RWTH AACHEN UNIVERSITY

A Mobile Sensor/Actuator Platform for Real-Time Mistake Detection and its Application to Snowboarding

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Adalbert Schanowski

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr.-Ing. Klaus Wehrle

Registration date: Nov 27th, 2007
Submission date: May 26th, 2008

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, May 26, 2008

Contents

List of Figures

List of Tables

Abstract

Learning new physical activities in different domains, such as sports, health care, or activities in everyday life, can prove to be difficult. Oftentimes, providing immediate feedback to the students is not possible. The reasons for that could be a spacial separation between the student and the instructor while performing specific activities such as snowboarding, but also the absence of an instructor. Therefore, sometimes most people learn new activities autonomously. To make progress quickly, most people often choose the easiest and fastest way for learning new movements. Based on this fact, even the basic movements might be learned and performed in a wrong way. In health care, wrong movements can lead to injuries, while athletes, who learned different techniques by themselves, can reach a point where further improvements become difficult. By using sensors and actuators mounted on the student's body, wrong movements might be detected and feedback could be provided immediately.

The goal of this thesis is the development of a supporting low-cost wearable sensor/actuator system that recognizes basic movements and gives feedback to the user automatically. Due to the fact that the system recognizes simple movements and sequences of them, the system can be used for different applications. We developed a robust and mobile hardware platform that allows to connect different sensors and actuators easily. In addition, we implemented a comprehensive Java ME library that allows rapid programing of mobile phone applications for this sensor /actuator system.
Guggenmos [2007] presented a first approach towards a wearable *Snowboard Assistant* that should detect common mistakes in snowboarding. We continued the work presented by Guggenmos and evaluated our system in the snowboarding domain. Furthermore, we developed the first version of a wearable *Snowboard Assistant* that recognizes two common beginner mistakes as well as analyses the descend of the snowboarder.
This work is not only to be regarded as a further step towards a wearable *Snowboard Assistant*, but also, based on the diversity of applications that can be built with this system, as an initial step towards a mobile toolkit for the development of wearable computing systems.

Überblick

Das Erlernen neuer physischer Aktivitäten in verschiedenen Bereichen, wie Sport, Medizin, oder Aktivitäten im Alltag, kann sich als schwierig erweisen. Oftmals ist ein sofortiges Feedback an den Schülers nicht möglich. Die Gründe dafür könnten eine räumliche Trennung zwischen dem Schüler und des Lehrers sein während einer bestimmten Aktivität, wie etwa beim Snowboarden, aber auch die Abwesenheit des Lehrers. Demzufolge erlernen die meisten Menschen neue Aktivitäten selbständig. Um schnell Fortschritte zu erzielen, wählen die meisten oft den einfachsten und schnellsten Weg beim Erlernen neuer Bewegungen. Aufgrund dieser Tatsache könnten bereits einfache Bewegungen falsch gelernt und ausgeführt werden. In der Medizin können falsche Bewegungen zu Verletzungen führen, während Sportler, die sich verschiedene Techniken selbst beigebracht haben oft an einen Punkt gelangen, wo es schwierig wird seine eigene Leistung noch weiter zu verbessern. Mit Hilfe von Sensoren und Aktuatoren, welche am Körper des Schülers befestigt sind, könnten falsche Bewegungen erkannt und sofort Feedback gegeben werden.

Das Ziel dieser Diplomarbeit ist die Entwicklung eines kostengünstigen tragbaren Sensoren/Aktuatoren Systems, welches einfache Bewegungen erkennen und Feedback an den Benutzer geben soll. Aufgrund der Tatsache, dass dieses System einfache Bewegungen oder Sequenzen dieser erkennt, kann es in verschiedenen Bereichen angewandt werden. Wir entwickelten hierfür eine robuste und mobile Hardwareplattform an der man leicht Sensoren und Aktuatoren anschließen kann. Zusätzlich implementierten wir eine umfangreiche Java ME Bibliothek, welche eine schelle Programmierung von Handyaplikationen für dieses Sensoren/Aktuatoren System erlaubt.
Guggenmos [2007] präsentierte einen ersten Ansatz in Richtung eines tragbaren *Snowboard Assistenten*, welcher bekannte Anfängerfehler beim Snowboarden erkennen soll. Wir setzten die Arbeit von Guggenmos fort und evaluierten unser System im Bereich des Snowboardens. Des Weiteren entwickelten wir die erste Version eines tragbaren *Snowboard Assistenten*, der zwei bekannte Anfängerfehler erkennen sowie auch die Abfahrt des Snowboarders analysieren kann.
Diese Arbeit ist nicht nur als ein weiterer Schritt in Richtung eines tragbaren *Snowboard Assistenten* anzusehen, sondern auch aufgrund der Vielfältigkeit der Verwendungsmögilchkeiten, die man mit dem System bauen kann, als ein erster Schritt in Richtung eines mobilen Werzeugsatzes für die Entwicklung von Wearable Computing Systemen.

Acknowledgements

First of all, I want to thank Prof. Dr. Jan Borchers for giving me the opportunity to write this thesis on the Media Computing Group that offers an enjoyable working atmosphere.

Thanks to my supervisor Daniel Spelmezan for his permanent support, feedback, and for proof-reading my thesis. His ideas helped me to cope with different situations during the last months.

I also want to thank Prof. Dr.-Ing. Klaus Wehrle for having agreed to be my second examiner.

Furthermore, I want to thank all participants of our user study for their time and patience during the whole evaluation phase on the cold indoor slope.

I also want to thank my father's employer *GRAFOTEAM GmbH*, where I could borrow tools to build the hardware at my parents' home.

Thanks to Daniel Thomas and Jan Berg for proof-reading my thesis and for their feedback on several issues in such a short time.

Finally, I want to thank my father Andreas Schanowski for teaching me soldering basics and my mother Eva Schanowski for supporting and motivating me during the last months.

Last but not least, I want to thank my girlfriend Agnieszka Kolek for her support, help and patience during the last months.

Conventions

Throughout this thesis we use the following conventions.

The plural ”we” will be used throughout this thesis instead of the singular ”I”, even when referring to work that was primarily done by the author.

Unidentified third persons are always described in male form. This is only done for purposes of readability.

Source code and implementation symbols are written in typewriter-style text:
`public class myClass(){...}`

Links to project sites or webpages of mentioned products and applications are shown in a footnote at the bottom of the page.

The whole thesis is written in American English.

Chapter 1

Introduction

"Learning is like rowing upstream; not to advance is to drop back."

—Chinese Proverb

Learning with an instructor

The best way to learn new physical activities such as new sports is to have an instructor who step by step demonstrates the basic movements . While practicing these basics, the instructor can immediately give feedback on the students' performance. He can interrupt wrong movements, give constructive criticism and help the student to perform exercises correctly. No feedback or even delayed feedback might slow down the students' learning pace, increase frustration, and extend the learning time even for simple exercises. Although students might not learn basics properly, they could achieve positive short-time successes. However, in the long run with such self-teaching skills students could reach a point where further improvements become difficult.

Close student-instructor collaboration

In sports like tennis or golf the instructor can observe and analyze each movement of the student. He can immediately talk to the student and give advices on how to perform an exercise correctly. Moreover, the instructor can guide the student's hand or his whole body to demonstrate a specific movement. While receiving this haptic information and feedback, the student can focus on

his performance. Figure 1.1 shows Cole Pickavance during a golf lesson with his father. Having regularly learned golf with an instructor, Cole is now a top-ranked junior player.

Figure 1.1: The instructor guide the student's hand to demonstrate a movement.

Some sports disallow close collaboration

In sports like snowboarding, skiing, or surfing a close student-instructor collaboration is not possible. The student can only receive feedback or advices before or after he performs an exercise. While practicing, the student depends on himself and must learn the right movement only by remembering the instructions.

The oblivion of rehabilitation patients

Another application domain where real-time feedback is very useful is health care. Patients after an accident should avoid specific movements. For instance, patients with sprained or fractured arms should take care of not stressing their arm too much. Usually, a medic advises the patient and informs him about wrong movements, which the patient should avoid. At the beginning, the patient follows the instruction and takes care of his arm. While feeling the pain in his arm, he automatically avoids wrong movements. However, the pain in his arm continuously

decreases until the pain is completely gone. This phase is critical, because it seduces the patient to be careless. Although he is not completely healed, this situation encourages him to perform movements that he should not perform.

Wrong movements in everyday life

Aside from sports and health care, real-time feedback is useful in everyday life. Most of the accidents occur in our everyday life [1]. Wrong movements while lifting heavy things can result in severe injuries. A wrong sitting posture can cause elbow and back pain. Receiving no feedback on how to perform correctly daily activities encourages everyone to choose the simplest way to cope with them, which is sometimes the wrong way. For example, someone who rarely lifts heavy things would bend his back to pick up a box. In contrast to that, a mover would straighten his back and bend his knees in order to pick up the box.

Importance of real-time feedback

All in all, real-time feedback while learning new physical activities is very important. People would benefit from having an instructor who guides him through the variety of physical activities. However, only few people can afford a personal trainer in sports or a non-stop medical supervisor during the healing process for a longer time period. A low priced system, which can partially replace an instructor would be beneficial for all application areas.

Basic movements as common denominator

Today, several wearable computing projects focus on this problem of missing feedback and develop systems that support the users while learning different physical activities. One problem is that most of the systems are tailored only to one application domain and cannot be used directly in any other area. Most physical activities are too different and too complex to use only one supporting system for both applications.
One solution is to divide complex movements into smaller basic ones. Thus, a software that can detect each basic movement and any combination of them can be adapted to most situations and can be used among the most application domains. For instance, correctly lifting a box up from the ground consists of elementary movements of bending and stretching both knees while straightening the back at

[1] http://www.baua.de/

the same time. The program which can detect each of these elementary movements and the combination of them could provide feedback if the user bended his back while lifting something up.

Snowboard Assistant

Due to the problem of missing feedback in sports, the Media Computing Group, RWTH Aachen[1] started the wearable computing project Snowboard Assistant[2] that focuses on snowboard beginners. By using mounted sensors and actuators on the students body, the systems is intended to detect wrong movements such as common beginner mistakes, give real-time feedback, and support students during their learning process.

SensAct box

As a part of the *Snowboard Assistant* project, we developed a sensor/actuator platform (*SensAct box*) for mistake detection in different application domains. By using the idea of dividing complex movements into elementary ones, the *SensAct box* can be used for different physical activities, detect wrong movements and provide feedback to the user.

1.1 Goals and Requirements

The main goal of this work is to build a system that provides feedback to the user based on prior defined gestures and postures in different application domains. The system consists of a hardware box, where users can easily attach sensors and actuators, and different detection algorithms on a mobile phone, which is an ideal platform for such applications (Abowd et al. [2005]).

Evaluation in the snowboarding domain

As a subsequent work of Guggenmos [2007], who initiated the *Snowboard Assistant* project, we want to evaluate the *SensAct box* in the snowboarding domain. The system should detect common mistakes of snowboard beginners and provide, based on the detected mistake, feedback via actuators.

[1] http://hci.rwth-aachen.de
[2] http://hci.rwth-aachen.de/snowboard

Software requirements

Besides the application to snowboarding, the software should fulfill the following requirements in order to be usable in other domains:

- The software should detect most of the basic movements such as limb bending, weight distribution, and simple activity recognition.
- It should be easy to combine these basic movements and build a posture model.
- The software should allow the users to easily connect the posture model to specific actuators.
- It should be possible to extend the software with new detection algorithms and feedback patterns.

Hardware requirements

Based on the diversity of the application domains, the hardware should be constructed in such a way that it can be applied to different physical activities. For instance, sport athletes need a robust system, which can be used outdoors. Patients prefer small and mobile systems, which do not disturb them in everyday life.

The hardware should comply the following requirements :

- The system must be mobile in order to do not disturb the user during the physical activity.
- The hardware must be robust and applicable to a variety of physical activities.
- It should be easy to connect different sensors and actuators at runtime.

1.2 Structure of the Thesis

This thesis is structured as follows:

Chapter 2—"Related work" provides an overview about different domain specific wearable computing systems in health care, sports, or daily life and domain independent wearable computing toolkits. Finally, we compare our work with the related work and focus on the differences between them.

Chapter 3—"A Mobile Sensor/Actuator Platform" describes the design of the sensor/actuator hardware box and a further improved version of the box. This chapter also includes several detection algorithm implementations, that allow rapidly prototyping of mobile wearable systems.

Chapter 4—"Evaluation in the Snowboarding domain" discusses the software algorithms in view of the application domain of snowboarding.

Chapter 5—"Final Implementation: The First Snowboard Assistant" provides an overview of the first mistake detection software that we implemented during this work by using the developed algorithms. Here, we also focus on challenges and problems, that we identified during the implementation of mobile phone applications.

Chapter 6—"Summary and Future Work" gives an overview about this work, summarizes the identified software and hardware problems, and proposes solutions, which should be realized in the near future.

Appendix A—"Software Library Documentation" contains an sample run of the Snowboard Assistant application, which we have implemented in this work.

Appendix B—"First Snowboard Assistant: Sample run" contains an instruction of a mobile phone application, which supports the designer by displaying sensor values as graphs in real-time.

Appendix C—"Sensor Monitor: Sample run" contains an instruction of a mobile phone application, which allows manually triggering actuators that are connected to the hardware box.

