shown in Figure 5.8 where one cluster is horizontally displaced to cross three fixed clusters and eventually is completely overlapping one of them in 2D space as shown in Figure 5.6c. Additionally, the results of bagging applied to the similarity search method are provided with two different thresholds for merging the aggregated decision trees.

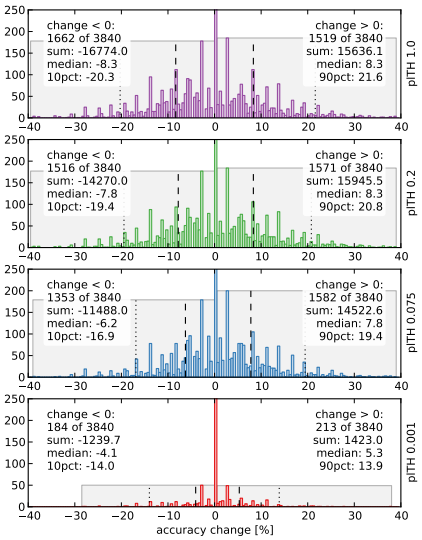
the the influence of bagging (see Figure 6.1a) in conjunction with the first two strategies.

Results for the first two strategies are displayed in Figure 6.5. Compared to results of the label inferring method alone (Figure 5.9c), the combination with the similarity search method reveals more occasions where an improvement is achieved, e.g., for B=2 and plTH 0.075 there are 1514 of 3840 tested sensor pairs (39.4%) for which an improvement is achieved as opposed to 1146 (29.8%) with the base method only. On the other hand, the addition of the similarity search method also increases the number of occasions where the classifier performance is mistakenly decreased by our unsupervised method, e.g., 1112 of 3840 sensor pairs (29.0%) with negative change instead of 555 (14.5%) for the base method alone (same parameters as above).

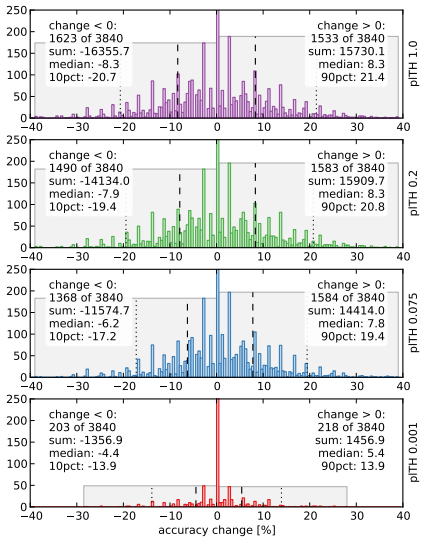
The results in Figure 6.5 also clearly show that the similarity search method



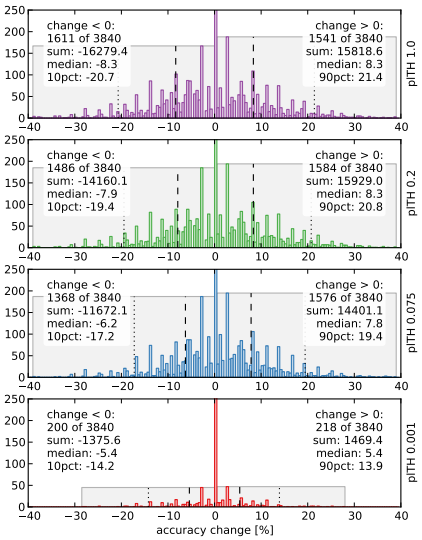
(a) label inferring (B=2) + similarity search



(b) label inferring (B=5) + similarity search



(c) label inferring (B=10) + similarity search



(d) similarity search (only)

Figure 6.5: Results for our similarity search method with different parameterizations on the three real-world data sets (13 subjects, $768 \cdot 5$ tested sensor combinations in total). Each plot shows results for four plausibility thresholds (plTH). Bounding boxes for positive and negative results are highlighted, including 10/50/90 percentiles.

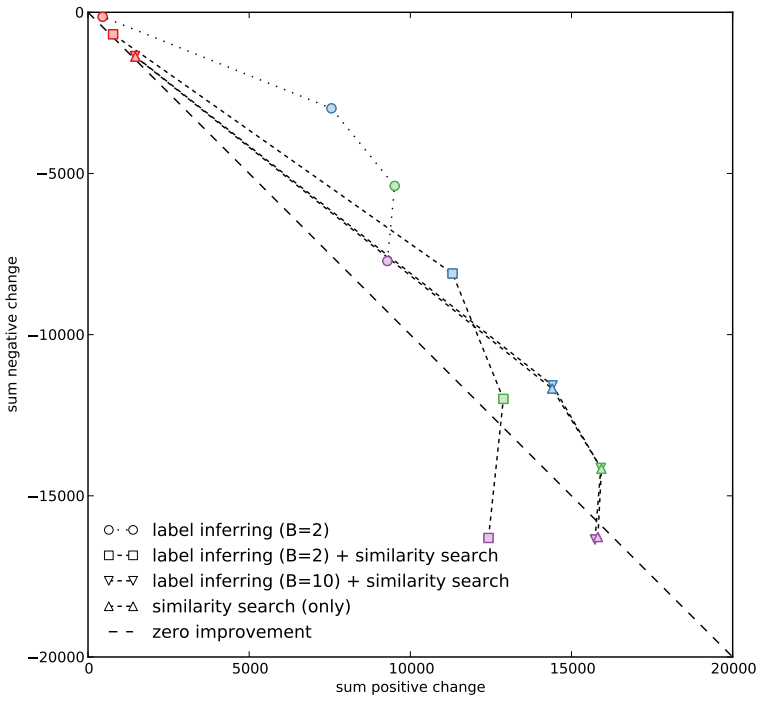
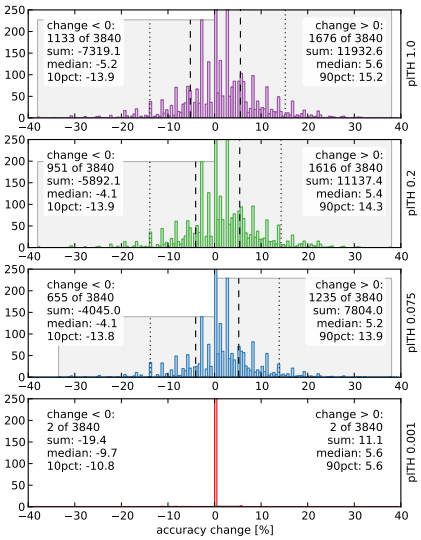


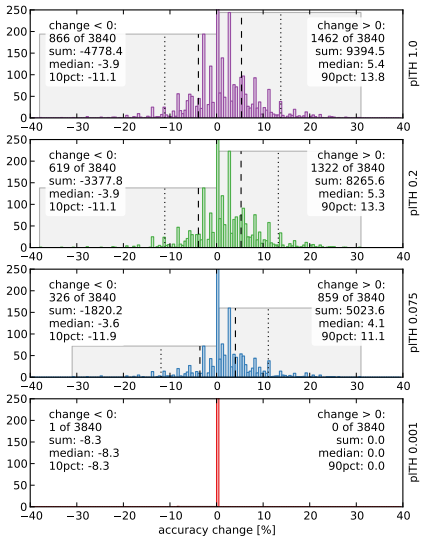
Figure 6.6: Comparison of results from the label inferring and similarity search methods. Colors correspond to plausibility thresholds (plTH) as in Figure 6.5.

is able to compensate for situations where the label inferring method is not able to find a solution for integrating the new sensor: while with increasing B the label inferring method alone is achieving nearly no improvement (see Figures 5.9b and 5.9a), in combination with the similarity search method the amount of improvement remains stable (see Figures 6.5b and 6.5c). Even with the similarity search method alone (Figure 6.5d) the performance remains in a similar range.

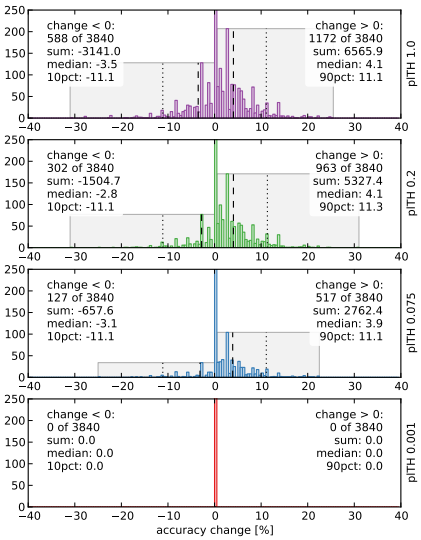
Figure 6.6 summarizes the above results regarding the total amount of positive change vs. the sum of negative change. With the integration of the similarity search method we can almost double the amount of positive change compared to the results of the label inferring method alone, but at the cost of 3-4 times more overall negative change. The best result of the similarity search method alone (plTH 0.075) yields a ratio of just above 1.2:1 for overall positive change vs. negative change while the more conservative label inferring method already had a ratio of more than 2.5:1 (same plTH). Those results clearly indicate that the plausibility estimate, which is considered for detecting and discarding erroneous cluster labelings with the plausibility threshold, is less effective with the similarity search method: for the label inferring method a plausibility threshold



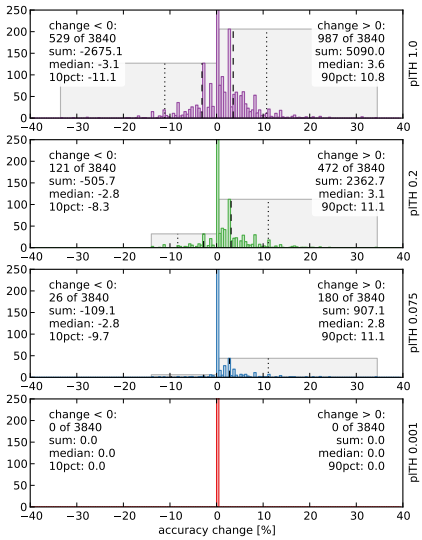
(a) similarity search + bagging 0.7



(b) similarity search + bagging 0.8



(c) similarity search + bagging 0.9



(d) label inf. (B=2) + s.s. + bagging 0.9

Figure 6.7: Results for our similarity search with bagging method with different parameterizations on the three real-world data sets (13 subjects, $768 \cdot 5$ tested sensor combinations in total). Each plot shows results for four plausibility thresholds (plTH). Bounding boxes for positive and negative results are highlighted, including 10/50/90 percentiles.

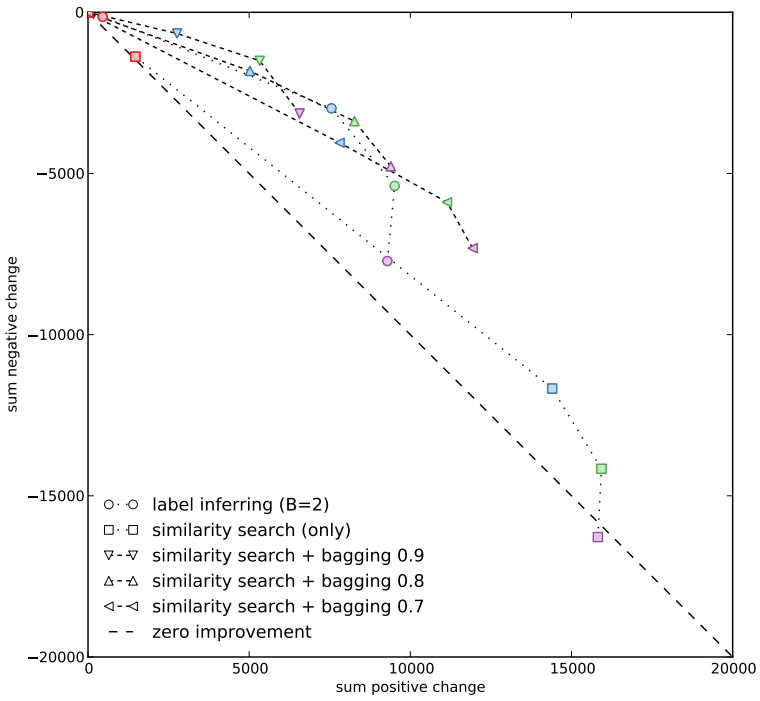


Figure 6.8: Comparison of similarity search results with bagging. Colors correspond to plausibility thresholds (plTH) as in Figure 6.7.

of 0.075 (blue) discards more than 60% of negative change (compared to the unrestricted, purple result), as opposed to just 28% for the similarity search method.

The lack of efficiency of the plausibility threshold is compensated by the bagging method as can be seen in the results of the third evaluation strategy with bagging enabled in Figure 6.7. With increasing bagging threshold (a bagging threshold of 0.7 means that 70% of the aggregated trees must agree for adapting a region) the number of outcomes with negative accuracy change are strongly reduced while still keeping most of the results with positive change. This effect is summarized in Figure 6.8 regarding the total amount of positive change vs. the sum of negative change. The bagging method brings the results of the similarity search method back into the range of the label inferring method, but with the bagging threshold as a powerful means for controlling the amount of risk to take. The similarity search method in conjunction with a bagging threshold of 0.9 achieves a positive- vs. negative change ratio of over 3.5:1 (plTH 0.2) and a threshold of 0.7 yields more positive change than the label inferring method (11137 vs. 9514) at nearly the same level of negative change. (5892 vs. 5386)

6.5.2 Discussion of Results

The motivation behind the similarity search approach for cluster labeling is to avoid the limitations found when evaluating the label inferring approach in Chapter 5, specifically to avoid situations where not all clusters could be labeled. We actually reach this goal on synthetic data sets (see Fig. 6.3a) where the similarity search method clearly outperforms the label inferring method when clusters are heavily overlapping in the initial feature space.

However, when it comes to more ambitious situations with only 10 instances per class the advantage of the new approach vanishes. The noise introduced by the small sample size hinders accurate estimation of plausibility and gain which can result in wrong cluster labelings being preferred. We can successfully reduce the negative effect of noisy data sets with bagging (see Fig. 6.4d). Yet, when discarding a vague solution (i.e., a new sub-region with different class label in a region of the initial classifier) we also loose the chance to improve accuracy of that class in that region. This is where we reach a similar limitation as seen with the label inferring method in Chapter 5. Compared to the label inferring method the limitation is slightly less grave since for the similarity search method it is still possible to achieve improvement for another class in that region, if the bagging results sufficiently agree, whereas the label inferring method cannot improve at all in that region when just one of the clusters present is not labeled.

Figure 6.9 depicts such a difficult situation from the OPPORTUNITY data set. The initial classifier is trained on one axis of the accelerometer mounted on the subject's back (BAC_accY). From the histogram and the regions of the classifier (top left plot) we see that the initial system reaches an accuracy of 50% on 8 classes. The numbers indicate the (absolute) class distribution in each region. The new sensor in this example is one axis of an accelerometer mounted on the right lower arm (RLA_accZ). The unlabeled data recorded with both sensors is shown in the top right plot.

Regarding the broad blue region in the center (third from right) there are only slightly more training instances for class A (17 instances, 52%) than for class C (15 instances, 45%). From the test data (lower left plot) we, as the supervisors, can see that the upper cluster is actually belonging to class A and the instances of class C are in the lower half. Unfortunately, the unlabeled data (upper right plot), which is what our unsupervised method sees, has a different class distribution; there the upper cluster has 9 instances (45%), while there are visually two clusters in the lower half with 1 (5%) and 10 (50%) instances respectively, suggesting class C for the top cluster and classes E and A for the lower clusters. This seems plausible from the similarity of the distributions but is obviously wrong. The reason for this is that the assumption of identical class distributions is not met here.

The importance of identical class distributions is reinforced by the nearly ambiguous situation of two classes (A and C) having similar probabilities. Especially with small number of instances (9 vs. 10 in the unlabeled set) the distribution can easily be inverted with just two instances moved. Our method addresses this issue with bagging, i.e. we construct 10 slightly different data sets out of the unlabeled set, apply our method on each of them, and aggregate the resulting classifiers, accepting only modifications where most results agree (see also Section 6.4). The idea here is, that if a small change in the class distributions is resulting in a different labeling, then it is ambiguous and should be

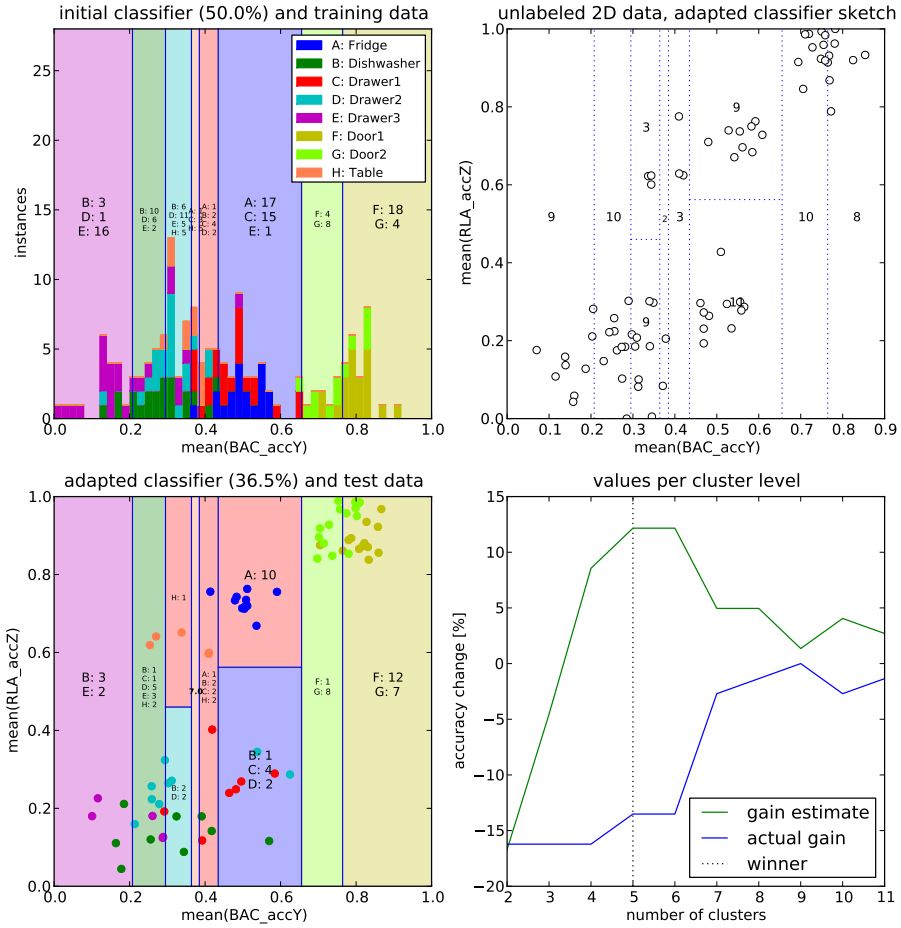


Figure 6.9: Example of negative outcome on the OPPORTUNITY data set.

avoided. Figure 6.10 displays all of the 10 bagging data sets and resulting classifiers for the example discussed here. Regarding the center region again, there are seven results which assign the wrong label C (red) to the upper part of the region. Thus, when accepting bagging results with 70% of agreement (which is the case in the example shown in Fig. 6.9) we are sticking to a wrong decision. In this case it massively influences the accuracy of the classifier because we are misclassifying exactly those instances that were classified correctly before. With a higher bagging threshold (e.g. 90%) we would avoid this mistake but at the same time we would miss the chance to improve in that region. We can avoid negative influence of noise up to a certain degree but it hinders to fully exploit the discriminative capabilities of the new sensor.

6.6 Conclusion

The promising results of the unsupervised method to integrate a new source of information into an existing classification system presented in Chapter 5 has

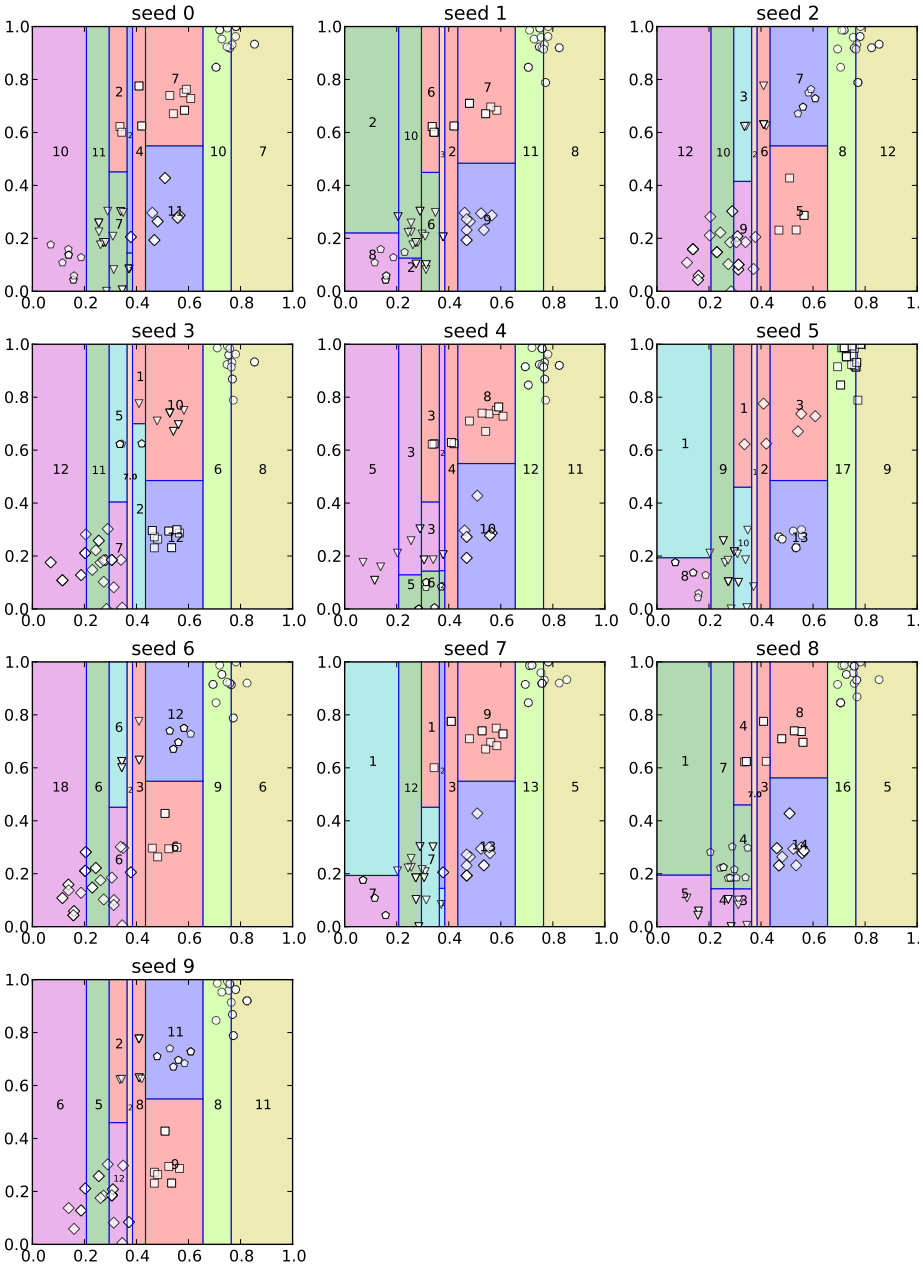


Figure 6.10: Detailed results for the 10 bagging datasets and -results of the example shown in Fig. 6.9 (same axes as upper right plot of that figure). The shape of the data points indicate their cluster membership. The colored regions show the resulting classifier for each seed.

motivated us to address the main drawbacks with an advanced method. In this chapter we presented the similarity search method which searches the most plausible among all possible combinations of labels for the clusters by comparing the resulting class distributions to the class distributions in the training data. In order to reduce the influence of noise on the labeling decision we integrated the bagging method and presented a technique for merging the aggregated results received from the different bagging data sets.