Appendix D—"Motor Control: Sample run" contains the mathematical formulation of the smoothing filters that we used for our algorithms.

Chapter 2

Related work

"Research is what I'm doing when I don't know what I'm doing."

—Wernher von Braun

Domain specific wearable computing systems

Application areas for wearable computing range from e-Health, to sports, to everyday life, whereas each domain can strongly benefits from wearable computing systems. Health personnel can use wearable systems for monitor their patients all the time, athletes can observe and analyze their performance, and individuals can make their everyday life easier. However, most of the wearable computing systems are tailored to the application domain, which makes these systems unfeasible to other application areas.

We first intend to present some projects in each application domain and show how helpful these wearable systems can be. Additionally, we point out the close relationship between wearable systems and the corresponding application domain before we introduce current domain independent wearable computing systems. Finally, we compare all the wearable system with our *SensAct box*.

2.1 Health care

Wearable systems in health care

Wearable computing can be applied to a variety of medical applications, which vary from observing the patient and collecting specific medical data, to supporting disabled people with various tasks. Such wearable systems not only facilitate the work of medicals but also the convalescence of patients who do not need to stay in hospital all the time. One part of wearable systems in health care deals with monitoring the condition of patients. Since health professionals cannot observe their patients all day long, wearable computing allows doctors to get an overview of the patients state of health over a long time period.

2.1.1 HealthGear

Detection of sleep apnea events

Oliver and Flores-Mangas [2006] have designed a system to monitor, visualize and analyze physiological signals in order to detect sleep apnea events[1].Figure 2.1 shows a system overview of HealthGear.

The system provides Sensors that measure the user's blood oxygen level, pulse rate, and plethysmographic[2] signals, while the user is asleep. These data is sent via Bluetooth to a smartphone, which analyzes the data and displays a diagnosis on the mobile phone's screen.

Storing data for further studies

Additionally, the application stores the data on the mobile phone for further studies. The system was able to identify and to classify every kind of breathing interruptions during the night.

[1] Breathing interruption while sleeping.

[2] Plethysmography is a set of noninvasive techniques for measuring volume changes of different body parts.

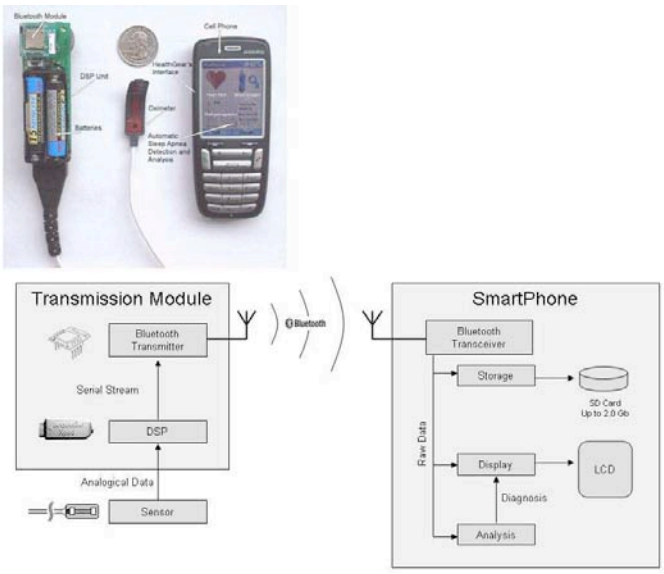

Figure 2.1: System architecture of HealthGear developed by Oliver and Flores-Mangas [2006]

2.1.2 Wearable System for Visually Impaired People

Improving mobility of visually impaired people

Besides monitoring systems, wearable computing can also be used to support elderly or disabled people. Cardin et al. [2006] proposed an obstacle detection system to improve the mobility of visual impaired people. The system reacts to obstacles by providing tactile feedback in such a way that the user can approximately determine the position of the obstacle. The hardware consists of four sonar sensors fixed to the shoulders and eight vibration coin motors around the chest. The system detects objects up to three meters at an angle of 60°. Figure 2.2 shows the sensing range of the system.

Successful evaluation

To evaluate the system, Cardin et al. instruct users to walk across a corridor full with dynamic obstacles such as walking persons or opening and closing doors. They observed that users with blindfolded eyes need only slightly more time to pass the corridor than user without any visual limitations.

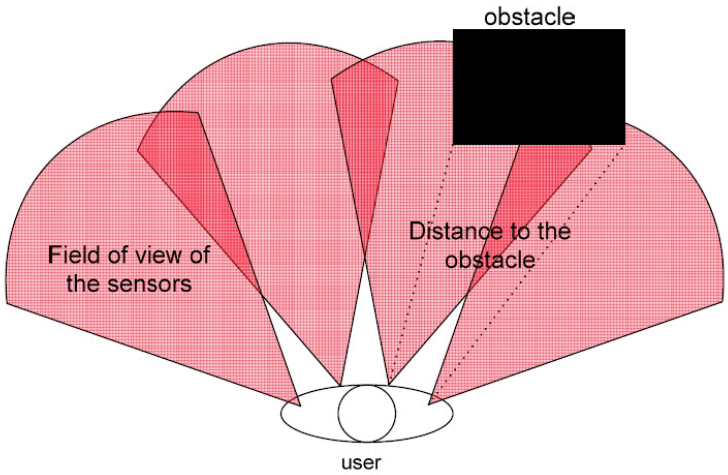

Figure 2.2: Sensing map of the 4 sonar sensors (Cardin et al. [2006]).

2.2 Sports

Fast and complex sport movements

Sports can be considered as an application domain that contains lots of fast and complex movements. Thus, wearable computing systems in sports have to be mobile, robust, and very accurate.

2.2.1 Recognizing Tai Chi

Recognizing Tai Chi movements

Kunze et al. [2006] described a first approach how to recognize Tai Chi movements with body mounted sensors. For analyze these movements, Kunze et al. used the XBus Master System by XSens[1] , which will be discussed in this chapter later on.

After interviews with Tai Chi experts, Kunze et al. placed the MT9 nodes on the right and left arm, on the right and left lower leg, on the right and left knee, on the neck, and on the rear hip of the monitored subject. These nodes consists of a 3-axis accelerometer, a 3-axis gyroscope, and a 2-axis magnetic field sensor

[1] http://www.xsens.com

Distinguish between Tai Chi exercises and user skills

To evaluate their system, Kunze et al. recorded sensor data from two Tai Chi experts and two Tai Chi amateurs, who performed three different Tai Chi movements. After having trained a K-nearst-neighbour algorithm with features such as the 75% percentile and the frequency range power of the neck mounted accelerometer x-axis, Kunze et al. were able to classify the subjects' expertise with an accuracy of 76% and to distinguish between two Tai Chi movements with an accuracy of 85%.

2.2.2 iTrainer^{TM}Golf

iTrainer helps improving golf swing

Besides scientific approaches to support athletics in different disciplines, Insight Ltd.[2] proposed a golf swing training system, which helps beginners as well as professionals to improve their game. The system analyses and improves the golfer's swing.

iTrainer hardware setup

The system captures sensor data from a "Sports Motion Capture Device" that is mounted on the golf club shaft. This device consists of gyroscopes, magnetometers and accelerometers. The hardware can communicate with a Bluetooth head set, which provides spoken and audio tone instructions in real-time. Additionally, the user can analyze his golf training session on a PC.

Currently, Insight Ltd. plans to use up to eight additional sensors, which should be mounted on the users' body to measure fine-grained changes of golf swinging movements. The system is under development and will enter production in 2008. In addition, Insight Ltd. plans further training systems for other sports such as Tennis, Squash, Soccer, etc.

[2] http://www.insight-sports.com

2.3 Everyday Life

Although the amount of every day activities is equal to other application domains, there are not so many projects in this domain as compared to health care or sports. Nevertheless, there are some promising approaches that support the user and help him to avoid injuries in the every day life.

2.3.1 Monitoring of Seated Posture

Detecting wrong sitting postures

Dunne et al. [2007] described a system that corrects sitting postures of computer users. Since wrong sitting postures might lead to back injuries, the system continuously monitors the sitting posture of the computer user.

Optical Sensor Shirt

The hardware consists of an optical sensor shirt (Dunne et al. [2006]) with an integrated plastic optical fiber and a microcontroller, which collects and sends the data via Bluetooth to a PC. Figure 2.3 shows the sensor shirt with the optical bend sensors mounted on the back of the user. When the users siting postures changes into a wrong one, the software on the computer alerts the user by changing the color of an icon in the tray from green to yellow and to red.

Most wearable systems are too domain specific

Although the most wearable systems used similar hardware and detection algorithm software, all theses systems are tailored to one application domain and cannot be applied directly to other application areas. In addition, these wearable systems were basically designed by professionals and focused more on the engineering phase. In the next section we will present domain independent systems that focus primarily on the design of wearable computing systems.

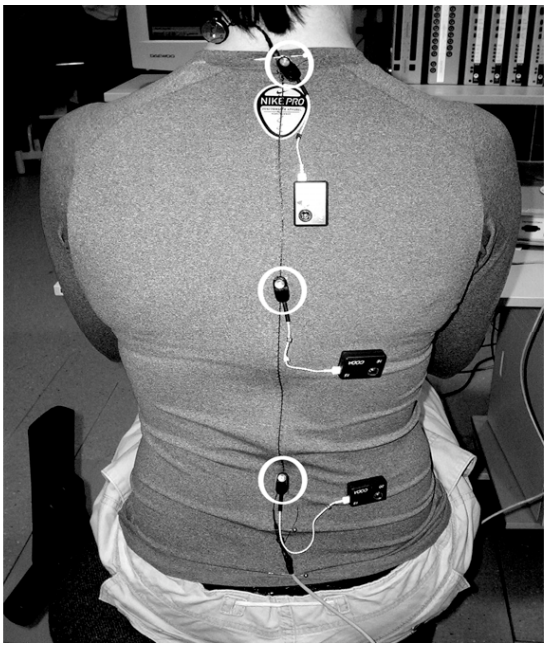

Figure 2.3: Optical sensor shirt to detect user's sitting posture((Dunne et al. [2006])).

2.4 Domain Independent Wearable System Toolkits

Besides the domain specific wearable computing systems, there are also projects that focus on the creation of domain independent systems. These can be used as a toolkit to develop such wearable computing systems that we presented above.

2.4.1 Construction Kit for Electronic Textiles

Toolkit to introduce novices in e-textile

Buechley [2006] presented a toolkit that ease the e-textile usage and introduce novices to electronics and wearable computing design. The hardware setup consists of a microcontroller, light sensors, temperature sensors and pressure sensors, LEDs, vibrating motors, and a IR transreceiver, which can be connected by the user with a conductive thread. Figure 2.4 shows a subset of the construction

kit. Additionally, the toolkit offers software libraries in C to assist the user with programming the microcontroller for controlling the IO pins, reading data, or communicating via IR with PDAs or another IR transreceiver.

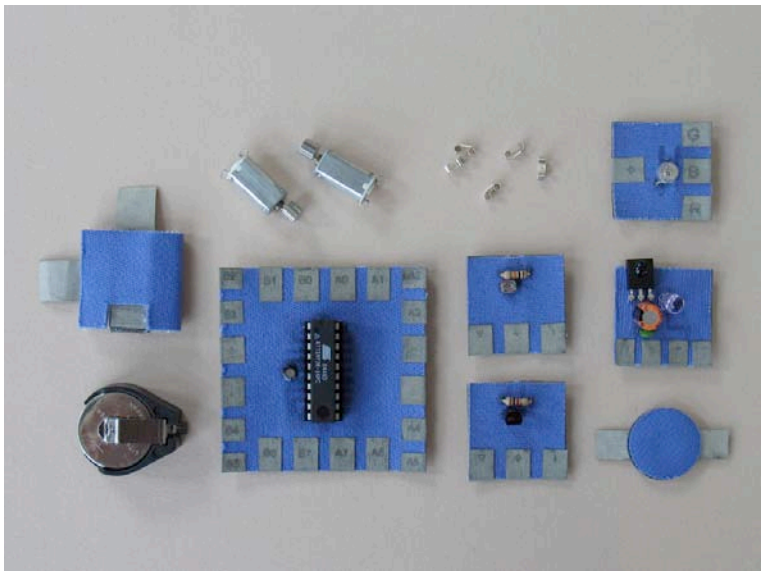

Figure 2.4: Hardware construction kit to create wearable computing systems presented by Buechley [2006].

Most users were able to complete working design

To evaluate the toolkit, Buechley allowed novices to create their own wearable systems and observed how well the subjects coped with this toolkit. About 87% of the test subjects were able to complete working wearable systems, such as communicating shirts that can communicate with other shirts via IR, a temperature sensing hat, which changes the color of the pompom on the top depending on the temperature, or wearable LEDs, which show different animations on clothes.

One drawback of this construction kit is that the individual devices are not protected or covered. Thus, the prototypes are not robust and can be damaged easily.

2.4.2 Rapid Prototyping of Activity Recognition Applications

Visual programming allows rapid prototyping

Bannach et al. [2008] presented the Context Recognition Network (CRN) Toolbox, which allows a fast construction of multi-modal[1] context recognition systems by simplifying the process of creating activity recognition systems. The system consists of a set of ready-to-use algorithms, which enable the user to construct complex applications rapidly. The CRN box offers a visual programming editor for realizing activity recognition systems simply by connecting and configuring a set of tasks and algorithms. Figure 2.5 shows the concept of the CRN Toolbox.

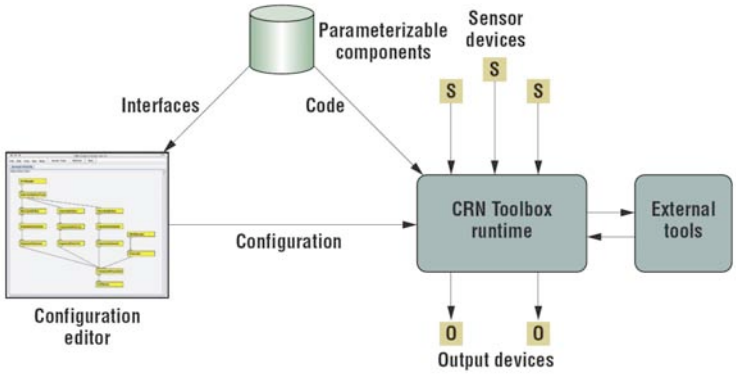

Figure 2.5: Concept of the Context Recognition Network Toolbox developed by Bannach et al.

After short training, the system is ready to use

To create an activity recognition application, the user just needs to specify the input sensor and record the training data of one activity. After that, the user can include a classifier and program it with the recorded training data. Finally, the user determines an output task such as an image as visual feedback, and the toolbox is ready for recognizing the trained movements.

The CRN Toolbox runs on different platforms, such as Linux, Mac OS X, Cygwin, or QBIC[3] - Belt Integrated Computer.

[1] Sensor data from different types of sensors.
[3] http://www.qbic.ethz.ch

2.4.3 The Mobile Sensing Platform

Collecting sensor data from a single body location

Choudhury et al. [2008] described a mobile sensing platform (MSP), which collects sensor data from different sensor types from a single body location. This platform is a small device, which consists of a wireless IMote 2[4] sensor node platform , a microphone, a visible light phototransistor, an accelerometer, a digital barometer, a temperature sensor, a humidity sensor, a digital compass, and a digital IR and visible IR light sensor(Figure 2.6).

Figure 2.6: Mobile Sensing Platform developed by Choudhury et al.

Training data to build a motion model

This node can communicate via Bluetooth and runs on batteries for several hours. Before starting to work, a training process has to be done by collecting sensor data. This trainings data is used to extract parameter values and build a motion model, which later is implemented on the hardware platform.

Distinguishing between five types of activities

To evaluate the system, Choudhury et al. implemented a mobile phone application called UbiFit Garden. This application encourages users to be physically active by

[4] http://www.xbow.com

displaying flowers on the mobile phone screen that started to bloom when the user performs physical activities. The software distinguishes with an accuracy of 93,8% between five types of activities such as walking, running, cycling, using an elliptical trainer, and using a stair machine.

One drawback of the system is that it provides no actuators and only a set of sensors that cannot be extended with further sensors.

2.4.4 A Portable Kit for Naturalistic Data Collection

Collecting multi-modal sensor data

Tapia et al. [2006] presented a wireless sensing kit for pervasive computing for collecting multi-modal sensor data in natural surroundings like at home without disturbing the user's everyday life. Figure 2.7 shows a subset of the environmental and wearable sensors. The

Figure 2.7: A subset of sensors of the wireless sensing kit provided by Tapia et al. [2006].

Environmental sensors and wearable sensors

hardware consists of environmental sensor types such as light sensors, temperature sensors, proximity sensors, and movement sensors. In addition the system provides wearable sensors like accelerometers, RFID readers, heart rate sensors, a ultra-violet radiation exposure, and location beacons.

The sensor data can be stored either on a PC or a mobile device in real-time for further analysis. This system has been used successfully in a couple of application areas such as medicine, where researchers are using this system to study the relationship between the user's heart rate and certain activities, e.g. television at home.

2.4.5 EduWear

Wearable computing system toolkit for children

EduWear[5] is a project that allows developing simple wearable systems for beginners and even children. The hardware setup consists of the Arduino Mini[6] , LEDs and Piezo speakers as actuators, and a set of different sensors, such as stretch sensors. Programming the Arduino Mini is realized through the Amici software, a visual programming editor that supports beginners and children to develop Arduino programs easily by connecting predefined program blocks. After developing a small application, the software translate program into a C-like language for the Arduino. Figure 2.8 shows the visual programming editor that can be used by children.

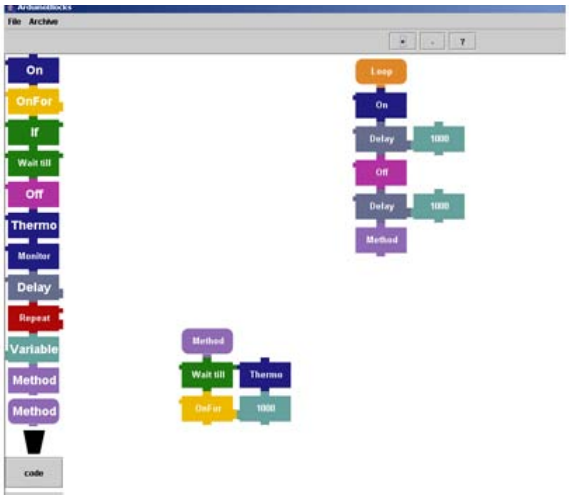

Figure 2.8: Amici: The Visual Programming Editor for beginners and children

[5] http://dimeb.informatik.uni-bremen.de/eduwear
[6] http://www.arduino.cc/en/Main/ArduinoBoardMini

Workshops for children

The EduWear project regularly organizes workshops for children, where children successfully devise and develop simple wearable systems. Since the system is for beginners and children, the toolkit uses only simple techniques and simple algorithms for developing wearable systems.

2.4.6 Exemplar: Authoring Sensor Based Interactions

Programming by demonstration

Hartmann et al. [2007] presented the program Exemplar[7] , which offers new techniques for developing sensor-based interactions through programming by demonstration. Exemplar is a plug-in for Eclipse[8] , which is a free development platform for different operating systems. Figure 2.9 shows the graphical user interface of Exemplar. To develop a sensor-based application, the designers only have to perform an action with some sensors, mark the resulting sensor data curve as a training example, and test the generated behavior. Then, the program recognizes trained movements if the actual sensor data matches the selected sensor data. Therefore the system uses different features such as thresholds, distance matrices, and dynamic time warping.

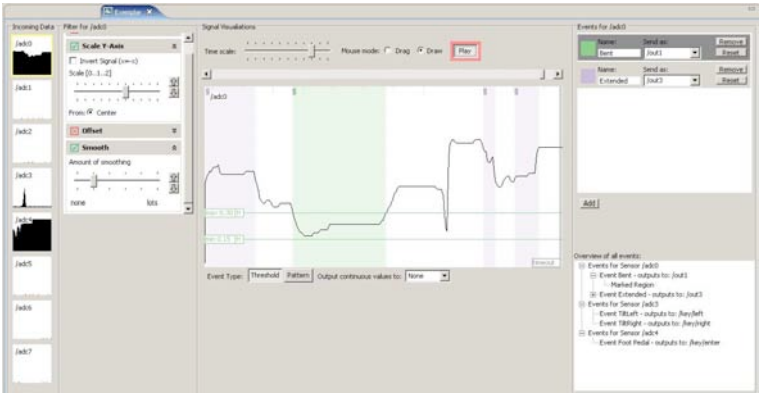

Figure 2.9: The Graphical User Interface of Exemplar (Hartmann et al. [2007])

[7] http://hci.stanford.edu/exemplar
[8] http://www.eclipse.org

All users completed all tasks

To evaluate this program, Hartmann et al. instructed 12 participants to build several sensor-based systems. These systems range from displaying "Hello World", while using a pressure sensor, to developing a motion-based controller for computer games. All participants completed their tasks successfully. Furthermore, they observed that most of the time was spent on design thinking rather then on implementation.

Exemplar provides no additional hardware

Exemplar is only the program and Hartmann et al. provided no additional hardware. However, the program supports several platforms such as the Arduino[1] that we also used for our work.

2.4.7 TactaBoard

Controlling up to 16 actuators

Lindeman and Cutler [2003] developed the *TactaBoard*, a small and robust case (19cm x 11cm x 5.8 cm) that allows to connect and control up to 16 actuators such as vibrotactile motors. Figure 2.10 shows the *TactaBoard* with attached actuators.

API for controlling actuators

The *TactaBoard* uses the MaxStream[9] 9XStream-DEV^{TM}wireless development kit for wireless communication between a host computer and the *TactaBoard*. Additionally, Lindeman and Cutler implemented an application programing interface (API) that allows to control all actuators with different intensity levels by using pulse width modulation (Barr [2001]). The *TactaBoard* offers the possibility to store the different output levels on the system.

Touch feedback in simulations

Basically the *TactaBoard* is developed to provide touch feedback in simulation environments such as virtual reality simulations. Hence, the system offers no connectors for sensors and can only be used as an extension for existing systems.

[1] www.arduino.cc
[9] http://www.maxstream.net

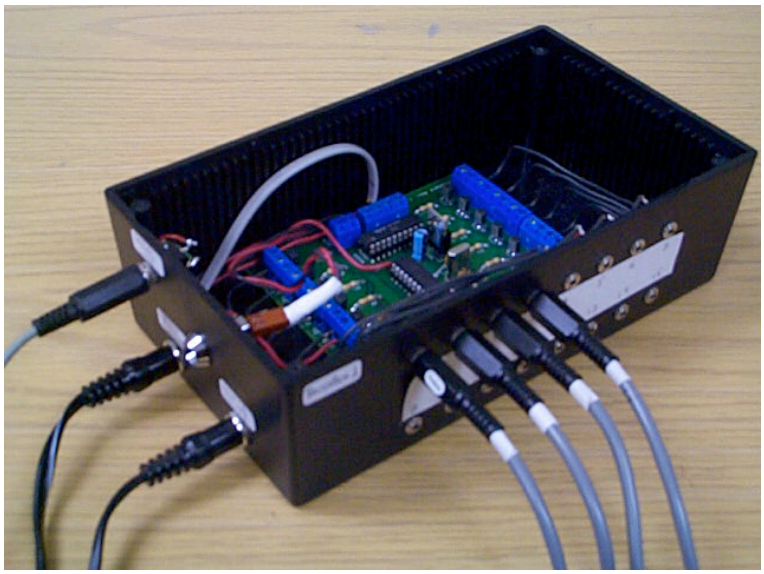

Figure 2.10: The interior of the *TactaBoard* developed by Lindeman and Cutler [2003].

2.4.8 XSens Xbus Kit

Hardware kit for human motion tracking

XSens[10] offers a wearable hardware kit for tracking human motion similar to system proposed by Kunze et al.. This kit consists of the Xbus Master box and eight smaller MT9 sensor nodes. Each node includes a 3-axis accelerometer, a 3-axis gyroscope, and a 2-axis magnetometer. The master box has a Bluetooth transceiver and can be connected to a PC or a mobile device in order to stream and record sensor data for an application running on these devices. Additionally, users can connect up to ten MT9 sensor nodes to only one master box. Figure 2.11 shows how the Xbus Kit can be mounted and used on the users' body. The system offers no possibility to attach actuators. Therefore the system can be only used for data logging.

[10] http://www.xsens.com

Figure 2.11: XBus Master System, which was used for recognizing Tai Chi movements.

2.5 Discussion

Domain independent wearable system toolkits

Wearable computing systems can be used in a diversity of application domains. This ranges from sport to health care and to everyday life. Although these domains differ significantly, most of the wearable computing projects use the same sensor types, similar hardware platforms, and similar algorithms to detect and recognize specific motions.
Based on this fact, many domain independent wearable toolkits exist, which reduce the development time and ease the development of such domain specific systems.

2.5.1 Toolkits preferences

Important aspects of toolkits

Usability is an important aspect of existing toolkits. Therefore, toolkits should support the design and implementation phase by providing options for rapidly developing systems as well as allowing quick modifications on the created prototypes. According to Myers et al. [2000], who evaluated user interface toolkits, wearable toolkits can be characterized by these following aspects:

1. **Development pace**(Threshold) is the effort and the time for developing prototypes including a prior learning phase.
2. **Prototype quality**(Ceiling) describes quality aspects of finished prototypes such as mobility or robustness. In other words, ceiling describes how much can be build by using the toolkit.
3. **Extensibility** is an aspect that defines whether it is possible to extend the toolkit with new sensors, actuators, and new algorithms.
4. **Modifiability** focuses on the effort for slight changes on an existing prototype.
5. **Mode of Operation** specifies whether the system is used for data logging or for advising the user.
6. **Costs** describes the price that the user must pay for buying or building the toolkit.

Development pace

Development time of new prototypes

The time to build functional prototypes successfully is an important preference of these toolkits. This depends on different aspects such as the usability of the hardware kit, the simplicity of the software development applications. Most of the systems fulfill these requirements and allow to decrease the time required to build new prototypes.

The *construction kit for e-textiles* (2.4.1) presented by Buechley [2006] offers a complete set of different hardware devices and predefined libraries in C to assist the user during the developing phase.

The *CRN Toolbox* (2.4.2) allows rapid prototyping through visual programming. In addition, the system offer a couple of algorithms that can be chosen by the user can choose for activity recognition.

EduWear (2.4.5) simplifies the prototype development by providing a visual programming GUI, which allows programming of applications that are comprehensible for children.