It is interesting to see that while on synthetic data the similarity search clearly outperforms the label inferring approach (in particular on the first data set with just $0.8\,\sigma$ distance between class centers), on real world data the conclusion is not clear. On one hand it can improve the performance in more cases. However, the ratio of the sum of positive change to the sum of negative change is less favorable. For the best configuration (B=2 and plTH=0.075) the ratio is over 2.5:1 for the label inferring approach and only just around 1.2:1 for similarity search. The bagging approach efficiently reduces the risk of negative change and raises this ratio across 3.5:1 with a bagging threshold of 0.9 (plTH 0.2). The results showed that bagging can effectively reduce the negative influence of noise in the data sets. The costs come at a limited exploit of the true information of the new sensor.

The approaches presented in this and the previous chapter are completely unsupervised methods for integrating a new sensor into an existing system as they only rely on the initial classifier model and unlabeled data from the new sensor. The training method used for the initial classifier is not relevant. In the following chapter we investigate how a small number of labels provided for data points in the extended feature space can be utilized for increasing the amount of improvement the system generates and for reducing the number of occasions where the system would decrease the initial performance.

Chapter 7

User Feedback to Support Integration of New Sensors

In succession to Chapters 5 and 6 which introduced unsupervised methods for integrating a new sensor to improve a a system's performance, we here investigate a semi-supervised variation of the method. From the unsupervised variants we learned that they can achieve perfect results in absence of sample noise. Though still positive and promising, results on real world activity recognition data sets were more limited. In this chapter we propose two semi-supervised extensions to the method that are designed to – with minimal user input – increase the amount of improvement that can be achieved on real world data sets, respectively decrease the number of occasions where the system's accuracy would be reduced. We analytically prove our approach and compare it to supervised and semi-supervised baseline methods applied to real world activity recognition data sets.

David Bannach, Bernhard Sick, and Paul Lukowicz. Automatic adaptation of mobile activity recognition systems to new sensors. In *Workshop Mobile sensing challenges, opportunities and future directions at Ubicomp 2011.*

The methods for integrating new sensors into existing classifiers described in Chapter 5 and 6 are unsupervised methods in the sense that they learn from unlabeled data how to modify the initial classifier model in order to achieve better classification accuracy with the help of an extra sensor. Nonetheless, those methods have similarities with semi-supervised learning methods with respect of how they utilize existing knowledge about the class distribution in the initial feature space to augment the unsupervised clustering of instances in the extended feature space with information about class membership. With the label inferring method (Chapter 5) we project the clustered instances into the original feature space and look for regions that are covered by a single cluster only. For those regions the class distribution found in the initial model may be assigned to the involved cluster. But finding such regions is not always possible. Thus, with the similarity search method (Chapter 6) we take an opposite

approach (in the way cluster labels are retrieved) and search for the cluster labels that are most plausible when comparing the resulting class distribution in the initial feature space with the original model. This allows us to always find labels for all clusters. Still, sample noise may cause wrong labelings to be favored occasionally.

The question we address in this chapter is, how our method performs if the user is providing a small amount of reliable labels for the instances recorded with the additional sensor. There are two basic strategies for utilizing such labels, apart from just training a new classifier with them. One would be to use them directly as trusted labels for the respective cluster structures in order to increase the effectiveness of the similarity search. The clusters being *pre-labeled* this way are regarded as "anchors" on which the method can base its search for suitable labels for the remaining clusters. The other strategy aims at detecting if a resulting classifier would perform worse than the initial one (*fault detection*), and consequently replace it with a better candidate or reject it to avoid accuracy decrease. Note that for this strategy labels are only needed in regions of the feature space where the classifiers disagree.

7.1 Chapter Outline

In this chapter we investigate an extended version of the similarity search method which supports semi-supervised learning, such that it can integrate user feedback in form of additional labels for instances recorded with the additional sensor. We present two possible approaches and for the second one, an active learning method, we analytically prove its positive influence on accuracy change. Moreover, we evaluate the method on the same synthetic and real-world data sets as in the previous chapters and compare the results to different baseline methods.

7.2 Related Work

Basically, the fields of *semi-supervised learning* (SSL) and – to a lower extent – also *active learning* (AL) are relevant to the work presented in this chapter.

SSL [Rat95, Cha06] makes use of both labeled and unlabeled data to train a prediction model. Therefore, SSL falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). SSL may be based on generative models such as probabilistic models (e.g., Gaussian mixture models), low density separation algorithms, or graph-based methods, for instance. The most common example for the second are the transductive support vector machines (TSVM) [Ben98] that build a connection between a density model for the data and the discriminative decision boundary by putting the boundary in sparse regions. Doing so, TSVM uses unlabeled data in a semi-supervised manner. A typical example for the third class are Laplacian SVM [GC08] that use a graph based model, a so called Lapalacian graph, for semi-supervised learning. A detailed overview of semi-supervised learning is given in [Zhu08], for example.

In the field of AL, membership query learning (MQL) [Ang88], stream-based active learning (SAL) [Atl90] and pool-based active learning (PAL) [Lew] are

the most important learning paradigms. MQL is not relevant here because it may generate artificial samples that cannot be understood or labeled by human experts [Set09]. SAL focuses, like our work, on sample streams. That is, each sample is typically "drawn" once, and the system must decide whether to query the expert or to discard the sample [Set09]. PAL builds a ranking on a given pool of unlabeled samples and, depending on a certain selection strategy, chooses a set of samples that must be labeled. A number of different selection exists for that purpose and four main categories can be distinguished: uncertainty sampling strategies select samples for which the considered classifier is most uncertain [Ton02, Con06, Set08], density weighting strategies consider the samples' distribution in the input space [Ngu04, Don07, Set08], estimated error reduction strategies aim at reducing the generalization error of the classifier directly [Mit04, Guo07], and diversity sampling strategies prevent the selection of "redundant" samples if more than one sample is selected in each query round [Dag06].

The two approaches we investigate in this chapter clearly fall within the SSL category as they both utilize partly labeled data. Yet, unlike classical SSL methods, we do not expect to have enough labeled instances to directly assign distributions to all clusters (pre-labeling) or to reliably detect bad classifiers (fault reduction). Instead, we rely on the amplifying effect of reliably labeled data points inserted into the, otherwise unsupervised, method presented in the previous chapter. The fault reduction approach additionally is a SAL method as it sequentially decides for each new instance if a label should be requested for it. The criterion for requesting a label is the disagreement of the two classifiers.

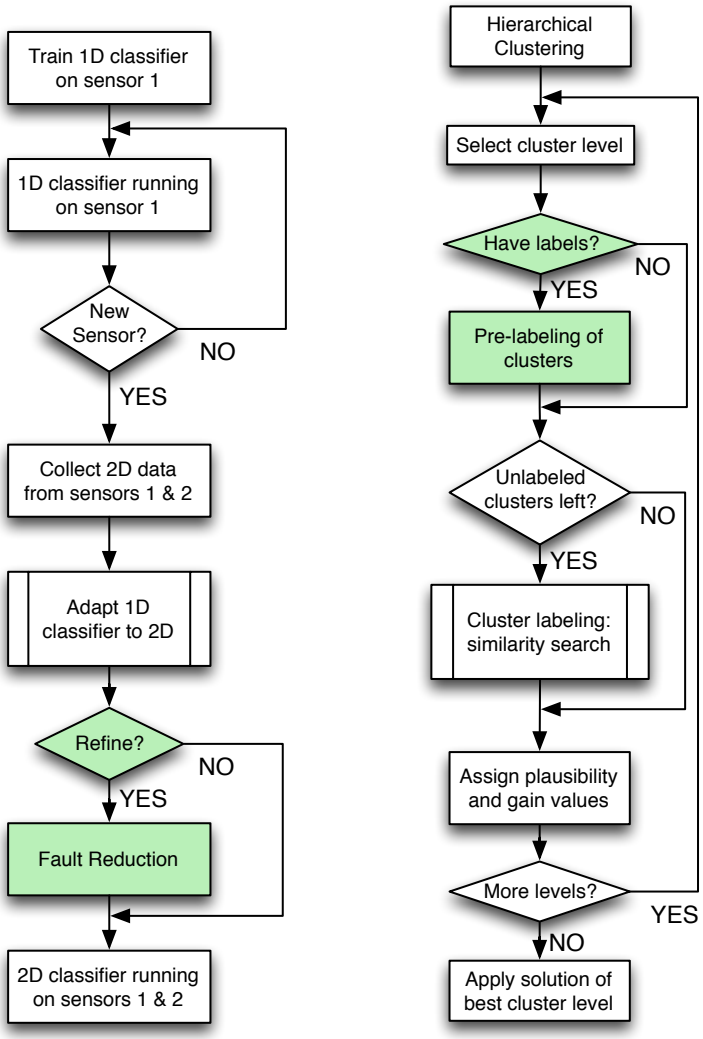
7.3 Approach 1: Pre-Labeling

For the pre-labeling approach we add an additional stage to the method that we presented in the previous chapter, just before any unsupervised labeling of clusters is happening. This is highlighted in Figure 7.1b. If, at that stage, any labels for 2D data points are available we proceed as follows: For each labeled data point

1. determine the cluster structure corresponding to the data point, and
2. add the label of the data point to the label distribution of the cluster.

Clusters for which no labeled data point is available will remain unlabeled at this stage. For those clusters the label distribution will be determined by the subsequent similarity search.

We expect the pre-labeling of clusters using expert knowledge to have an accelerating effect on the effectiveness of the similarity search, even with a very small amount of labels provided and only a few clusters affected. The dependency graph introduced in Section 6.3, with clusters as nodes and classifier regions as edges, clearly shows that a single pre-labeled cluster can influence the decision for many yet unlabeled clusters. The label information injected to one cluster propagates along the edges of the dependency graph and influences the labeling decision on all clusters within a connected component. Ambiguous situations, where multiple labelings for a subset of clusters are ranked equally well, may be resolved this way and can itself positively influence decisions on further nodes.



(a) Overview (b) Adapt 1D to 2D

Figure 7.1: Overview of the overall procedure (a) and its most important step, the classifier adaptation (b). The new steps compared to Figure 6.1 are highlighted

Note, in order to study the effects of the pre-labeling and fault reduction approaches we concentrate on the similarity search method alone, without the label inferring stage. We also leave away the bagging stage in order to simplify interpretation of results.

7.4 Approach 2: Fault Reduction

While the pre-labeling approach is primarily meant to increase the amount of performance improvement that the system can generate without increasing the number of occasions where it causes a performance decrease, the fault reduction approach aims at reducing the latter without reducing the first.

The fault reduction approach is using user-provided labels to directly detect and discard classifier variants that lead to performance degradation. The general idea is based on the observation that, to assess if a classifier using an additional sensor increases or decreases classification accuracy, we only need to consider data points where the new and the old classifier disagree. As a consequence we

1. run the old (1D) and the new (2D: old plus new sensor) classifier (C_1, C_2) in parallel,

2. actively ask the user for labels for instances where the classifiers C_1 and C_2 disagree, and

3. reject the new classifier C_2 if it does not fulfill a test criterion which requires C_2 to be correct at least a certain predefined number of times. In general, to keep the number of labels small, we consider only a few instances of disagreement between the old and the new classifier but require the new one to be correct in at least 75% of the cases.

As illustrated in Figure 7.1a, the fault reduction method is applied right after all cluster configurations have been labeled and assigned final gain and plausibility values. Without fault reduction, a solution with the best gain/plausibility trade-off (according to application-specific criteria) would be chosen (if it exists) and all other solutions discarded. With fault reduction we take a few (we found three to work well) best solutions and run them in parallel with the old classifier as described above. If a solution fails the test criterion, it is discarded. If in the end all solutions have been discarded then we consider the sensor combination to be not useful and do not use it (we get no improvement but also run no risk of a performance decrease). Otherwise, we pick the one with the best gain/plausibility trade-off from the "surviving" classifiers.