Exemplar (2.4.6) allows rapid prototyping by using programming through demonstration. Thus, the learning pace decreases because no programming skills are needed.

Prototype quality

Rapidly constructed prototypes often have a low quality

Basically, prototypes are used to proof new concepts quickly(Schrage [1996]). Due to this fact, most prototypes are constructed rapidly and have a low quality. However, the following toolkits can be used to build high quality prototypes that are robust and mobile enough to be tested in the field.

To build high quality prototypes, the *XSens Xbus Kit* (2.4.8) provides a set of robust and mobile hardware, which can be used for different applications in the field.

The *Mobile Sensing Box* (2.4.3) is embedded in a hard case and offers similar quality characteristics as the *XSens Xbus Kit*.

Extensibility

Extending toolkits with new sensors, actuators, and algorithms

One important aspect is the extensibility of such toolkits. In other words, extensibility describes the possibility of extending the set of sensors or actuators with new types of sensors or actuators.

The *construction kit for e-textiles* (2.4.1) is one toolkit that can be easily extended with new types of sensors and actuators. To add new sensors, only a couple soldering operations are required.

EduWear (2.4.5) has the same characteristics and allows to extend the existing set of sensors and actuators.

Modifiability

Modification on existing prototypes

The development process consists of multiple Design-Implementation-Analyze (DIA) cycles, where several prototype versions are built and tested. Therefore, toolkits that allow rapid modifications on prototypes instead of building new prototypes at each implementation step, decreases the developing time of new products. In addition, toolkits that can be extended with new types of sensors, actuators, and new algorithms increase the development pace of new products.

For modifying *XSens Xbus Kit* (2.4.8) prototypes, the user can easily displace the sensors on the body and simply add or remove sensors to an existing prototype. Based on the user's needs, the prototype can be adapted seamlessly to different requirements.

The *Portable Kit for Naturalistic Data Collection* (2.4.4) offers similar aspects to users. Since the system was developed for sensor data logging, users can rapidly change the sensor configuration by adding, removing, and displacement of sensors according to the user's needs.

Mode of Operation

Operation mode of the prototypes

We distinguish between two operation modes of the prototypes.

1. Data logging systems, which only can be used to collect sensor data for further analysis.
2. Advising systems that additionally provide feedback in order to teach, warn, or support the user.

The *XSens Xbus Kit* (2.4.8) and the *Portable Kit for Naturalistic Data Collection* (2.4.4) are basically used to monitor the user and collect sensor data for further processing on the PC.

The *CRN Toolbox* (2.4.2) and *Mobile Sensing Platform*

(2.4.3) are also used to monitor sensor data, but they offer some visual feedback by using the display on the PC or a mobile device. This type of feedback is primarily intended to provide information to the user and not directly for teaching or supporting him.

The *TactaBoard* (2.4.7) and *Exemplar* (2.4.6) are not complete toolkits. The *TactaBoard* can be used in existing systems in order to extend their functionality. *Exemplar* is only a Eclipse plug-in and offers no hardware, but supports some hardware platforms such as the Arduino. For that reason, we cannot classify the mode of operation for these systems.

Costs

Costs for realizing a prototype

Costs are an important aspect of toolkits. Prototypes are low-quality pre-versions and should not be more expensive than the final product.

EduWear (2.4.5) is built around Arduino Mini and a couple of low-costs sensors, which makes the system affordable. By contrast the *XSens Xbus Kit* (2.4.8) is a very expensive toolkit that costs thousand of dollars.

The *construction kit for e-textiles* (2.4.1) and the *Portable Kit for Naturalistic Data Collection* (2.4.4) are also low-cost toolkits.

2.5.2 Comparison

Comparison of all prototypes

Most of these toolkits can be used for rapid development of wearable computing prototypes for different application domains. However, each of these toolkits has small drawbacks. For instance, toolkits which allow rapid modification on existing prototypes are expensive and allow only develop data logging systems. Toolkits for developing advising systems do not support quick modifications on existing prototypes. With our *SensAct* box we want to close this gap and provide a low-cost toolkit that fulfill the most of these requirements.

Table 2.1 summarizes and compares the characteristics of related work with the *SensAct* box.

	Developing Pace	Prototype Quality	Extensibility	Modifiability	Operation Mode	Costs
e-Textiles (2.4.1)	(✓)	(✓)	✓	✓	advising	cheap
CRN Toolbox (2.4.2)	✓		(✓)	(✓)	advising	cheap
MSP (2.4.3)		✓			monitoring	adequate
Nat. Data Col. (2.4.4)	✓	✓	(✓)	✓	monitoring	cheap
EduWear (2.4.5)	✓	✓	✓		advising	cheap
Exemplar (2.4.6)	✓	✓			-	free
TactaBoard (2.4.7)	✓	(✓)	✓	✓	-	cheap
XSens (2.4.8)	✓	✓		✓	monitoring	expensive
SensAct box	✓	✓	✓	✓	advising	cheap

Table 2.1: Comparison of related toolkits. (✓ = fulfills, (✓) = partially correct)

Chapter 3

A Mobile Sensor/Actuator Platform

"Everything should be made as simple as possible, but not simpler."

—Albert Einstein

Requirements for our system

Before we started to develop our prototype, we thought about different requirements that should be fulfilled by the system. On the one side, the system should be applicable in different domains. In other words, the hardware should be small and robust in order to do not disturb users during their activities. Additionally, it should be easy to adapt the hardware to meet the user's needs. On the other side, the software of the system should run on a mobile phone. Since we have planed to have an open-source system, we want to switch to Android[1] in the near future.

Java ME for mobile application programming

On this account, we chose Java ME as the programming language for our software, because the transition to Android is much easier from Java ME than from any other programming language such as Symbian C or Python. We followed the DIA cycle and made two iterations in the hardware and software design.

[1] http://code.google.com/android

3.1 System Design Architecture

System architecture

As a subsequent work of Guggenmos [2007], we used the same system architecture as a starting point for our work. The system consists of the *SensAct box*, that allows connecting sensors and actuators at runtime and a mobile phone, which runs the software and the algorithms to detect user gestures and postures.

Two-way Bluetooth communication

A two-way Bluetooth Serial Port Profile communication connects both devices. The *SensAct box* sends the raw or preprocessed sensor values to the mobile phone, whereas the phone responds with several commands, such as motor control instructions or configuration instructions.

Connect up to seven boxes to one phone

By using Bluetooth communication, our system can be extended with more than only one *SensAct box*. Due to the Bluetooth characteristics, we can connect up to 7 *SensAct boxes* to one mobile phone and increase the amount of sensors and actuators. First experiments showed that data streaming and data logging from multiple *SensAct boxes* to one mobile phone are possible. Figure 3.1 shows the system architecture of our hardware prototype.

3.2 Hardware Setup

Hardware setup of our system

The *SensAct box* consists of an open-source ArduinoBT [1] board, which can be programed in a C-like language. This board offers eight analog input pins and 14 digital input/output pins of which we use the six pulse width modulation (PWM) pins to control the actuators and additionally two digital pins for experimental usage. Two of the eight analog input pins cannot be used directly. Therefore, some soldering operation are needed before we could attached additional two sensors. The ArduinoBT has an ATMEGA168 microprocessor running at 16 MHz, 1 KB RAM, and 14 KB flash RAM for the program code.

[1] http://www.arduino.cc

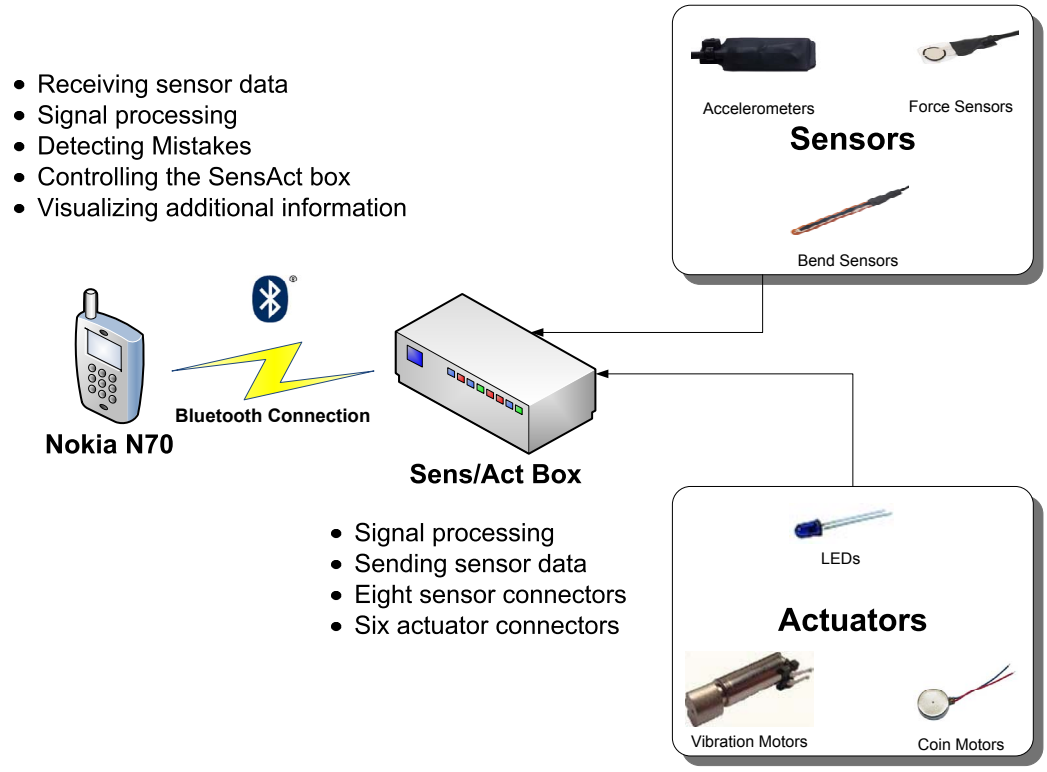

Figure 3.1: System architecture of the Sensor/Actuator Platform.

Custom motor shield

We used a custom motor shield that was developed at our group on the top of the Arduino. This motor shield allows direct attachment of sensors and actuators (Figure 3.2). The shield has a separate electric circuit for the actuators in order to not influence sensor values.

We used a Nokia N70 mobile phone as a host device for our detection algorithms. The phone has 32 MB RAM and an ARM-9 CPU running at 220 MHz.

3.2.1 First hardware prototype

Size and the interior of the SensAct box

Based on the prior work on the Snowboard assistant by Guggenmos, we chose a different case for our prototype. This case (15cm x 8cm x 5cm) is a little bit larger than the prototype of the prior work and offers space for the

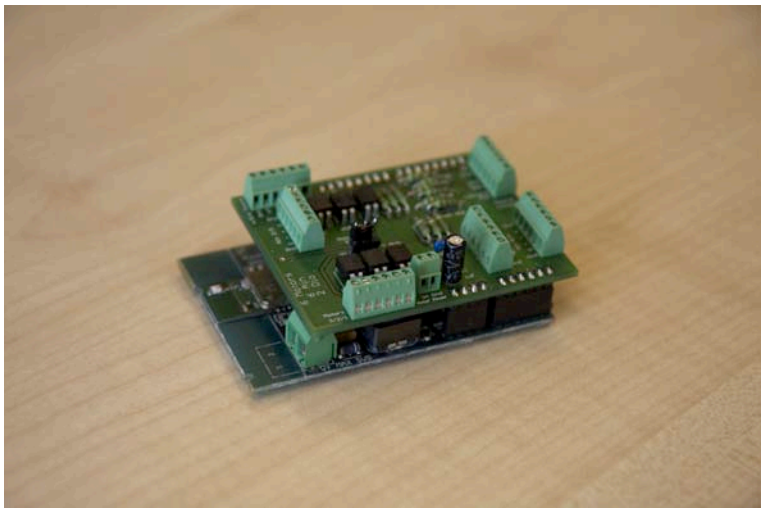

Figure 3.2: ArduinoBT and the custom motor shield on the top. This shield offers connectors for six sensors, eight actuators, a LED, and an externel power supply for the actuators.

ArduinoBT, the custom motor shield, two batteries for the ArduinoBT, and four batteries connected to the motor shield for the actuators.

TRS connectors for sensors and TS for actuators

To provide an easy way to connect sensors and actuators, we chose 2.5mm TS (tip/sleeve) connectors for the actuators and 2.5mm TRS (tip/ring/sleve) connectors for the sensors. Thus, we can combine different sensors and actuators to adapt our hardware to a specific application domain.

Additional hardware parts of the SensAct box

Finally, the first hardware prototype had six sensor connectors, eight actuator connectors, an ON/OFF button and a status LED, which offers the user information about the state of the hardware, such as booting, running, streaming, or low battery status.

Figure 3.3 displays the first version of the *SensAct box*. We attached all connectors on the front of the box and chose the upper row for the sensors and the lower row for the actuators. We mounted the status LED also on the front of the case and ON/OFF button on the left side.