7.4.1 Success Probability Estimation

Obviously, such as all the methods presented in this work, fault reduction is a heuristic that works often but can not be guaranteed to be always right. Thus, we may have a new classifier that, overall, leads to a significant performance degradation, but the instances that we acquire labels for (which are random) happen to be just the ones on which it is right. On the other hand, we may reject a very good new classifier because of picking just the very few instances on which it is wrong. However, as shown below, the strength of our method is that it can be theoretically proven that it is significantly "skewed" in favor of keeping good classifiers and rejecting poor ones.

The two classifiers disagree exactly at the instances where C_1 is wrong but C_2 is correct ($\mathbb{D}_1$), and where C_2 is wrong but C_1 is correct ($\mathbb{D}_2$). This is illustrated in Figure 7.2.

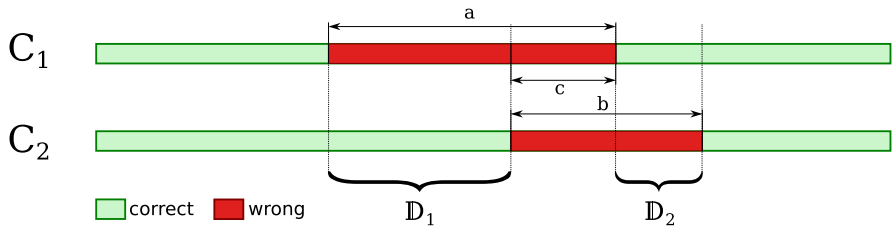


Figure 7.2: Schematic of two classifiers C_1 and C_2 applied to test data.

The probability that C_2 is correct for a randomly drawn instance from the disagreement set $\mathbb{D} = \mathbb{D}_1 \cup \mathbb{D}_2$ hence is

$$P_{C_2} = P(x \notin \mathbb{B} \mid x \in \mathbb{D}) = \frac{|\mathbb{D}_1|}{|\mathbb{D}_1| + |\mathbb{D}_2|} = \frac{a-c}{(a-c)+(b-c)}, \tag{7.1}$$

where $\mathbb{B}$ is the set of all instances classified wrongly by C_2, a the number of instances classified wrong by C_1, $b = |\mathbb{B}|$, and c the number of instances classified wrongly by both classifiers ($0 < c < \min(a, b)$). When expressing the improvement which C_2 achieves compared to C_1 as $v = a - b$ we receive $b = a - v$, and hence

$$P_{C_2} = \frac{a-c}{2(a-c)-v}. \tag{7.2}$$

If we accept the classifier C_2 only in case it is correct on all n randomly drawn instances from $\mathbb{D}$, we can express the probability of accepting the classifier as a function of the improvement v:

$$p(v) = \left(\frac{a-c}{2(a-c)-v}\right)^n. \tag{7.3}$$

From this function and Figure 7.3 we see that better performing classifiers ($v > 0$) are more likely to be accepted than those that perform worse than the initial classifier ($v < 0$). This effect is amplified with increasing number of tested instances n, but at the same time classifiers with small improvement tend to be rejected as well. With larger "overlap" c, the bad classifiers ($v < 0$) are strongly filtered out and even classifiers with small improvement get higher probability of being accepted, yet the maximal possible improvement ($v \leq a-c$) is more restricted by the "overlap". With two identical classifiers ($c{=}a, v{=}0$) the function is undefined. In that case there are no disagreement instances to test with ($\mathbb{D}{=}\emptyset$).

7.5 Evaluation

In general, we follow the same evaluation methodology as introduced in Chapter 5. Only for selecting the user-provided, labeled instances we slightly update the procedure.

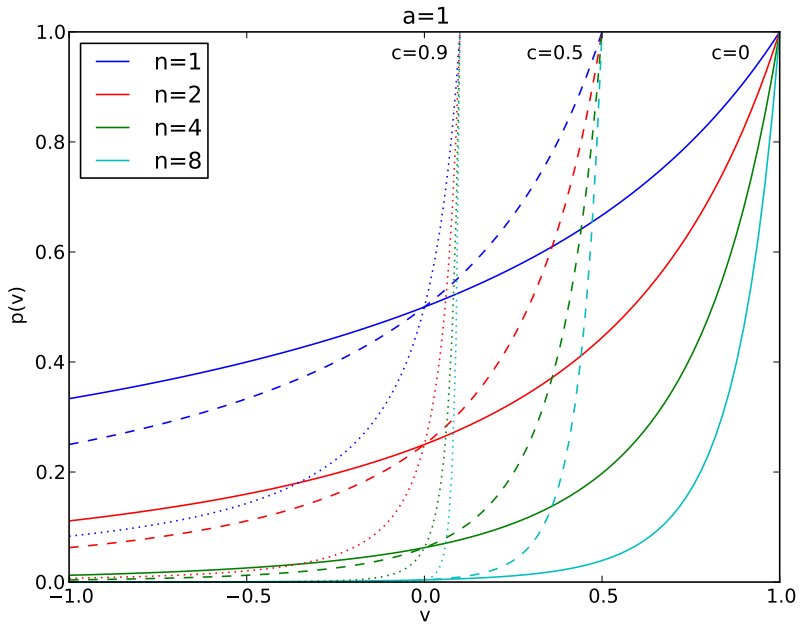


Figure 7.3: Probability of accepting classifier C_2 as a function of its improvement (v). The function is plotted for different "overlap" (c) and number of labeled instances drawn from $\mathbb{D}$ (n).

7.5.1 Evaluation Methodology Update

For evaluating the pre-labeling approach we proceed as follows. From the unlabeled 2D data set we chose the first n instances for which the user provides a label. In our case, obviously, we rely on expert annotations stored with the data set. The data sets are shuffled randomly before splitting into training-, improving-, and testing parts. Thus, the first n instances are considered as randomly drawn.

For the fault reduction approach we proceed nearly identically for choosing labeled 2D instances. We just select the first n instances that meet the method's requirements, i.e., the first n instances for which the classifiers disagree.

As a measure of the amount of user provided information in case of fault reduction, we use the number of disagreements between the old and the new classifier that have to be considered for the test criterion. Note that since we are running the test on several of the new classifiers this is not necessarily equal to the number of labels. In the best case it may be, in the worst case we need a different set of labels of each of the new classifiers. Thus, in the evaluation we provide the average number of labels that the system used, not the setting for the number of disagreements (which is our control parameter).

7.5.2 Baseline Methods

We compare our semi-supervised method to two different baseline methods.

2D Decision Tree Baseline

The decision tree baseline classifier is trained only on the 2D instances for which a label is provided. This classifier is ignoring any information from 1D training data or unlabeled 2D instances, just relying on supervised training. We set a minimal leaf size of 2 instances. Regarding the strongly limited number of available instances it would not make sense to be more restrictive.

EM/GMM Baseline

As a true representative of semi-supervised learning methods we compare to the Expectation Maximization algorithm (EM) applied on a gaussian mixture model. Nigam et al. [Nig00] have successfully used the EM algorithm to iteratively train text classifiers from a few labeled and many unlabeled documents. The model is initialized with the labeled instances and iteratively improved with unlabeled instances until the likelihood is maximized. In our case we apply gaussian mixture models (GMM) with multivariate normal distributions and unrestricted covariance matrix to meet the characteristics of our data sets. Note that with this model we are making an assumption about the functional form of the data which our method does not make. Depending on how well the data matches this model the EM/GMM baseline will achieve different accuracy. We must keep this in mind when comparing results.

The discrete feature of the reed switches from the OPPORTUNITY data set would lead to singularities in the covariance matrix of the normal distribution in case when all instances of a component show the same discrete value, causing a zero variance in that dimension. To avoid this situation we add artificial noise with small variance to that feature.

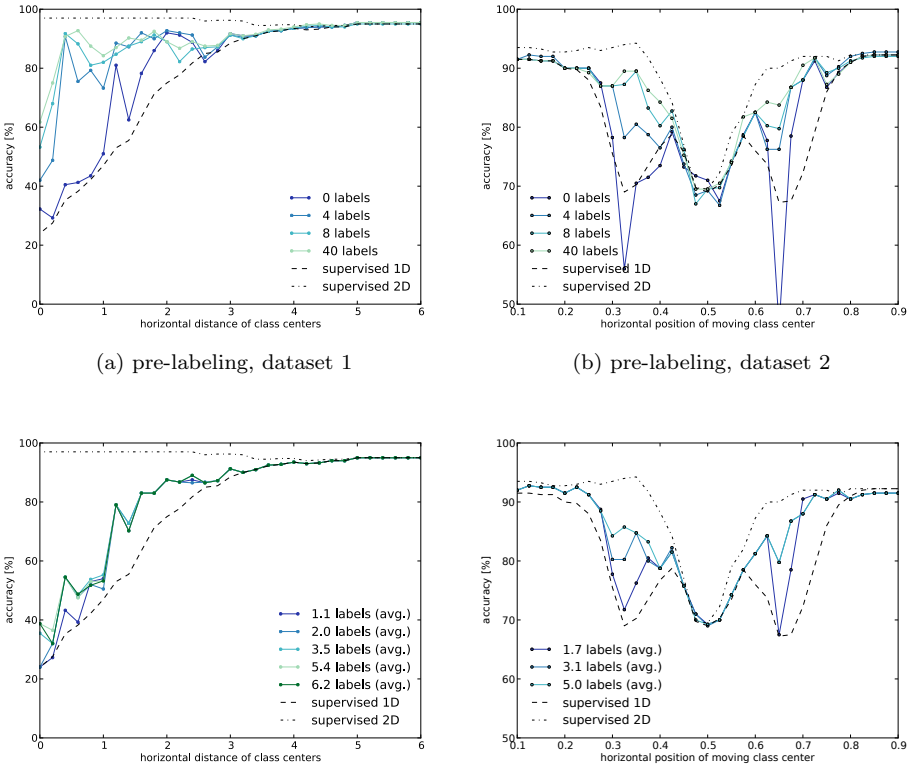
We choose the number of components in the GMM to match the number of classes. However, this might limit the performance in case of classes that are better modeled with more than a single component. Still we consider it a reasonable choice with respect to the data sets at hand.

7.5.3 Results on Synthetic Data Sets

Like in the previous chapters we begin by evaluating our method on the same synthetic data sets as in Section 5.4.1 which incorporate situations that resemble real-world conditions. Yet we concentrate the evaluations on the more challenging variants of synthetic data sets with just 10 2D instances per class.

Figure 7.4a displays results of the similarity search method with pre-labeling on the first synthetic data set with different amounts of labels provided. The results with only four user-provided labels already clearly outperforms the unsupervised method in situations with large cluster overlap ($\mathsf{dist}<1.2\sigma$) and can increase the accuracy to over 80% in most cases. When doubling the amount of 2D instances labels (eight) the overall performance still increases, but less rapidly and the improvement largely levels off beyond that. On the second synthetic data set, depicted in Figure 7.4b, pre-labeling with just 4 labels prevents the large performance decrease which the similarity search method had around a shift of 0.3 and 0.65, and achieves a reasonable improvement instead. This comes at the cost of decreased performance around shift of 0.5.

The effects of fault reduction best manifest on the second synthetic data set with just 10 learning instances shown in Figure 7.4d. We look at the critical



(a) pre-labeling, dataset 1

(b) pre-labeling, dataset 2

(c) fault reduction, dataset 1

(d) fault reduction, dataset 2

Figure 7.4: Average accuracy of the pre-labeling and fault reduction approaches on both synthetic data sets with 10 (unlabeled) learning instances per class (plTH 1.0).

regions around a shift of 0.3 and 0.65 for which the similarity search alone had lead to a performance decrease of over 30% (see Figure 6.4d). Even combined with an aggressive bagging threshold of 0.9 (i.e., 90% of solutions must agree when aggregating) a decrease of around 20% remained. As can be seen in Fig. 7.4d, fault reduction solves this problem without causing degradations at other places. The performance decrease totally disappears for an average of 1.7 labels and significant increase is seen from three labels on. This is achieved at the price of a less pronounced performance increase for simpler data sets such as the first synthetic one, where similarity search alone was already getting very good results (see Figure 7.4c).