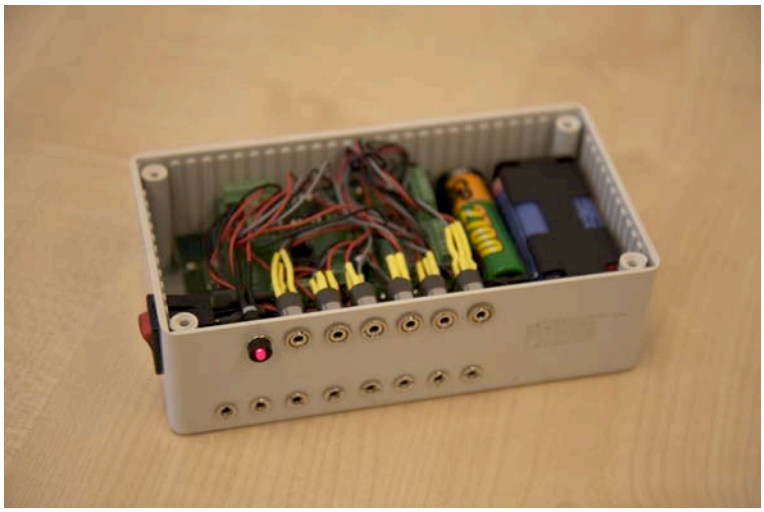

Figure 3.3: First Hardware Prototyp

Discussion

Evaluation on an indoor slope

We tested the box while recording sensor data from snowboarders on the indoor slope SnowWorld[2] in Landgraaf. For recording sensor data, we chose a sampling rate of 50 Hz. By using the TRS connector, we could easily choose which sensors we want to use for the next recording. The box was robust enough for outdoor use, since the box survived all downfalls of some snowboarders during the recordings. Figure 3.4 shows a snowboarder with the first *SensAct box* during two exercises. The snowboarder wears all sensors at once. Thus, for recording different sensor configurations, we only needed to attach the specific sensors.

TRS connectors are error-prone

However, during the recording phase we identified two important problems. One drawback was that the used TRS connectors are not suited to attach sensors to our platform. The box often crashed while connecting or replacing sensors between two runs. The reason for that was a short circuit triggered by the plug, which connects two poles. This made us to switch-off the box before every changes of the sensor configuration.

[2] http://www.snowworld.com

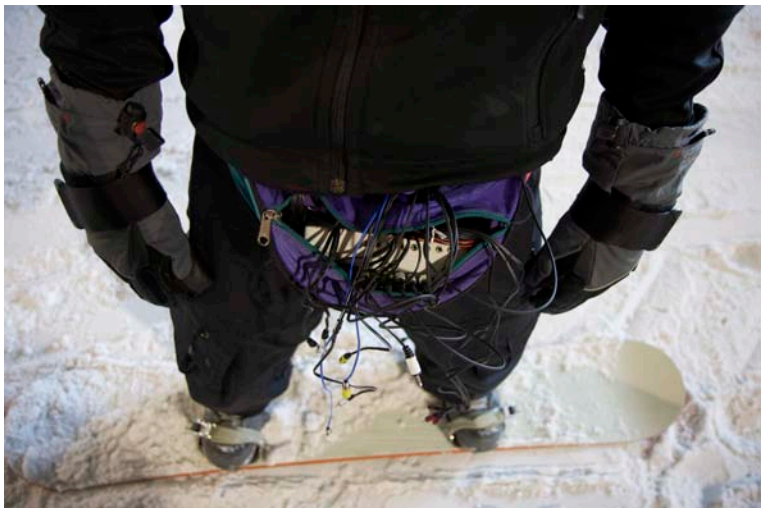

Figure 3.4: First hardware prototyp in a bumbag on the slope.

Loose connectors cause signal noise

A further drawback was that these connectors cannot fasten the sensors to the box properly. It is possible to move these connectors slightly, which leads to wrong sensor values. We observed these sensor value noises after evaluating the sensor recordings that sensor values suddenly raise up to the maximum value when we move these connectors.

Drawbacks made the box unfeasible for outdoor use

Finally, the box offered only 6 analog inputs and did not allow to sense both feet with each three force sensors and both knees at the same time. We realized that the box is usable in the field conditionally, but suitable in the lab. Hence we decided to build an improved *SensAct box* that overcomes these problems.

3.2.2 Improved hardware prototype

Improved box with new connectors

During the recording phase we decided to build quickly a new version of the *SensAct box*. Due to the connector problems, we decided to use another type of connectors for sensors. The new connectors should solve the problems of short circuits and the sensor value noises caused

by slightly moving connectors. Furthermore, it should be possible that sensors can be fixed to the box. Since we noticed no problems with the TS connectors for the actuators, we decided to change only the sensor connectors.

D-Sub connectors for sensors

Thus, we chose 9-pin D-Sub (DE-9) connectors for the sensors and built an improved hardware prototype. We combined more sensors to one connector, which, however, leads to a small limitation of the variety of sensor configurations. For instance, one connector for three force sensors, one connector for two bend sensors, and one connector for a 3-axis accelerometer. In contrast to the first *SensAct box*, it is not possible to use only one force sensor with only one bend sensor with the improved box.

Additional hardware parts of the SensAct box

Figure 3.5 displays the first version of our improved *SensAct box*, which has three D-Sub connectors for the sensors, eight TRS connectors for the actuators, and a status LED on the front. The ON/OFF button remained in the same position. This version of the *SensAct box* contains an unmodified ArduinoBT board that only offers six sensor connections.

Figure 3.5: Second Hardware Prototype: First version with only six sensor connectors.

Improved box built out of necessity

The first version of the improved *SensAct box* we built out of necessity. During the evaluation phase, we realized the

error-proneness of the first *SensAct box* and we immediately needed an new box that overcome these problems. Therefore, we quickly built this first improved version as a transition box before we develop the final one.

Final version of the SensAct box

Figure 3.6 shows the final improved version of the *SensAct box*, which differs in the amount of now eight sensor connectors and the space-saving connector placement on the box. By using an modified ArduinoBT board, we increased the amount of sensor connections. Based on the fact that the first improved version of the *SensAct box* has the same connectors, we want also extend the amount of sensors that could be attached to the first improved *SensAct box*. Comparing to the first improved version, the connectors are arranged in a space saving way. That is very important, since the *SensAct box* is a really small case that offers not so much space. That was also the reason for the placement of the status LED on the left side. However, due to the convex form, the LED is still visible from the front of the box.

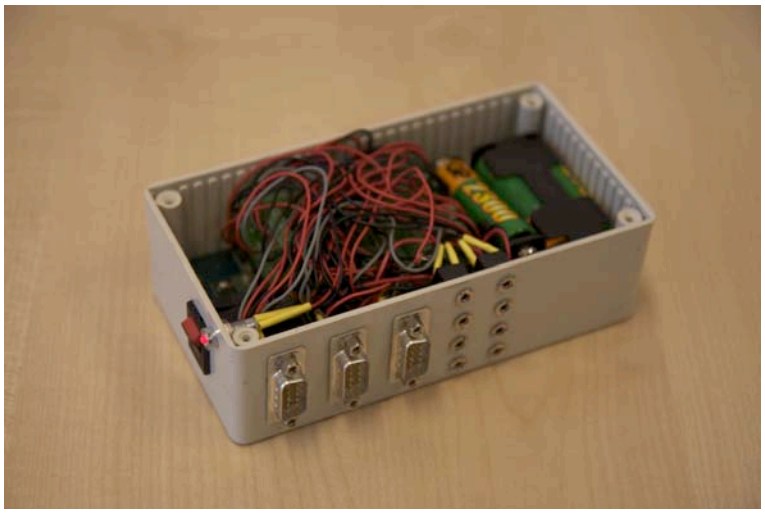

Figure 3.6: Second Hardware Prototype: Final version with up to eight sensor connectors.

Discussion

Evaluation on an indoor slope

Since we developed the improved *SensAct box* during the

recordings, we could evaluate the new hardware prototype in the same recording phase. This time we did not notice any of the problems that we had experienced with the first *SensAct box*. We had no system crashes caused by short circuits as well as no sensor value problems caused by slightly moving connectors.

All hardware problems were solved

The improved prototype allowed changing the sensors at runtime. It was possible to record several runs with different sensor configuration without restarting the hardware. The box worked properly during the whole recordings, that often took several hours. Due to the fact that we used D-Sub connectors, it was possible to fasten the sensors to the box.

Increasing the amount of sensors that can be connected

Finally, we modified the ArduinoBT and made it possible to attach further two sensors. Thus, the final version of our box offered the possibility to attaching and streaming from up to eight sensors at the same time.

3.2.3 Sensor and Actuator Configuration

A small set of sensors and actuators

Besides the *SensAct box* and the mobile phone, the system consists of different sensors and actuators. To made the sensors and actuators usable with the *SensAct box*, we had to prepared them with the same connectors like the boxes have. Therefore, we provided a variety of sensors, such as force sensors on insoles, bend sensors for measuring limb bending, and an accelerometer to detect different activities.

Sensors for all types of boxes

According to the used box, we had sensors with TRS connectors as well as D-Sub connectors. Depending on the mistakes, which the algorithm should detect, the user can choose among a set of sensors and adapt the sensor configurations to his own needs.

Force Sensors

Force sensors for detecting weight distribution

To calculate the weight distribution and other movements depending on the weight distribution, we used force

sensitive resistors (FSR) and investigated, which positions under the feet are most suitable for detecting weight distribution. The FSR sensors reacts on pressure and the higher the pressure, the higher the sensor value. We attached sensors under the heel, the inner and outer ball, and toe of each foot and analyzed the sensor values.

Investigation of force sensor positions under the feet

We tried several combination of different sensor placements under the feet and found out that using ball and the heel of each foot was sufficient for detecting weight distribution. Force sensors under the toe are not suited for mistake detection, since the user can move his toe and influence the sensor value without any other body movements. We did not investigate sensor data from the outer ball FSR more precisely, but we supposed that this position could be also useful for weight distribution.

Figure 3.7 shows the different force sensor positions under the feet. The red circles specify the positions that we used for detecting weight distribution. The blue circles describes the positions under the toe and the outer ball that we either recorded and dismissed or recorded and not investigated more precisely.

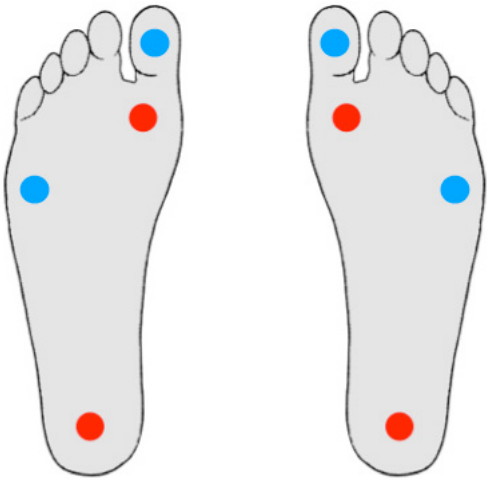

Figure 3.7: Force sensor placement under the feet. The red circles specify the position that we used for the weight distribution detection.

Optical Bend Sensors

Optical bend sensors

To detect limb bending, bend sensors are well suited. Common bend sensors have not a linear response and sometimes offers not so precise sensor values. Since such bend sensors are partially suited for detecting limb bending, we used optical fiber sensors (Kuang et al. [2002]) for detecting limb bending. Such optical bend sensors are well suited, because they offer highly linear response. Additionally, optical bend sensors are thicker and more robust than the common bend sensors, which makes them less error-prone.

Sensor configuration

The optical bend sensors consists of a LED and a photo cell, and a fiber cable, that is abraded at the middle, between them. The LED flashes through the fiber cable to the photo cell. While bending the fiber cable, the abraded section leads to the reduction of the light intensity that is used as an indicator for bending. The more the fiber cable is bended, the more light is lost at the abraded part of the cable. To protect these sensors, we enclosed them into a foam cover, which is shown in Figure 3.8. For the power supply of the LEDs, the actuator connectors can be used. We combined two bend sensors to one D-Sub connector.

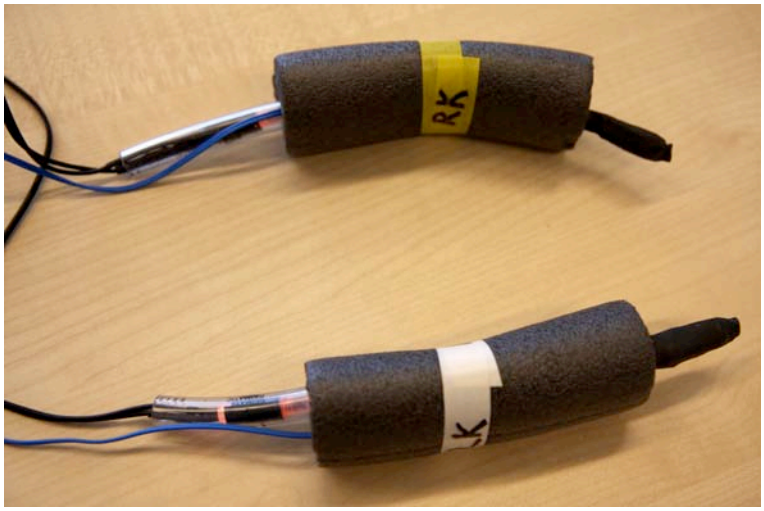

Figure 3.8: Optical bend sensors in a foam cover.

Accelerometers

Accelerometer used for activity recognition

Accelerometers can measure the the tilt as well as the acceleration in different directions. We used 3-axis accelerometers for detecting activities such as standing or walking as well as for detecting back postures. To find an optimal placement for this accelerometer, we investigated different positions on the body such as the user's ankle, chest, or back. Finally, we used the position on the back between both shoulders to detect back postures. For detecting simple activities, we could used either the back or the chest of the user. We attached one 3-axis accelerometer to one D-Sub connector.