7.5.4 Results on Real-World Data Sets

When applied to the real-world data sets the positive effect of pre-labeling is also clearly visible. Figures 7.5a and 7.5b show the results of pre-labeling in combination with the similarity search method (no bagging). In case of plTH 0.2

the number of sensor pairs with negative accuracy change decreases from 1486 (38.7%) down to 949 (24.7%) with just four labels and to 625 (16.3%) with eight labels (1.15 labels per class on average). At the same time the number of positive outcomes increase from 1583 (41.2%) to 2060 (53.6%) for four labels and to 2227 (58%) for eight labels. The improvement is particularly well visible in the sum over all improvements which goes from 15916 with no labels to 24499 with four and 28810 with eight. The results with plTH 0.075 behave similarly with increasing number of labels added, just from a slightly different starting point.

The histograms in Figure 7.5c confirm the anticipated effect for the fault reduction method. An average of four labels already reduces the number of faults from 1368 (35.6%) to just 320 (8.3%) while still keeping 1365 (35.5%) occasions where the accuracy is improved. With 11.2 user-provided labels the faults further decrease to 3.7% without affecting the positive outcomes too much (33.5%). Moreover, when combining pre-labeling (with eight labels) and fault reduction with 10.1 labels (Fig. 7.5d) the positive outcomes can even be increased to 49.3% with just 3.5% of negative ones.

Figure 7.6 puts the different variants of the method proposed in this and the two previous chapters in the context of two standard approaches directly trained on the new feature space. The first is a supervised tree classifier and the second one is a well know EM semi-supervised method in conjunction with Gaussian mixture models (see Section 7.5.2). For all methods we consider the real-world data sets. The evaluation criteria are the overall amount of improvement that a method can achieve when the new sensors are added (sum positive change) vs. the amount of performance decrease that it causes (sum negative change, see Section 5.4.4). For approaches that rely on user provided labels the results are plotted as a function of the number of available labels. For our unsupervised methods we show different thresholds reflecting different amounts of "risk" that the methods are taking.

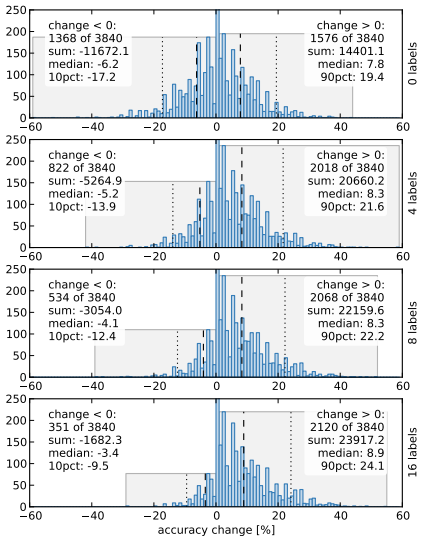
It can be clearly seen that our methods are in an entirely different region of the solution space than the standard supervised and semi-supervised approaches. The latter are meant for situations where

- A significant number of labels is available.
- The new sensor is consciously chosen in such a way that it is known to contain useful information and can be handled by the system. In other words there is no need to worry about potential performance degradation for unfavorable sensor configurations because we assume that the system designer makes sure that they do not occur.

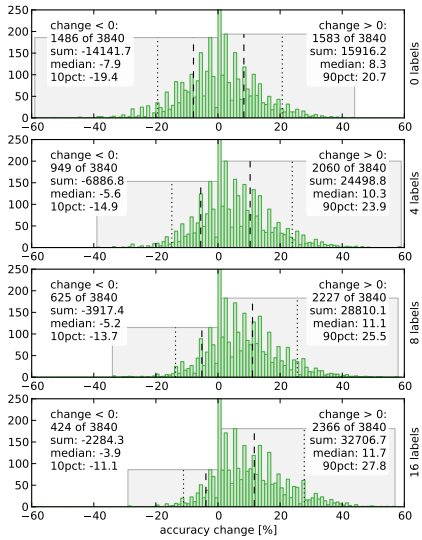
By contrast, our methods have been designed with a focus on

- No or minimal number of labels.
- Making sure that they "do as little harm as possible" when facing sensor combinations that are unfavorable. This is because our system is supposed to be able to handle any sensor combination that happens to be in the environment, not just combinations that have been designed to work.

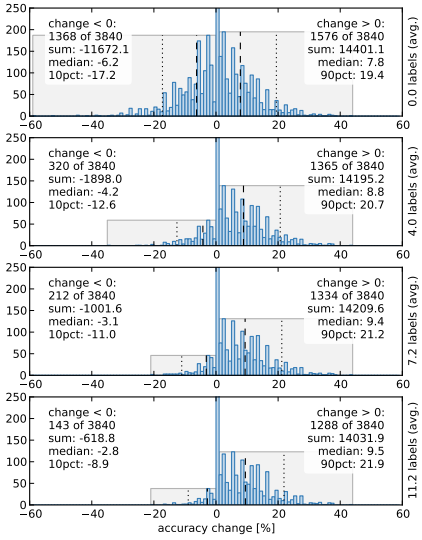
The above differences are best illustrated by the following observations from Figure 7.6:



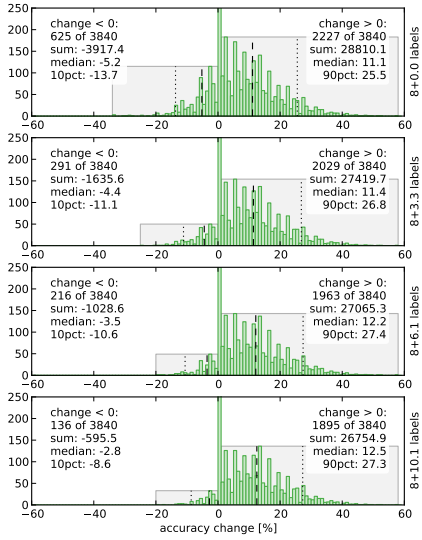
(a) pre-labeling, plTH 0.075



(b) pre-labeling, plTH 0.2



(c) fault reduction, plTH 0.075



(d) pre-labeling + fault reduction, plTH 0.2

Figure 7.5: Results for our pre-labeling and fault reduction method with different parameterizations on the three real-world data sets (13 subjects, $768 \cdot 5$ tested sensor combinations in total). Each plot shows results for four plausibility thresholds (plTH). Bounding boxes for positive and negative results are highlighted, including 10/50/90 percentiles.

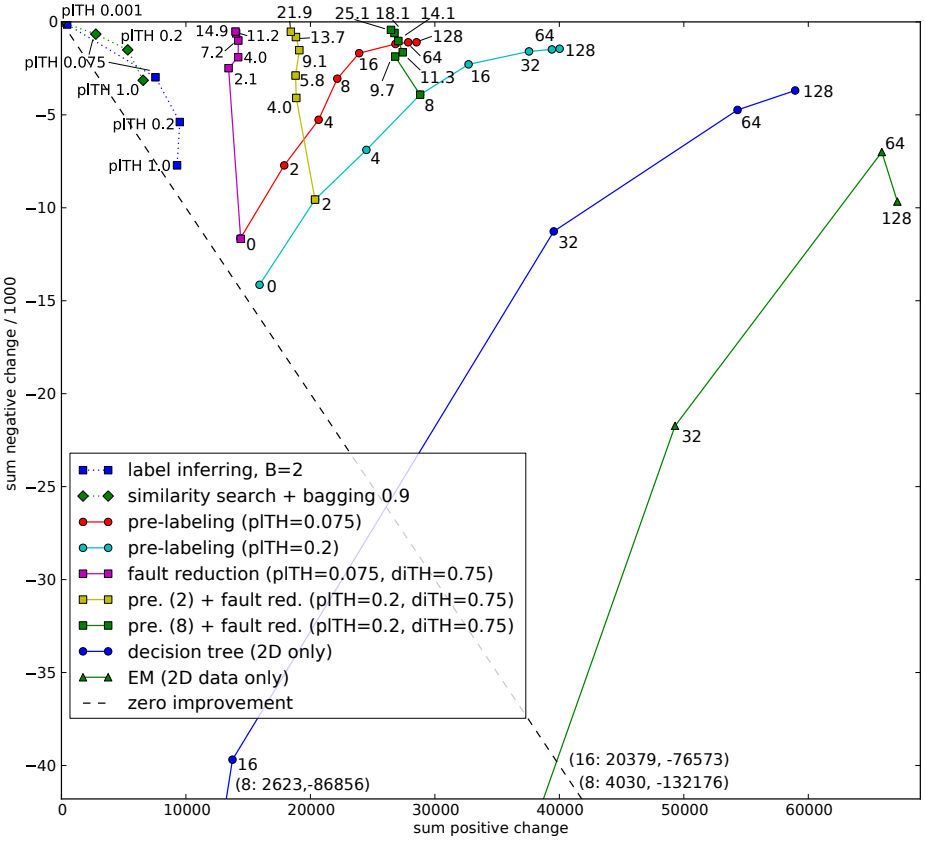


Figure 7.6: Comparison of results from the pre-labeling and fault reduction methods with previous results and semi-supervised baseline methods. Numbers in the graph denote the amount of user-provided labels if not stated otherwise.

- For 16 labels, the 2D tree classifier has about the same amount of overall improvement as the similarity search method with 0 labels (piTH 0.075) but nearly 4 times more negative change. Compared to the fault reduction method it also has the same amount of improvement but around 15 times more negative results for just 2 labels and over 70 times more negative results for 11 labels.

- For 16 labels, the EM semi-supervised method has just 60% of the improvement of our pre-labeling method (less than 15% with 8 labels) with more than 30 times as many negative results. It has about the same amount of improvement as the bagging supported similarity search with no labels while having 30 times more negative results.

- For 128 labels, the 2D tree classifier has about the same number of negative results as the pre-labeling version of our similarity search with 8 labels (while having about double as much improvement) and around 2 times as many negative results as the fault reduction method with just 7 labels (while having around 4 times more improvement).

Figure 7.6 also clearly displays the difference of the pre-labeling and fault reduction methods. While pre-labeling focuses on increasing the total amount of improvement using the user-provided labels, the fault reduction method aims at reducing the sum of negative accuracy change while holding the level of improvement. The fault reduction method increases the ratio of positive- vs. negative change for similarity search with plTH 0.075 from 1.2:1 to well over 5:1 with just 2 labels, and to over 14:1 with 7 labels. Pre-labeling with 16 labels reaches the same ratio with more than double the amount of improvement (plTH 0.2). When combining pre-labeling and fault reduction, a ratio of over 26:1 is achieved with 14 labels (8 for pre-labeling, 6.1 for fault reduction).

7.6 Conclusion

In this chapter we have extended the unsupervised method for automatic integration of new sensors into existing systems, which we presented in Chapter 6, to support semi-supervised learning. We proved the method analytically and we evaluated it on synthetic and real-world data sets by providing random labels for the extended feature space instances. We compared its performance to supervised and semi-supervised baseline methods.

The information of the added labels were integrated in two different ways. The first was to pre-label the clusters with the available labels in order to increase the effectiveness of the similarity search. In the second approach, which we called fault reduction, the labels were actively requested for just the feature space regions which are relevant to compare the performance of a candidate solution with the original classifier and to decide which one should be used. For the fault reduction method we could analytically show its ability to systematically prefer classifiers that yield better accuracy compared to the initial classifier and to discard the ones that would decrease the system's accuracy.

The evaluation on synthetic data sets showed that, with very few extra labels, the fault reduction approach can effectively prevent our method from eventually decreasing a systems performance and the pre-labeling approach significantly increases the amount of improvement. Those effects were also confirmed on real-world data sets. Compared to the semi-supervised baseline method (EM) with 16 labels, our method with pre-labeling achieved nearly double the sum of improvement and 30 times less amount of performance decrease. With less labels these numbers are more extreme.

The achieved results suggest our method of adapting the initial classifier to be superior over traditional semi-supervised and supervised methods in case when very limited amount of labels are available for the data recorded with the new sensor.