Actuators

LEDs and vibration motors as actuators

In addition to the sensors, we prepared a set of actuators in order to use it with the *SensAct box*. Therefore, we attached to cylindrical vibration motors and to LEDs TS connectors, which allows to connect these actuators to the box easily. Figure 3.9 shows the cylindrical vibration motors that we used for our system and the final version with a cable that can be connected to the *SensAct box*. By using the PWM output pins of the ArduinoBT for the actuators, the intensity for each actuator can be adjusted.

3.3 Software Implementation

Python for data logging, Java ME for data processing

Guggenmos [2007] used a Python script on the mobile phone to record sensor data. Python is a high-level programming language, which can be used to create full functional programs and prototypes rapidly. However, Python works slowly and is unsuitable for real-time signal processing.

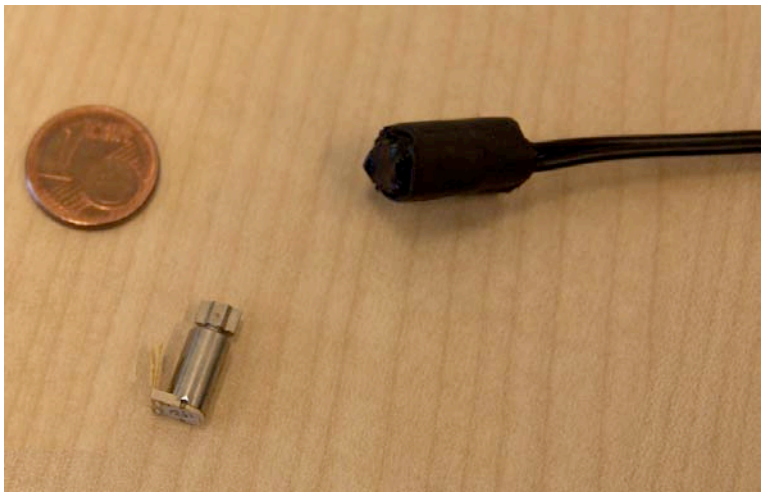

Figure 3.9: The cylindrical vibration motors that we used for the *SensAct box*. On the right side is the prepared motor that can be directly connected to the box.

Thus, we chose Java ME for detecting movements and mistakes because it fulfills the following requirements. Java ME

- is fast enough for signal processing that we used in our work. We conducted several initial tests before starting with the implementation.
- offers a high level and a low level UI to display rapidly additional domain specific information on the mobile phone screen.
- applications run on different mobile platforms that we tested with different mobile phones (Nokia N70 and SonyEricsson W800i). This eases the transition to newer mobile phones.
- eases the transition to Android. This is important, because we want to have an open-source system in the near future.

Algorithms for basic movement detection

The software should be applicable to different application areas such as sports, health care, or the everyday life. Therefore, the recognition algorithms should not be

tailored on one specific domain. We wanted to implement a number of small algorithms that can detect basic movements, such as bending limbs, and can be combined to detect more complex movements like lifting something up.

The collection of basic movements, that can be detected by the system, consists of:

- **Weight Distribution:** Our software distinguishes between the left and right weight distribution as well as the front and back weight distribution.
- **Limb Bending:** Here, the system detects bending and straightening limbs.
- **Back Bending:** Our software recognizes back postures. Thereby, it differentiates between forward and sidewards back bending postures.
- **Simple Motion Recognition:** This function detects simple body movements and can be used to distinguish between activities such as standing, walking or running.

Although the collection of the basic movements is small, the system can be adapted to lots of different physical activities in different application areas. The reason is that our system seamlessly allows any combination of the algorithms.

iSense as a forge for our algorithms

For developing our recognition algorithm, we investigated several methods on recorded off-line sensor values by using the application *iSense* (Figure 3.10). This program was written for the *Snowboard Assistant* project and allows testing several filters and arithmetic operations on the sensor values. We used *iSense* as a testbed for new algorithms that we implemented for the mobile phone later on.

3.3.1 Preprocessing sensor data

Raw sensor signals contain lots of noise

Based on different characteristics of application domains, especially fast movements in sports, most of our detection

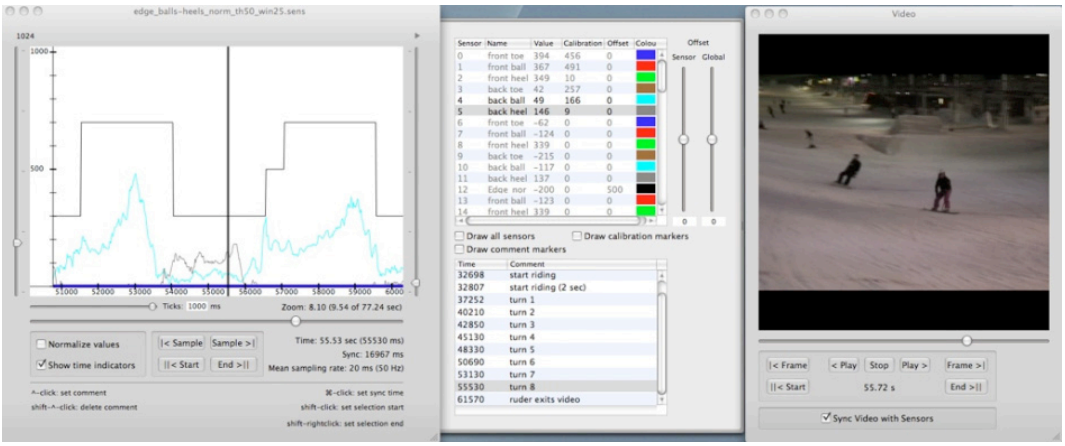

Figure 3.10: iSense: A testbed for our algorithms that offers a couple of mathematical tools for sensor value modifications.

algorithms cannot operate on raw sensor values. These raw signals contain lots of noise, which occur under different circumstances while performing specific physical activities. For example, in snowboarding small bumps on the slope disturb the sensor value of the force sensors under the feet. Therefore, we had to preprocess the data in order to filter such interferences. To reduce the signal noise, we used two types of low-pass filters.

- Simple Moving Average (SMA)
- Exponential Moving Average (EMA)

Both smoothing filters are described in the appendix E.

Normalization of the Sensor Values

Most of our algorithms need a reference value to recognize specific mistakes. Due to the fact, that the sensors differ a little in their behavior, we need a reference value to compare them directly to each other. Thus, we need to calibrate the sensors and to normalize the sensor values before starting mistake detection.

Calibration values in reference values for the algorithms

Sensor value snapshot is not sufficient as reference value

To calibrate the sensors, we can either take a sensor value snapshot or calculate the mean of sensor values for several seconds. A snapshot of the sensor values is not sufficient as reference, because it could happen that the user's posture is wrong at this short moment. Therefore, we calibrate the sensors by calculating the arithmetic average of the sensor values in a neutral position for several seconds. This is a common procedure and is, for example, practiced by Brunelli et al. [2006]. After that, we used the calibration values $val^{cal}(s)$ to normalize the raw sensor values $val(s)$:

$$val^{norm}(s) = val(s) - val^{cal}(s)$$

Thus, the reference values for our mistake detection algorithms becomes zero.

3.3.2 Detecting Weight Distribution

Two aspects of weight distribution

We used the preliminary idea of the snowboard assistant project to detect the weight distribution of the user and distinguished between two aspects of the weight distribution:

1. Left-Right Weight Distribution that distinguishes between:
   - more weight is on the right foot.
   - more weight is on the left foot.
   - the weight is distributed equally on both feet.
2. Front-Back Weight Distribution that distinguishes between:
   - more weight is on the toes.
   - more weight is on the heels.
   - the weight is distributed equally between toes and heels.

Weight Distribution Algorithm

To detect left-right weight distribution, we used the preliminary considerations of Guggenmos [2007]. Before the

algorithm started to work, all corresponding force sensors (FSR) have to be calibrated:

- Calibrating the sensor values from all FRS sensors, while the user is standing in a neutral position.

  1. Left Foot:

  $$\text{left ball: } F_{lb}^{cal} = CAL(FSR_{lb})$$
  $$\text{left heel: } F_{lh}^{cal} = CAL(FSR_{lh})$$

  2. Right Foot:

  $$\text{right ball: } F_{rb}^{cal} = CAL(FSR_{rb})$$
  $$\text{right heel: } F_{rh}^{cal} = CAL(FSR_{rh})$$

After having calibrated the sensors, the weight detection algorithm works as follows:

1. Reading the actual sensor values from all FSR under the feet:

   (a) Left Foot:

   $$\text{left ball: } F_{lb} = val(FSR_{lb})$$
   $$\text{left heel: } F_{lh} = val((FSR_{lh})$$

   (b) Right Foot:

   $$\text{right ball: } F_{rb} = val(FSR_{rb})$$
   $$\text{right heel: } F_{rh} = val(FSR_{rh})$$

2. Normalizing the sensor values by subtracting the calibrated values from the actual ones. Thus, the reference value becomes 0.

   $$F_{lb}^{norm} = F_{lb} - F_{lb}^{cal} \qquad F_{lh}^{norm} = F_{lh} - F_{lh}^{cal}$$
   $$F_{rb}^{norm} = F_{rb} - F_{rb}^{cal} \qquad F_{rh}^{norm} = F_{rh} - F_{rh}^{cal}$$

3. Filtering actual sensor values by using EMA.

4. Calculating the sum of the sensors of one foot and the difference between both feet.

$$\begin{aligned} \Sigma F_{lf} &= EMA(F_{lb}^{norm}) + EMA(F_{lh}^{norm}) \\ \Sigma F_{rf} &= EMA(F_{rb}^{norm}) + EMA(F_{rh}^{norm}) \\ \Delta F_{lf-rf} &= \Sigma F_{lf} - \Sigma F_{rf} \end{aligned}$$

5. Calculating the SMA of ΔF_{lf-rf}, by determining the mean weight distribution of a predefined window size of w samples.

6. The algorithm outputs the weight distribution based on a predefined threshold T that defines a tolerance range around the reference value.

$$Output = \begin{cases} right & \text{,if } SMA(\Delta F_{lf-rf}) < -T \text{ ,} \\ left & \text{,if } SMA(\Delta F_{lf-rf}) > T \text{ ,} \\ center & \text{,otherwise} \end{cases}$$

The front-back weight distribution works similarly and differs only in step four. To detect the weight distribution towards the front or the back, step four must be replaced by:

$$\begin{aligned} \Sigma F_{balls} &= EMA(F_{lb}^{norm}) + EMA(F_{rb}^{norm}) \\ \Sigma F_{heels} &= EMA(F_{lh}^{norm}) + EMA(F_{rh}^{norm}) \\ \Delta F_{balls-heels} &= \Sigma F_{balls} - \Sigma F_{heels} \end{aligned}$$

Additionally, the algorithm output must be adapted to front, center, and back.

Discussion

In the worst case, the detection takes w samples

The algorithm calculates the weight distribution and filters most of the signal noise by using two low pass filters. In the worst case, the algorithm detects a change of the weight distribution after w samples. w describes the window size that the SMA used for the calculation. The reason for that is the usage of the SMA and a simple threshold value T. To explain the w samples of the worst case, we provide a small example. Assuming that $w = 10$ and $T = 40$. Furthermore, the last $w - 2$ sensor values as well as the actual value are

41 and the $w - 1$ (oldest) value is 35. This leads to a SMA value of 40 that is equal the threshold $T = 40$. However, the algorithm detects a change in the weight distribution only if $SMA > T$. If the next value is greater than or equal 41 the algorithm calculates a SMA value of 41 and detects a change in the weight distribution.

In the best case, the algorithm needs one sample

In the best case the algorithm detects a change of the weight distribution after one samples. Assuming that $w = 10$ and $T = 40$. Furthermore, the last $w - 1$ sensor values are 40. When the actual sensor value is 50, the algorithm calculates a SMA of 41 and detects, only after one sample, the change of the weight distribution.

Both examples showed, that the time, which the algorithm needs to detect the weight distribution, ranges from one to w samples. Basically, both examples only rarely occur and very often the algorithm detects the weight distribution with an average of $w/2$ samples. Based on the sampling rate and the window size w, the algorithm is usable for different applications. For instance, we used a sampling rate of 50 Hz and a window size of $w = 25$ (0.5 sec.) for snowboarding.

3.3.3 Detecting Limb Bending

Two aspects of limb bending

Based on the prior work, this algorithm detects the bend of the limbs by processing the data from a bend sensor mounted on the specific limb. In other words, the algorithm can either be used to detect leg bending or arm bending. Basically, the mode of operation of this algorithm is analogous to the weight distribution algorithm.
We distinguished between the two aspects of limb bending:

1. The limb is straight.
2. The limb is bend.

Limb Bending Algorithm

Besides the FSR sensors, the optical bend sensors (OBS) offer the same signal noise characteristics. Therefore, we also used the same low-pass filters EMA and SMA to reduce the signal noise.