Chapter 8

Summary of Key Findings and Conclusion

Within this thesis we pursued the goal of making activity recognition better suitable for real world scenarios. The fundamental idea behind ubiquitous and pervasive computing demands that the technology itself should be invisible to the user. In the context of activity recognition systems this would mean that the system must be accurate, because wrong results would immediately attract the user's attention. Additionally, such a system must be easily maintainable in the long term and should therefore adapt to the sensor hardware that is available, instead of dictating the set of sensors a user must wear.

In the two parts of this thesis we explored those two basic requirements. We started with software tools that support and facilitate the creation of activity recognition systems from the very beginning of a project, when an initial data set is recorded from diverse sensors, until the deployed online activity recognition system. In the second part we proposed and evaluated a novel method for opportunistic activity recognition systems which enables existing systems to integrate new, unknown sensors in order to improve the recognition accuracy with no or with minimal user input.

8.1 Activity Recognition Toolchain

In the first part we introduced the CRN Toolbox, a reconfigurable software toolbox to ease the process of building online activity recognition systems (Chapter 2). It is based on a repository of parameterizable data stream processing components which can be connected in the desired order or topology to define the data flow. The toolbox has been used in various research projects and scientific publications already. Furthermore, we proposed a novel method for automatic synchronization of data streams from heterogenous sensor networks (Chapter 3). The method is based on detecting physical events in data streams received from different sensors and then finding the corresponding event on each data stream for alignment. The method works across different sensing modalities and we were able to achieve automatic synchronization with less than 0.3 s error in a multi-user real world data set. Ultimately, we introduced additional software tools which we integrated to a complete toolchain that supports all

phases of the development of activity recognition systems (Chapter 4). In sum those tools cover

- the recording of large, multimodal data sets from a variety of sensing systems (CRN Toolbox, smartphone data loggers) and monitoring of the progress (MASS),
- storing, annotating, and sharing of such data sets (ContextDB.org, Labeling Tool),
- extracting of reproducible traces from selected parts of the database which can be used as base for classifier training and evaluation (Trace Generator), and
- online activity recognition from live sensor data streams (CRN Toolbox) and dynamic sharing of sensor data and recognition services among multiple Toolbox instances (service oriented extension).

The toolchain has been developed and implemented over the process of multiple activity recognition research projects and has most recently been used to establish a large-scale data set with 72 sensors of 10 different modalities mounted on body, objects, and in the environment, with totally 60 recording sessions annotated with nearly 40 000 high- and low-level activity labels.

Its utilization in various projects proved the applicability of the CRN Toolbox' concept and user studies confirmed an adequately short learning curve. New activity recognition systems could be designed and tested conveniently on desktop computers and later transferred to small, wearable computers without the need of modifications or re-coding. Additional algorithms could either be implemented as a Toolbox component or just integrated in the data flow using TCP sockets, and most importantly, the components could be re-used with different parametrizations in later applications. The acceptance of the CRN Toolbox among research groups was promising and the main reason for researchers and developers to hesitate adopting the concept despite its advantages seemed to be the fear of coding additional components in C/C++ and the strictly data-driven design. This could be alleviated in future by providing a more complete set of available algorithms and a thorough documentation. Furthermore, generic components that allow the usage of scripting language for defining the component's logic could facilitate adoption.

A frequent problem with recording of data streams from multiple sensors is their proper synchronization. As manual alignment of the signals may be feasible in some cases of offline analysis, it is just not practical for real world activity recognition systems. The method for automatic, event-based synchronization of data streams we presented in this thesis proved to be a feasible solution to the problem. Using very simple event detectors the method was able to correctly align the events found on multiple signals. Moreover, the transitive property of this method even allowed for synchronization of sensors with completely different modality, i.e., sensors which are not able to sense a common physical event. While the event detectors were already implemented within the CRN Toolbox, the synchronization method was optimized for offline analysis and yet needs to be implemented as Toolbox task. A Toolbox user would configure the synchronization task to detect the desired events on the individual data streams and the task would then automatically determine the clock offsets between those

data streams and correct the timestamp of each transmitted data packet accordingly. Later tasks in the network can then safely merge data streams according to timestamps.

We introduced artificial synchronization actions to our experiments (e.g., clapping hands, push/release of a button) to explicitly synchronize the involved sensors. Yet, we are confident that future work will be able to utilize more subtile actions which naturally appear frequently enough for synchronizing the sensors. This will further reduce undesirable user interaction in real life deployments and hence increase the system's invisibility.

We are convinced that the integrated toolchain presented in thins thesis significantly reduces the effort for creating new activity recognition systems. It facilitates the tedious tasks of recording and annotating large, multimodal data sets and it allows for sharing them with the community to effectively reduce the necessity of recording new data sets and to increase the quality of the existing ones instead. As the effort of creating activity recognition systems is reduced we believe that the toolchain can actually accelerate research. Moreover, with the easy access to a context database we expect to attract an even larger community which do not have the resources for recording own activity recognition data sets.

8.2 Opportunistic Activity Recognition

Within the second part of this thesis we proposed and studied a novel method for automatic adaptation of activity recognition systems to new sensors. Our method allows an opportunistic activity recognition to benefit from new sensors, which were not known at design time, without or with minimal user input. It thus enables a system to autonomously evolve together with its changing sensing environment.

The core of our method is based on well-established principles of semi-supervised learning. However, traditional semi-supervised methods usually combine unsupervised clustering with sparse label information retrieved from the user to infer labels for points throughout the clusters. In our case we do not rely on label information from the user at first, but we utilize the initial classifier model (n dimensions) to generate labels for the new, n+1 dimensional data points. As such, our approach can be categorized as unsupervised method. Nevertheless, the variations of our method that additionally incorporate a limited amount of user feedback actually make it a semi-supervised method again.

In an optimal sensor setup, i.e., when the classifier trained on the initial sensor perfectly recognizes each activity class in at least some regions of the 1-dimensional feature space but confuses the classes in other regions, and the 2-dimensional feature space extended with the new sensor perfectly separates the classes, in that case the simplest version of our method already reveals an adapted classifier with perfect accuracy without any labeled data from the new sensor. Not surprisingly, such setups are rarely found in real world activity recognition scenarios. This has two fundamental effects:

1. the regions of labeled data points in the initial feature space that can be used to infer labels for single clusters are strongly reduced (single cluster regions), and

2. hence the sample noise is strongly amplified when applying those label distributions to entire clusters in the extended feature space.

In the advanced approach we avoided to infer cluster labels from small samples by directly choosing the cluster labels such that the resulting distribution is most similar to the original training data when projected back to the initial dimension. While this second approach evades the above effects at first, it happens to be affected by similar symptoms originating from differences in the two sample distributions that are once more amplified from a small sample size. Yet, with the bagging technique we found an efficient way to reduce the influence of sample noise to our method. The occasions where integrating the new sensor would reduce overall accuracy could be significantly reduced with bagging.

Another efficient way to reduce occasions where the adapted system would reveal lower accuracy than the initial system and to increase the amount of improvement that can be achieved, was to include user feedback. We could analytically show that for small number of labels this heuristic is already strongly biased towards keeping good solutions (which achieve better accuracy than the initial system) and discarding bad ones. With just two labels requested from the user, our method already performed better than the baseline methods with 64 or 128 labels when regarding the sum of negative improvement over all tested sensor pairs. In other words, our method better handled the cases where the new sensor delivered nonsense data, i.e., where the baseline 2D classifiers achieved lower accuracy than the initial 1D classifier. In terms of overall accuracy gain (sum of positive accuracy change) our method significantly outperformed both baseline methods (fully supervised and semi-supervised) when the number of labeled data points was low, and with 32 labels we still achieved over 75% of the improvement which the semi-supervised baseline realized and 10 times less negative accuracy change.

We have chosen the decision tree classifier for evaluating our method as it proved a suitable model for real life activity recognition systems in previous work, and because it can be easily refined locally with new dimensions. For future work we expect that using more sophisticated classifier models, such as mixtures of discriminative and generative models, could further increase the effectiveness of our method. At the same time, the heuristic used to estimate plausibility could be replaced by a proper measure of the difference between two probability distributions such as the Kullback-Leibler divergence.

In the context of discussing this method we referred to a "sensor" as a single, 1-dimensional feature extracted from a sensor. And for simplicity, we limited our investigation to those configurations where the initial classifier was trained with a single feature (the "old sensor") and was extended with a different feature (the "new sensor"). In real world scenarios, however, the addition of a new sensor (e.g., a 3-axis gyroscope) would provide a whole set of new features which is primarily limited by the amount of available feature extraction algorithms. Yet, as our method includes heuristics to rate individual solutions on their expected accuracy gain, it allows for selecting the most promising feature. Still, the concept of our method may be extended to support addition of multiple features, e.g., by incrementally going from N to $N+1$ dimensions.

8.2.1 Lessons Learned

The methods presented in Part II of this thesis are a result of incremental experiments with real world activity recognition data. Exploring and tuning of the methods was an essential part of this work. It revealed another field for tools that support such exploration processes. The problem here is that the methods, which take activity recognition data as input, may produce intermediate results at different stages. Storing those intermediate results and using them at subsequent stages may strongly reduce the time needed for executing the later stages, and hence the effort for exploring the methods. However, without a thoroughly designed architecture for managing code and intermediate data, which usually is not the first priority when exploring new methods, the intermediate results and different code versions can quickly get difficult to manage, e.g., when manipulating an earlier stage after several versions of a subsequent stage exist already. This calls for tools that support managing and versioning of hierarchically linked processing stages with their intermediate results. As suggested already in Chapter 4, a possible solution would be to store the code and intermediate results for each stage within the same context database, implement versioning, and provide scripts that dynamically execute the necessary stages and update the intermediate data.

We have restricted the evaluation of our methods to a selected set of sensors and features and to the 1D-to-2D case in order to limit the effort needed for assessing the method's performance on each sensor combination. Higher-dimensional feature spaces would require even more data for learning the initial decision tree (training set) and clustering of the extended feature space (adaptation set). Yet, with the limited amount of available data it would only emphasize the effects of limited sample size. Still we believe that given the proper amount of data it would be possible to successfully apply our method onto the N-to-(N+1) or N-to-(N+M) case. Compared to traditional, fixed feature set activity recognition systems which only require a single training set (and a test set), our methods additionally require a data set for adaptation, recorded from the same setup but with the extra sensor added. Also, our methods are sensitive to discrepancies between those data sets (i.e., different probability distributions).

The general N to N+M dimensions case could be solved by sequentially adding the feature which results in the best gain value until a gain threshold is no longer exceeded. Such a sequential procedure could, however, only exploit information from statistically independent features and error propagation could become an issue. Hence, adapting our methods to allow for considering multiple additional dimensions at once would be beneficial but it still cannot evade the issues known as *curse of dimensionality*.

Our methods may improve the recognition performance of classes which were recognized poorly or not at all with the initial system, but there must exist at least some labeled data points for those classes in the initial training data. New classes, which were not known at design time of the initial system can not be added. It would require other methods to integrate a new class. Still, in certain cases, our methods could provide cues for new classes when the clustering phase would reveal clearly separated clusters for a single class. This is limited to the cases where the activity associated to the new class was already performed in the initial training phase but not labeled yet. The other case, when the subject introduces a new activity at runtime, would violate the assumption of

equal distributions in training- and adaptation set. This requirement could be overcome if the introduction of the new activity could be detected (e.g., by detecting changes in the probability distributions) and separated into a new cluster. It would enable us to still compare the two data sets and treating the new cluster separately.

8.3 Outlook

The current implementation of our method for automatically integrating new sensors is clearly targeted for offline analysis. Naturally, it needs some adaptation to be suitable for online activity recognition systems but we expect those modifications to be marginal. There is no need for this method to incrementally execute as new data points arrive, rather it would buffer the data until enough information is collected for integrating the new sensor. While an implementation as a CRN Toolbox task would certainly be possible we would prefer implementing it at a higher level, beyond the concept of stream tasks. This is because it does not produce continuous output which would be processed by further tasks in the Toolbox setup, but it would instead adapt the Toolbox configuration at certain times. Specifically, this would include

1. adding reader- and feature extraction tasks for sensors that appear,
2. collecting a predefined amount of data from the new sensor,
3. applying the method for extending the existing classifier model with the new dimension, and
4. writing the new classifier model to the classifier task and adapting the data flow in the Toolbox configuration.

This higher layer utilizes the efficient CRN Toolbox runtime, which usually was manually configured, and makes it suitable for the autonomous evolution of opportunistic activity recognition systems, by automatically adapting the Toolbox configuration.

The automatic synchronization method presented in Chapter 3 would fit well as a group of CRN Toolbox tasks. Unlike the opportunistic method above, it has no need to reconfigure the Toolbox configuration. instead, it actually has to constantly detect events on all data streams. Those event detectors may be built from multiple Toolbox tasks which are commonly used. The core of the synchronization method would be implemented in a dedicated task which would periodically correlate the detected events as described in Chapter 3 and compute the alignment offset for each data stream. Those offsets would then be applied to each data stream in a separate task. The event detectors used in the evaluation of our synchronization method are already implemented using Toolbox tasks. Only the task for correlating the events to derive the data stream offsets must be ported to the Toolbox yet. The entire synchronization method should be wrapped into a task group which automatically instantiates the necessary tasks and connections for each data stream such that only the dedicated event detectors would have to be configured manually.

In Chapters 5–7 we explored three different sources of labels for labeling the clusters and applied them separately or in sequence: single-cluster regions,

distribution similarity, and user feedback. Future work could search for ways to combine different label sources in a weighted manner in order to better exploit their individual strengths and to allow for integrating new label sources more formally. Furthermore, the reasons for clusters that receive labels of multiple classes and hence are labeled with a class distribution instead of a single label, could be explored and possibly exploited for improved performance.

Future work could also extend the opportunistic method to not just allow for adding of new sensors but also dynamically removing or re-adding of previously known sensors to even better suit real life scenarios. Such a system would for each sensor keep a record that contains the accumulated knowledge about the sensor's contribution to the recognition goal. Using those records the system can quickly decide which sensors and which features to include in the recognition process. At a further stage, such knowledge records could also be retrieved from the public context database in case there are enough recordings uploaded for the specific scenario. Specifically, sensor type, sensor positioning (location and orientation), and labeled activities must match for the recordings to be useful. In some cases, however, recorded sensor data could be transformed to match the specific setup. Following this approach, the context database that was initially intended for researchers to share data sets for evaluating and comparing activity recognition algorithms would then be directly used by the opportunistic methods themselves.

Simplifying and structuring the process of creating activity recognition systems allows researchers to better focus on real world applicability of the methods, and it may enable those methods to themselves take advantage of the structured process.

Bibliography

[Abo99] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In H.-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66550-2.

[Ada08] K. Adamer, D. Bannach, T. Klug, P. Lukowicz, M. L. Sbodio, M. Tresman, A. Zinnen, and T. Ziegert. Developing a wearable assistant for hospital ward rounds: An experience report. In *Proceedings of the International Conference on Internet of Things*, volume 4952 of *Lecture Notes in Computer Science*, pages 289–307. 2008.

[Agü12] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julián. Developing Pervasive Systems as Service-oriented Multi-Agent Systems. pages 78–89, 2012.

[Amf04] O. Amft, M. Lauffer, S. Ossevoort, F. Macaluso, P. Lukowicz, and G. Troster. Design of the qbic wearable computing platform. In *Application-Specific Systems, Architectures and Processors, 2004. Proceedings. 15th IEEE International Conference on*, pages 398 – 410. 2004. ISSN 1063-6862.

[Amf07] O. Amft, M. Kusserow, and G. Tröster. Probabilistic parsing of dietary activity events. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, pages 242–247. 2007.

[Amf08] O. Amft and G. Tröster. Recognition of dietary activity events using on-body sensors. *Artificial Intelligence in Medicine*, 42(2):121–136, 2008.

[Amf10] O. Amft, D. Bannach, G. Pirkl, M. Kreil, and P. Lukowicz. Towards wearable sensing-based assessment of fluid intake. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 298 – 303, 2010.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Atl90] L. Atlas, D. Cohn, R. Ladner, M. A. El-Sharkawi, and R. J. Marks, II. Training connectionist networks with queries and selective sampling.

In *Advances in Neural Information Processing Systems 2*, pages 566–573. Morgan Kaufmann, Denver, CO, 1990.

[Ban06] D. Bannach, K. Kunze, P. Lukowicz, and O. Amft. Distributed modular toolbox for multi-modal context recognition. In *Proceedings of the 19th International Conference on Architecture of Computing Systems*, volume 3894 of *LNCS*, pages 99–113. Springer, 2006.

[Ban07] D. Bannach, O. Amft, K. Kunze, E. Heinz, G. Tröster, and P. Lukowicz. Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 32–39. 2007.

[Ban10] D. Bannach, K. Kunze, J. Weppner, and P. Lukowicz. Integrated tool chain for recording and handling large, multimodal context recognition data sets. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Ubicomp '10, pages 357–358. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0283-8.

[Bao04] L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. In A. Ferscha and F. Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-21835-7.

[Bec04] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - a component system for pervasive computing. In *Proceedings of the Second IEEE Conference on Pervasive Computing and Communications*, pages 67–76. 2004.

[Ben98] K. Bennet and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 11*, pages 368–374. MIT Press, 1998.

[Bia05] X. Bian, G. Abowd, and J. Rehg. Using Sound Source Localization in a Home Environment. In *Proc. of The 3rd Intl. Conference on Pervasive Computing*, pages 19–36. Springer, 2005.

[Blu98] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. pages 92–100, 1998.

[box] BoxLab Visualizer. `http://boxlab.wikispaces.com/Visualizer`. [Online; accessed 13-Jan-2011].

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Bru00] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In P. Thomas and H.-W. Gellersen, editors, *Handheld and Ubiquitous Computing*, volume 1927 of *Lecture Notes in Computer Science*, pages 12–29. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41093-5.

[Cal10] A. Calatroni, D. Roggen, and G. Tröster. A methodology to use unknown new sensors for activity recognition by leveraging sporadic interactions with primitive sensors and behavioral assumptions. In *Proc. of the Opportunistic Ubiquitous Systems Workshop, part of 12th ACM Int. Conf. on Ubiquitous Computing*. 2010.

[Cal11] A. Calatroni, D. Roggen, and G. Tröster. Automatic transfer of activity recognition capabilities between body-worn motion sensors: Training newcomers to recognize locomotion. In *Eighth International Conference on Networked Sensing Systems (INSS'11)*. Penghu, Taiwan, 2011.

[Cha06] O. Chapelle, B. Scholkopf, and A. Zien. *Semi-supervised learning*, volume 2. MIT Press, 2006.

[Che08a] J. Cheng, D. Bannach, K. Adamer, T. Bernreiter, and P. Lukowicz. A wearable, conductive textile based user interface for hospital ward rounds document access. In *Smart Sensing and Context*, volume 5279 of *Lecture Notes in Computer Science*, pages 182–191. 2008.

[Che08b] J. Cheng, D. Bannach, and P. Lukowicz. On body capacitive sensing for a simple touchless user interface. In *Medical Devices and Biosensors, 2008. ISSS-MDBS 2008. 5th International Summer School and Symposium on*, pages 113 –116. 2008.

[Che09] L. Chen and C. Nugent. Ontology-based activity recognition in intelligent pervasive environments. *International Journal of Web Information Systems*, 5(4):410–430, 2009. ISSN 1744-0084.

[Cho08] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, et al. The mobile sensing platform: An embedded activity recognition system. *Pervasive Computing, IEEE*, 7(2):32–41, 2008.

[cmu] Kitchen Capture. `http://kitchen.cs.cmu.edu/`. [Online; accessed 13-Jan-2011].

[Con06] C. Constantinopoulos and A. Likas. Active learning with the probabilistic RBF classifier. In *Artificial Neural Networks–ICANN 2006*, volume 4131 of *Lectures Notes in Computer Science*, pages 357–366. Springer, Berlin, Germany, 2006.

[Cos07] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. The RUNES middleware for networked embedded systems and its application in a disaster management scenario. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 69–78. 2007.

[Cou01] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems*. Addison-Wesley Reading, Mass, 2001.

[Cro06] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), 2006.

[Dag06] C. K. Dagli, S. Rajaram, and T. S. Huang. Utilizing information theoretic diversity for SVM active learning. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR '06)*, pages 506–511. Hong Kong, China, 2006.

[Dai07] W. Dai, G.-R. Xue, Q. Yang, and Y. Yu. Co-clustering based classification for out-of-domain documents. pages 210–219, 2007.

[Dau07] H. Daumé, III. Frustratingly easy domain adaptation. *ACL*, 2007.

[Dea08] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Dey01] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16(2-4):97–166, 2001.

[Don07] P. Donmez, J. G. Carbonell, and P. N. Bennett. Dual strategy active learning. In *Proceedings of the 18th European Conference on Machine Learning (ECML '07)*, pages 116–127. Warsaw, Poland, 2007.

[Dua12] L. Duan, D. Xu, and I. Tsang. Learning with Augmented Features for Heterogeneous Domain Adaptation. *arXiv.org*, 2012.

[Edw03] K. Edwards, V. Bellotti, A. K. Dey, and M. Newman. Stuck in the middle: The challenges of user-centered design and evaluation for middleware. In *Proceedings of the Conference on Human Factors in Computing Systems*. 2003.

[End05] C. Endres, A. Butz, and A. MacWilliams. A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems*, 1(1):41–80, 2005.

[Far99] J. Farringdon, A. Moore, N. Tilbury, J. Church, and P. Biemond. Wearable sensor badge and sensor jacket for context awareness. In *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*, pages 107 –113. 1999.

[För09] K. Förster, D. Roggen, and G. Tröster. Unsupervised classifier self-calibration through repeated context occurences: Is there robustness against sensor displacement to gain? In *Wearable Computers, 2009. ISWC '09. International Symposium on*, pages 77 –84. 2009. ISSN 1550-4816.

[Gar02] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: toward distraction-free pervasive computing. *Pervasive Computing, IEEE*, 1(2):22 –31, 2002. ISSN 1536-1268.

[GC08] L. Gomez-Chova, G. Camps-Valls, J. Munoz-Mari, and J. Calpe. Semisupervised image classification with laplacian support vector machines. *Geoscience and Remote Sensing Letters, IEEE*, 5(3):336 –340, 2008. ISSN 1545-598X.

[Gel02] H. Gellersen, A. Schmidt, and M. Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mobile Networks and Applications*, pages 341–351, 2002.

[Gua07] D. Guan, W. Yuan, Y. Lee, A. Gavrilov, and S. Lee. Activity Recognition Based on Semi-supervised Learning. In *Proc. of the 13th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications*, pages 469–475. 2007. ISBN 0769529755.

[Gün10] H. Günther, F. E. Simrany, M. Berchtold, and M. Beigl. A tool chain for a lightweight, robust and uncertainty-based context classification system (ccs). In *Proceedings of the 1st Workshop on Context-Systems Design, Evaluation and Optimisation (CosDEO 2010)*. VDE Publishing House, Hannover, Germany, 2010.

[Guo07] Y. Guo and R. Greiner. Optimistic active learning using mutual information. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI '07)*, pages 823–829. Hyderabad, India, 2007.

[Gup09] J. Gupchup, R. Musaloiu-E, A. Szalay, and A. Terzis. Sundial: Using Sunlight to Reconstruct Global Timestamps. In *Proc. of The 6th European Conference on Wireless Sensor Networks*. 2009.

[Har02] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Wireless Networks*, 8:187–197, 2002. ISSN 1022-0038.

[Hil00] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not*, 35(11):93–104, 2000. ISSN 0362-1340.

[Huy06] T. Huynh and B. Schiele. Unsupervised discovery of structure in activity data using multiple eigenspaces. *Lecture notes in computer science*, pages 151–167, 2006. ISSN 0302-9743.