Before being able to work, we need to calibrate the bend sensor. Therefore, the user must band the corresponding limb slightly during the whole calibration phase:

$$O_{limb}^{cal} = CAL(OBS_{limb})$$

After having calibrated the sensor, the limb bending detection algorithm works as follows:

1. Reading the actual sensor values from the optical bend sensor attached to the specific limb.

$$O_{limb} = value(OBS_{limb})$$

2. Normalizing the sensor values by subtracting the calibrated values from the smoothed actual ones. Thus, the reference value becomes 0.
3. Filtering actual sensor values by using exponential moving average.

$$O_{limb}^{norm} = EMA(O_{limb} - O_{limb}^{cal})$$

4. Calculating the SMA, by determining the mean of the last w samples.
5. The algorithm detects limb bending based on a predefined threshold T, which defines a tolerance range around the reference value.

$$Output = \begin{cases} bend & \text{,if } SMA(O_{limb}^{norm}) < -T \\ straight & \text{,if } SMA(O_{limb}^{norm}) > T \end{cases},$$

Discussion

Same characteristics as the weight distribution algorithm

The algorithm works almost similar to the weight detection algorithm and filters most of the sensor signal noise, while retaining a fast reaction time. In the worst case, the detection takes w samples and only one sample in the best case. This results can be also explained with an example similar to the example of the weight distribution algorithm.

3.3.4 Detecting Back Postures

Simple back posture detection by using an accelerometer

Since accelerometers can be used to measure joint angles and joint postures (Hansson et al. [2001]), we used an accelerometer and implemented a simple back posture detection. To detect back postures, the algorithm reads sensor values from an 3-axis accelerometer which is mounted on the back between both shoulder blades of the user. Figure 3.11 shows the mounted accelerometer with three axes that we used for detecting back postures.

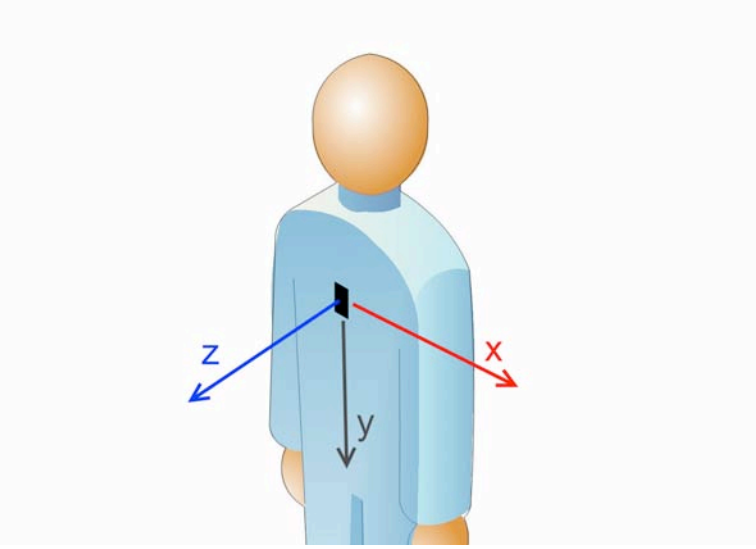

Figure 3.11: 3-axis accelerometer mounted on the upper body of the user.

For front-back bending the algorithms analyses the y-axis of the sensor. However, it is also possible to detect the front-back bending with the z-axis. We distinguished between two back postures:

1. The back is straight.
2. The back is bend.

Back Posture Detection Algorithm

Before starting to work, we need to calibrate the accelerometer ACC. Therefore, we calibrate the sensors, while the user slightly bends his back posture.

$$A_y^{cal} = CAL(ACC_y)$$

Then the algorithm works for the front-back bending (y-axis) as follows:

1. Reading the actual sensor values from the y-axis of the mounted accelerometer.

$$A_y = val(ACC_y)$$

2. Normalizing the sensor values by subtracting the calibrated values from the actual ones. Thus, the reference value becomes 0.
3. Filtering actual sensor values by using exponential moving average.

$$A_y^{norm} = EMA(A_y - A_y^{cal})$$

4. Calculating the SMA by determining the mean sensor value of the last w samples.
5. The algorithm detects the back posture based on a predefined threshold T, which defines a tolerance range around the reference value.

$$Output = \begin{cases} bend & \text{,if } SMA(A_y^{norm}) < -T \\ straight & \text{,if } SMA(A_y^{norm}) > T \end{cases},$$

Discussion

This algorithm has the same attributes as the other two algorithms before. By using an other axis for the bending calculation, the algorithm can be used for other back bending directions. Additionally, in the worst case, the detection takes w samples and one sample in the best case.

3.3.5 Simple Activity Recognition

Simple activity recognition by using an accelerometer

Activity recognition by using body mounted accelerometers is realized in lots of different projects (Knight et al. [2007], Ravi et al. [2005], Yang et al. [2007], Mantyjarvi et al. [2001]). Some of these approaches presented an activity recognition system that can detect a variety of different activities such as walking downstairs and walking upstairs. Therefore, some systems use machine learning techniques such as the Hidden Markov Model and need a training phase before the system can be used.

Standard deviation to detect simple activities

Although these algorithm can recognize different movements with a high accuracy, we decided to implement a simple activity recognition that recognizes simple activities. The reason for that was, we wanted to have simple algorithms that can be easily combined among each other and to run on a mobile phone. Thus, the simple activity recognition can be used to determine whether the user is standing, walking, or running. Figure 3.12 shows the algorithm output while running. For distinguishing between static states and dynamic states, the standard deviation is an adequate mathematical feature(Baek et al. [2004]).

Simple Activity Recognition Algorithm

Activity recognition as an automatic ON/OFF switch

Simple activity recognition can be used to automatically activate and deactivate the recognition system as well as to measure the total time that the user was walking or running. To detect these simple movements the accelerometer can be mounted at the user's chest in the same orientation

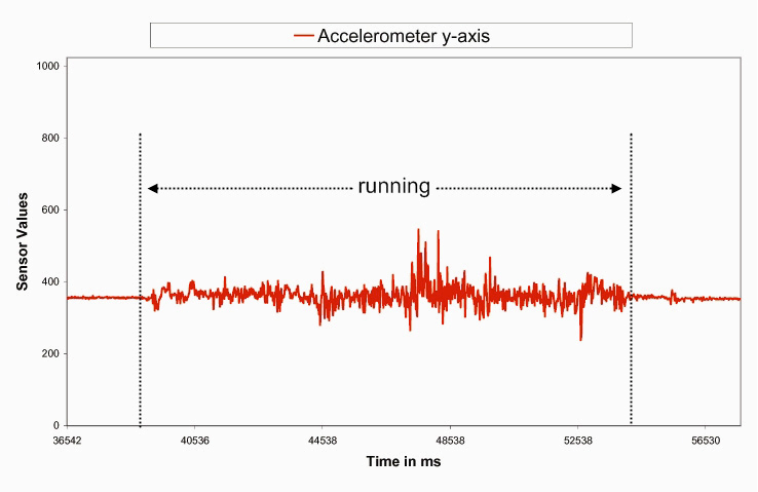

Figure 3.12: Sensor values of an accelerometer (y-axis) while running. By using accelerometer data, we could clearly distinguish between running and standing positions.

as we used the accelerometer for the back posture detection. Based on the moving direction, the accelerometer axis must be chosen. The SMA is calculated for the standard deviation that is used to determine movements. This activity recognition algorithm requires no calibration and works for the y-axis direction as follow:

1. Reading the value of the mounted accelerometer.

$$A_y = val(ACC_y)$$

2. Calculating the moving average $SMA(A_y)$, by determining the mean sensor value of the last w samples.

3. Calculating the standard deviation at time t of the last w samples, while using $SMA(A_y)$.

$$\sigma = \sqrt{\frac{1}{w}\sum_{i=1}^{w}((A_y)_{t-i+1} - SMA(A_y))^2}$$

4. The algorithm detects movements based on a prede-

fined threshold T:

$$Output = \begin{cases} standing & \text{,if } \sigma < T_1 \text{ ,} \\ walking & \text{,if } T_1 < \sigma \end{cases}$$

Discussion

The algorithm distinguishes only between similar activities

The algorithm recognizes simple activities without requiring much computational power. In contrast to other activity recognition algorithms, our algorithm can only detect and distinguish between similar activities like walking or running. The algorithm cannot distinguish between different types of movements such as between climbing upstairs and sit-ups (Ravi et al. [2005]). By providing an additional threshold value, the algorithm can be extended to distinguish between three similar activities such as standing, walking, and running.

3.3.6 Software Library Implementation

Java ME library

After having investigated and optimized these algorithms with *iSense*, we started to implement them with Java ME (CLDC 1.1[1], MIDP 2.0[2])to apply them to mobile phones.
To ease the work with our *SensAct box*, we implemented the library `mcg.arduino` in Java ME that is divided into three parts:

- `mcg.arduino.io`
- `mcg.arduino.move`
- `mcg.arduino.pattern`

These libraries provide a couple of procedures that increase the development time of new mobile phone applications for the *SensAct box*.

[1]Connected Limited Device Configuration - a Java ME application framework

[2]Mobile Information Device Profile - Profile that contain the API for mobile phone functions

mcg.arduino.io

Classes for system communication

`mcg.arduino.io` only contains the class `ArduinoComm`, which is the core class for the communication between the mobile phone and the *SensAct box*. This class offers lots of procedures for establishing and controlling the hardware platform. The user can connect to the *SensAct box* rapidly only via Arduino's MAC address, start and stop the sensor data stream, and control the attached actuators. Furthermore, the user can adjust the intensity for each attached actuator.

mcg.arduino.move

Classes for motion detection

`mcg.arduino.move` is the collection of all motion recognition algorithms, which we discussed in section 3.3. Each algorithm is implemented in one class in such a way that the user can seamlessly combine them for creating a wearable prototype.

Motion detection classes contain only a calculate() function

All classes contain only one `calculate()` function, which operates on two parameters. The first parameter is an array with current sensor values. The second parameter contains a flag array, where each algorithm sets a specific flag based on its output. Finally, the application can easily detect a complex movement, which consists of elementary movements, by analyzing the flag array.

mcg.arduino.pattern

Classes for simple feedback patterns

`mcg.arduino.move` is a small collection of feedback patterns, which contains simple feedback patterns for attached actuators such as motors or LEDs. However, the pattern collection only contains simple patterns like ON/OFF pattern or sinusoidal PWM output, but it can be extended with more feedback patterns.

Simple usage of the library

To demonstrate how simple it is to implement applications, we provide a pseudo code solution for the lifting

example from the introduction. Most persons bend their backs and straighten their knees to lift up a box from the ground. This is a common mistake that the most of us do regularly. The small pseudo code example demonstrates how simple it is to build a wearable assistant, which detects this mistake and provides feedback to the user.

```
calculateActivity();
if standing then
    calculateKneeBending();
    calculateBackPosture();
    if kneesAreStraigth && backIsBent then
        provideTactileFeedback(pattern1);
```

This application provides only feedback to the user on a wrong lifting technique when the user is standing. When the user is walking, the system is deactivated. An overview of all procedures of the library is available in the appendix of this thesis (Appendix A).

3.3.7 Supporting Software for Wearable Prototype Designs

Tools to support the wearable system designer

After having developed a hardware platform, which allows the user to attach sensors and actuators easily, and a software, which allows simple programming of recognition algorithms, we implemented two tools for the mobile phone.

These tools can help the user building a wearable prototype, by supporting him with the adjustment of sensors and actuators. Additionally, these tools can be used to perform a system check on the hardware platform. We developed separate tools for sensors as well as for actuators.

Sensor Monitor

Displaying sensor values as graphs

The *Sensor Monitor* is a tool that displays sensor values as graphs in real-time on the mobile phone's screen. While

streaming, the user can choose between the sensor values that should be displayed on the mobile phone screen.

Verifying the functionality and positions of sensors

This tool can help the user to verify the sensors' functionality and correct placement on the body. Figure 3.13 shows the main screen of the Sensor Monitor. While the user is shaking the accelerometer, the tool displays the sensor values as graphs in realtime. We often used this tool

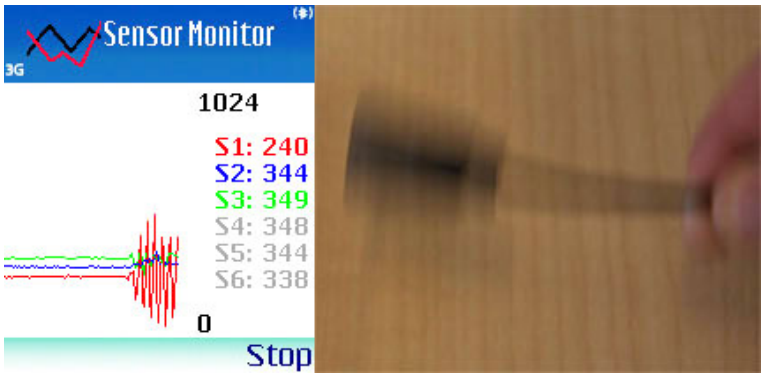

Figure 3.13: Sensor Monitor displaying accelerometer data on a mobile phone in real-time.

during our recording sessions on the slope. Due to the fact that sometimes the sensors slightly displace their positions and deliver wrong values, we immediately could identify this effect before starting the recordings. A small sample run is shown in the appendix of the thesis (Appendix C).

Motor Control

Test the functionality of actuators

Motor Control allows verifying the functionality of attached actuators. The user can test the functionality and different intensities of the actuators attached to the hardware platform. 3.14 displays the main screen that shows an image of the *SensAct box*, whereas the user can use the mobile phone joystick for triggering the actuators.