[Int06] S. Intille, K. Larson, E. Tapia, J. Beaudin, P. Kaushik, J. Nawyn, and R. Rockinson. Using a live-in laboratory for ubiquitous computing research. In *Pervasive Computing*, pages 349–365. 2006.

[Joh97] G. W. Johnson. *LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control*. McGraw-Hill School Education Group, 1997.

[Kas08] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-136-1.

[Kip01] M. Kipp. Anvil - a generic annotation tool for multimodal dialogue. In *European Conference on Speech Communication and Technoology*, pages 1367–1370. 2001.

[kit] TUM Kitchen data labeling tool. `http://ias.cs.tum.edu/download/kitchen-activity-data/labeling-tools`. [Online; accessed 13-Jan-2011].

[Kra03] A. Krause, D. P. Siewiorek, A. Smailagic, and J. Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, ISWC '03, pages 88–. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-2034-0.

[Kun05] K. Kunze, P. Lukowicz, H. Junker, and G. Troester. Where am i: Recognizing on-body positions of wearable sensors. *LOCA'04: International Workshop on Location and Context-Awareness*, 2005.

[Kun08] K. Kunze and P. Lukowicz. Dealing with sensor displacement in motion-based onbody activity recognition systems. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 20–29. 2008. ISBN 978-1-60558-136-1.

[Kun09] K. Kunze, P. Lukowicz, K. Partridge, and B. Begole. Which way am i facing: Inferring horizontal device orientation from an accelerometer signal. In *Proc. of Int. Symp. on Wearable Computers (ISWC)*, pages 149–150. IEEE Press, 2009.

[Kun10] K. Kunze, G. Bahle, P. Lukowicz, and K. Partridge. Can magnetic field sensors replace gyroscopes in wearable sensing applications? In *Proc. 2010 Int. Symp. on Wearable Computers*. 2010.

[Lae] K. van Laerhoven, M. Berchtold, and S. Reeves. Cstk (commonsense toolkit). `http://cstk.sourceforge.net/`. [Online; accessed 13-Jan-2011].

[Lee02] S.-W. Lee and K. Mase. Activity and location recognition using wearable sensors. *Pervasive Computing, IEEE*, 1(3):24 – 32, 2002. ISSN 1536-1268.

[Les04] J. Lester, B. Hannaford, and G. Borriello. "Are You with Me?"-Using Accelerometers to Determine If Two Devices Are Carried by the Same Person. *Lecture Notes in Computer Science*, pages 33–50, 2004.

[Les06] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. *Pervasive Computing*, pages 1–16, 2006.

[Lew] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '94)*, pages 3–12. Dublin.

[Li04] S. Li, Y. Lin, S. Son, J. Stankovic, and Y. Wei. Event detection using data service middleware in distributed sensor networks. *Telecommun Syst*, 26(2-4):351–368, 2004. Special issue on Wireless Sensor Networks.

[Luk07] P. Lukowicz, A. Timm-Giel, M. Lawo, and O. Herzog. Wearit@work: Toward real-world industrial wearable computing. *Pervasive Computing, IEEE*, 6(4):8 –13, 2007. ISSN 1536-1268.

[Luk09] M. Lukac, P. Davis, R. Clayton, and D. Estrin. Recovering Temporal Integrity with Data Driven Time Synchronization. In *Proc. of The 8th Intl. Symposium on Information Processing in Sensor Networks*. 2009.

[Luk10] P. Lukowicz, G. Pirkl, D. Bannach, F. Wagner, A. Calatroni, K. Förster, T. Holleczek, M. Rossi, D. Roggen, G. Troester, J. Doppler, C. Holzmann, A. Riener, A. Ferscha, and R. Chavarriaga. Recording a complex, multi modal activity data set for context recognition. In *Proceedings of the 1st Workshop on Context-Systems Design, Evaluation and Optimisation (CosDEO 2010)*. VDE Publishing House, Hannover, Germany, 2010.

[Mac01] P. MacWilliams, B. Prasad, M. Khare, and D. Sampath. Source synchronous interface between master and slave using a deskew latch. US Patent 6209072, 2001.

[mar] MARKER labeling tool. `http://people.ee.ethz.ch/~oamft/projects/marker/index.html`. [Online; accessed 13-Jan-2011].

[Min07] D. Minnen, T. Starner, M. Essa, and C. Isbell. Discovering characteristic actions from on-body sensor data. In *Proc. IEEE ISWC 2006*, pages 11–18. 2007. ISBN 1424405971. ISSN 1550-4816.

[Mit04] P. Mitra, C. A. Murthy, and S. K. Pal. A probabilistic active support vector learning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):413–418, 2004.

[Ngu04] H. T. Nguyen and A. Smeulders. Active learning using pre-clustering. In *Proceedings of the Twenty-First International Conference on Machine learning (ICML '04)*, page 79. Banff, AB, 2004.

[Nig00] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39:103–134, 2000.

[Ogr05] G. Ogris, T. Stiefmeier, H. Junker, P. Lukowicz, and G. Troster. Using Ultrasonic Hand Tracking to Augment Motion Analysis Based Recognition of Manipulative Gestures. In *Proc. IEEE ISWC 2005*, pages 152–159. 2005. ISBN 0769524192.

[Pan10] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[Pri00] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 32–43. ACM, New York, NY, USA, 2000. ISBN 1-58113-197-6.

[Puc88] M. Puckette. The patcher. In *The International Computer Music Conference*. 1988.

[Ran00] C. Randell and H. Muller. Context awareness by analysing accelerometer data. In *Wearable Computers, The Fourth International Symposium on*, pages 175 –176. 2000.

[Rat95] J. Ratsaby and S. S. Venkatesh. Learning from a mixture of labeled and unlabeled examples with parametric side information. In *Proceedings of the eighth annual conference on Computational learning theory*, COLT '95, pages 412–417. ACM, New York, NY, USA, 1995. ISBN 0-89791-723-5.

[Rog09] D. Roggen, K. Forster, A. Calatroni, T. Holleczek, Y. Fang, G. Troster, P. Lukowicz, G. Pirkl, D. Bannach, K. Kunze, A. Ferscha, C. Holzmann, A. Riener, R. Chavarriaga, and J. del R. Millan. Opportunity: Towards opportunistic activity and context recognition systems. In *World of Wireless, Mobile and Multimedia Networks Workshops, 2009. WoWMoM 2009. IEEE International Symposium on a*, pages 1 –6. 2009.

[Rog10] D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Förster, G. Tröster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, J. Doppler, C. Holzmann, M. Kurz, G. Holl, R. Chavarriaga, M. Creatura, and J. del R. Millán. Collecting complex activity data sets in highly rich networked sensor environments. In *Proceedings of the Seventh International Conference on Networked Sensing Systems (INSS), Kassel, Germany*. IEEE Computer Society Press, 2010.

[Rog13] D. Roggen, G. Tröster, P. Lukowicz, A. Ferscha, J. del R. Millán, and R. Chavarriaga. Opportunistic human activity and context recognition. *Computer*, 46(2):36 –45, 2013. ISSN 0018-9162.

[Sap08] T. S. Saponas, J. Lester, J. Froehlich, J. Fogarty, and J. Landay. ilearn on the iphone: Real-time human activity classification on commodity mobile phones. *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1043–1052, 2008.

[Sat01] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10 –17, 2001. ISSN 1070-9916.

[Sch94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85 –90. 1994.

[Sch99a] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde. *Advanced Interaction in Context*, volume 1707 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 1999.

[Sch99b] A. Schmidt, M. Beigl, and H. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[Set08] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*, pages 1070–1079. Honolulu, HI, 2008.

[Set09] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin, Department of Computer Science, 2009.

[She96] B. Sheehan, S. D. Fuller, M. E. Pique, and M. Yeager. AVS software for visualization in molecular microscopy. *Journal of structural biology*, 116(1):99–106, 1996.

[Sic98] A. Sicheneder, A. Bender, E. Fuchs, R. Mandl, and B. Sick. A framework for the graphical specification and execution of complex signal processing applications. 3:1757–1760, 1998.

[Sie03] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, and J. Shaffer. Sensay: a context-aware mobile phone. *Wearable Computers, 2003 Proc., Seventh IEEE International Symposium on*, pages 248–249, 2003.

[Siv04] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *Network, IEEE*, 18(4):45–50, 2004.

[Ste05] P. Steggles and S. Gschwind. The Ubisense smart space platform. 191:73–76, 2005.

[Sti06a] T. Stiefmeier, C. Lombriser, D. Roggen, H. Junker, G. Tröster, and G. Ogris. Event-based activity tracking in work environments. In *Proceedings of the 3rd International Forum on Applied Wearable Computing*. 2006.

[Sti06b] T. Stiefmeier, G. Ogris, H. Junker, P. Lukowicz, and G. Tröster. Combining motion sensors and ultrasonic hands tracking for continuous activity recognition in a maintenance scenario. *Wearable Computers, IEEE International Symposium*, 0:97–104, 2006. ISSN 1550-4816.

[Sti08a] T. Stiefmeier, D. Roggen, G. Ogris, P. Lukowicz, and G. Tröster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, 7:42–50, 2008. ISSN 1536-1268.

[Sti08b] M. Stikic, K. Van Laerhoven, and B. Schiele. Exploring semi-supervised and active learning for activity recognition. In *IEEE ISWC 2008.*, pages 81–88. 2008. ISSN 1550-4816.

[Sti11] M. Stikic, D. Larlus, S. Ebert, and B. Schiele. Weakly supervised recognition of daily life activities with wearable sensors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(12):2521–2537, 2011.

[Su05] W. Su and I. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, 2005.

[Sun05] B. Sundararaman, U. Buy, and A. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, 2005.

[Sze09] S. Szewcyzk, K. Dwan, B. Minor, B. Swedlove, and D. Cook. Annotating smart environment sensor data for activity learning. *Technology and Health Care*, 17(3):161–169, 2009.

[Tap06] E. Tapia, T. Choudhury, and M. Philipose. Building reliable activity models using hierarchical shrinkage and mined ontology. *Pervasive Computing*, pages 17–32, 2006.

[Ten09] M. Tenorth, J. Bandouch, and M. Beetz. The TUM kitchen data set of everyday manipulation activities for motion tracking and action recognition. In *IEEE Int. Workshop on Tracking Humans for the Evaluation of their Motion in Image Sequences (THEMIS). In conjunction with ICCV2009*. 2009.

[Ton02] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2002.

[VL00] K. Van Laerhoven and O. Cakmakci. What shall we teach our pants? In *Wearable Computers, The Fourth International Symposium on*, pages 77 –83. 2000.

[WA06] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. of The 7th USENIX Symposium on Operating Systems Design and Implementation*. 2006.

[Wan02] X. Wang, J. Dong, C. Chin, S. Hettiarachchi, and D. Zhang. Semantic space: An infrastructure for smart spaces. *Computing*, 1(2):67–74, 2002.

[War63] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of ASA*, 1963.

[War06] J. Ward, P. Lukowicz, G. Tröster, and T. 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.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

[Wei07] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle. Rapid prototyping for pervasive applications. *IEEE Perv Comput*, 6(2):76–84, 2007. ISSN 1536-1268.

[Wit05] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[Wya05] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically mined common sense. In *Proc. National Conference on Artificial Intelligence*, page 21. 2005.

[Yan09] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner. Demo abstract: mcrowd - a platform for mobile crowdsourcing. *SenSys '09*, 2009.

[Zhu08] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2008.

[Zim80] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

Curriculum Vitae

David Bannach received his diploma in computer science from ETH Zurich in 2003. Since then he has worked as research assistant in the fields of wearable and pervasive computing and human activity recognition at the Institute for Computer Systems and Networks at the University of Medical Informatics and Technology Hall in Tyrol (2004-2006), the Embedded Systems Lab at the University of Passau (2006-2012), and at the Wearable Computing Lab at ETH Zurich (2003, 2013, 2015). During that time he was involved in multiple EU-funded projects (WearIT@Work, MonAMI, Opportunity). In 2014 he co-founded the ETH spin-off company Bonsai Systems GmbH where he is member of the board of management and leads the software division.