Figure 3.14: Motor Control Application on a mobile phone.

Basically, we used this program to provide a system check on the actuators after having built or repaired the *SensAct box*. A small sample run is shown in the appendix of the thesis (Appendix D).

Chapter 4

Evaluation in the Snowboarding domain

"Fast is fine, but accuracy is everything."

—Wyatt Earp

Snowboard beginner would benefit from having a snowboard instructor

As a part of the snowboard assistant project, we chose the snowboarding domain for evaluating our sensor / actuator platform. Based on results of the prior work by Guggenmos, a snowboard assistant that supervises snowboard beginners during their learning phase would be very beneficial for everyone who starts to learn snowboarding.

Therefore, we tested and evaluated the weight distribution algorithm, the simple activity recognition, and the limb bending algorithm in the context of snowboarding and investigated how well common beginner mistakes could be detected. In addition to these algorithms, we implemented and evaluated a new turn detection algorithm that need no calibration phase before starting to work.

Common beginner mistakes

Guggenmos [2007] conducted interviews with snowboard instructors and classified four common beginner mistakes:

- **Wrong Weight Distribution:** Wrong weight distribution during the ride is a common beginner mis-

take. Normally, between turns the weight is centered between the front and the back foot. During a turn, the weight is more on the front foot. When descending the slope with a considerable speed, most of the beginners are afraid on distributing their weight towards the front foot. Moreover, they tend to shift the weight towards the back foot that often results in loss of control.

- **Straight Knees:** Advanced snowboarders bend their knees for better control their weight distribution during the ride as well as to compensate small bumps on the slope. However, beginners tend to do not bend their knees or to do nor bend their knees enough. One reason is that for beginners the situation on the snowboard is new and unfamiliar. In addition, snowboard beginners do not have enough perception of their own body. Indeed some beginners bend their knees during the ride, but not enough.

- **Wrong Upper Body Postures:** Beginners often bend their back in order to look down to the snowboard. This posture oftentimes leads to lose of balance and downfall.

- **Upper Body Counter Rotation:** Beginners sometimes tend to keep their upper body towards the downhill after riding a turn instead of returning to the basic snowboarding stance. This leads to counter rotation between the upper body and the feet, which can be very stressful for the snowboarder.

Algorithms that we focused on

In this work, we focused on the weight distribution problem, the straight knees mistake and partially on the counter rotation mistake. We had no sufficient sensors to detect the upper body counter rotation. Since the counter rotation mistake occurs after the snowboarder riding a turn, we investigated whether it is possible to detect turns and the riding edge during the descend. In addition to that, we tested if our simple activity recognition can be used to detect the descend of the snowboarder. Based on the result, this can be used to activate our system automatically or to measure statistical information such as the time that the snowboarder needs to ride down the slope.

4.1 User Study and Experimental Setup

Experimental setup of the user study

To detect the wrong weight distribution as well as the straight knees of snowboard beginners, we used the weight distribution and the limb bending algorithm of our software library. We could also used the weight distribution algorithm to detect the turns and the riding edge of the snowboarder on the slope. For the detection of the snowboarder's descend, we used a modified version of our simple activity recognition. Before we could evaluate the algorithm and determine their detection accuracy, we conducted a user study and recorded sensor data from eight subjects on the slope. One participant was a beginner, two were advanced beginners, three were advanced, two were professional snowboarders,and one was a snowboard instructor. Each subject descended two times an about 140 meters long part of the slope with the same sensor configuration. We recorded data at 50Hz from different configurations such as three FSR on each foot, 2 FSR on each foot and bend sensors mounted on the knees, and two accelerometers attached to the chest and the ankle of the snowboarder. Thereby, we recorded the sensor data and videotaped the snowboarder during the whole run.

Evaluation of recorded sensor data with iSense

The recorded sensor values as well as the video footage we used to test the algorithms off-line. After having implemented our algorithms by using *iSense*, we also decided to use iSense for the evaluation. One feature of this program is that it can display sensor values and a video footage of the snowboarder synchronously. We used this feature to classify the output of the algorithms. We used the first run of the subjects as the training set for our algorithms and determined their detection accuracy. To verify these results of the first run, we tested the algorithm on the second run that we used as a test set.

4.2 Turn / Edge Detection Algorithm

Before we started to evaluate and determine the accuracy of this algorithm we distinguished between three cases that

can occur:

- **Correct detection (True positives):** An actual turn is recognized by the algorithm correctly.
- **False alarm (False Positives, Type I error):** The algorithm spuriously recognizes a turn, although the snowboarder rode no turn before.
- **Missing turns (False Negatives, Type II error):** The algorithm misses a turn that the snowboarder rode.

Calculation of true negatives is not possible

The calculation of the True Negatives is not possible, because we do not know the total amount of absent turns. In other words, it is possible to determine the amount of turns, but not to determine the amount of riding no turns.

Ranking system

The turn edge detection has two parameters that can be adjusted. The first one is the smoothing factor $alpha$ of the EMA. The second one is the window size w of the SMA. To evaluate the algorithms, we needed to find the best algorithm parameter values that provides the highest detection accuracy. Therefore we used a ranking concept similar to the system presented by Knight et al. [2007]. We calculated a ranking R for each parameter configuration as follows:

$$R = \sharp CorrectTurns - \sharp FalseAlarms - \sharp MissedTurns$$

The ranking R indicates the best parameter configuration, whereas a high ranking corresponds to high detection accuracy.

Using the weight distribution algorithm to detect turns

To evaluate this algorithm, we instructed all participants to descend the slope as they always do. A turn in snowboarding corresponds to a transition from one riding edge to another. In other words, the weight distribution changes from the front to the back or vice versa.

Two independent variables

To determine the accuracy of this algorithm, we had to find the best algorithm configuration with the highest accuracy. Thereby, we hat two independent variables, the smoothing factor $alpha$ and the window size w. Experimental results allowed us to choose a static window size of

25 samples (0.5 sec) for the SMA filter and tested different smoothing factors $alpha$ for the EMA. We observed, that increasing $alpha$ leads to more missing turns, whereas a small $alpha$ value causes more false alarms.

Find the best parameter settings to achieve a high detection accuracy

Hence, we should find an $alpha$ that decreases both the missing turns and the false alarms while increasing the correct detection rate. We investigated $alpha$ values from $[0 - 0.9]$ in 0.1 steps and calculated in each case the ranking R. Based on R, we could determine the best algorithm configuration with the highest accuracy.

4.2.1 Results

All turns were detected in the training set

To evaluate this algorithm, we observed all subjects from the beginning of their descend to the slow down phase at the end of the slope. Finally, all subjects rode 56 turns. For the training set, there were no differences between the several $alpha$ values. Moreover, each algorithm configuration detected all turns. During the whole training set, no false alarms and no missing turns occurred.

Almost all turns were detected in the test set

We used the test set to proof the algorithm results from the training set. Again, we observed the subjects' ride from the starting point up to the breaking point. This time all subjects rode 61 turns. The results of the test set slightly differed from the prior ones. By using a smoothing factor of 0.9, the algorithm achieved the highest ranking and accuracy. This time, the algorithm detected 59 of the 61 turns $(96,7\%)$ and reported two false alarms.

4.2.2 Discussion

Threshold value as sensitivity factor

Figure 4.1 shows the output of the algorithm. The value 50 corresponds to frontside riding and -50 to backside. The threshold value of this parameter can be used as an sensitivity factor. The higher the threshold value, the more weight the user has to distribute towards the toes or heels. Therefore, for beginners a lower threshold value is more suitable. The algorithm detected turns and edges

with an overall accuracy of $98,3\%$ with a smoothing factor $alpha = 0.9$. By using only four FSR sensors, the algorithm achieved a high accuracy.

Algorithm need calibration values

The main drawback is that the algorithm depends on the calibration values of the sensors. A wrong calibration value immediately decreases the detection accuracy. We observed that after each run the FSR sensor positions under the feet slightly changed. Therefore, it is necessary to calibrate the sensors before each ride in order to preserve a high accuracy.

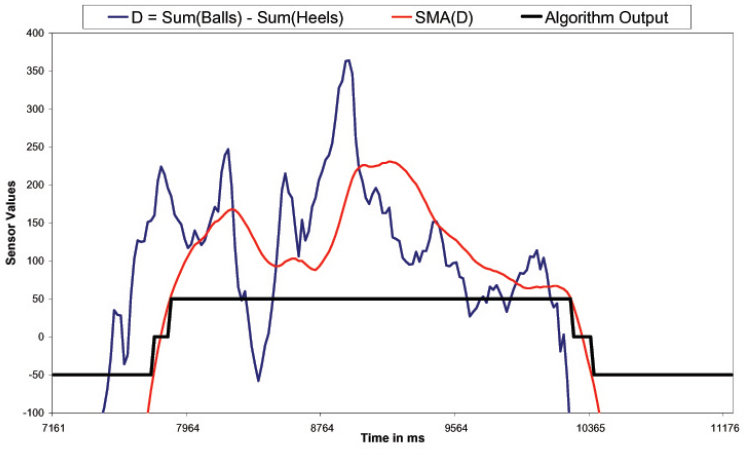

Figure 4.1: Algorithm output of the turn/edge detection algorithm. D is the difference between the toes and the heels. The algorithm output 50 corresponds to frontside and -50 to backside.

4.3 Calibration-free Turn Detection

Turn detection without calibration

During the development process of the weight detection algorithm, we noticed that it is possible to detect turns only by using sensor values from one foot. We saw that a turn corresponds to an intersection point of the sensor values from the ball and the heel of one foot. Additionally, we realized that no calibration was needed for the turn detection. We used the same ranking system that we have used for the

turn / edge detection. To detect turns, this algorithm works as follows:

- Reading and smoothing the sensor values from toe and heel of one foot by using EMA.
- Detecting intersection points as an indicator for an turn.
- The mean duration between two turns across all subjects were 3.4 sec. Based on this fact, the algorithm dismisses all consecutive intersection points that occur within a specific time interval after the last intersection point. The higher this interval, the higher the amount of missed turns. By using a short interval, the amount of false alarms increases. Experimental results allowed us to choose a 25 sample interval that corresponds to 0.5 sec. Since it is improbable to ride two turns within a half second, the algorithm dismisses all intersection points that occur 500 ms after the last one.

Figure 4.2 shows the algorithm output. The intersection points after the first detection are filtered by the 25 samples interval that is extended automatically after each consecutive intersection point.

4.3.1 Results

Using the ranking system for evaluation

We evaluated different smoothing factors $alpha$ by using the same ranking R as we used for the turn / edge detection distribution algorithm. Thereby, we differentiated between the front and the back foot. By using sensor values from the back foot and $alpha = 0.9$, the algorithm detected turns with an accuracy of $91,67\%$ for the training set. By contrast, the algorithm accuracy for the front foot was significantly lower with $59,62\%$ ($p < 0.05$, Student's t-test).

Since the turn detection was significantly better on the back foot, we only evaluated the back foot in the

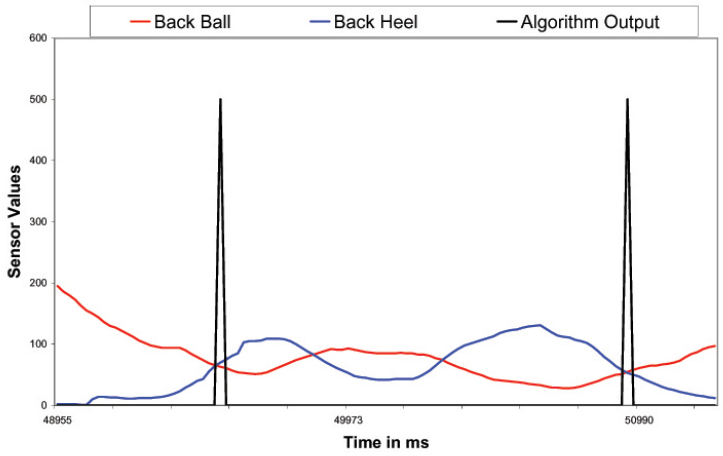

Figure 4.2: Algorithm detects two turns and dismisses consecutive intersection points.

training set. This time the algorithm detected 49 of 53 turns and achieved an accuracy of 92.45%. Additionally, the algorithm missed four turns but had no false alarms.

4.3.2 Discussion

The algorithm used FSR senors only from one foot and detected turns and edges with an overall accuracy of $92,06\%$. In addition to that, the algorithm worked clearly better on the back foot. One reason could be that the subjects distributed their weights toward the back foot, which is a common snowboard beginner mistake. One drawback of this algorithm is that is only detect turns, but not the riding edge of the snowboarder.

Algorithm not implemented in the library

Although the algorithm works without any calibration, we did not include it in our library collection. The reasons for that was the lower accuracy and the fact that the algorithm was too domain specific. The algorithm can only be used for snowboarding or similar sports such as surfing or skating. Nevertheless, we realized that it is possible to detect specific movements without a prior calibration.

4.4 Stop/Go Detection Algorithm

Simple activity recognition to detect stop and go

Basically, we intended to use our simple activity recognition for detecting the descend of a snowboarder. Based on the result, this algorithm can be used either to activate the *Snowboard Assistant* automatically or to measure the time that the snowboarder needs to descend the slope. For the stop and go algorithm evaluation, we instructed the subjects to take short breaks five seconds during their ride. We wanted to test whether it is possible to detect the snowboarder's descend with our simple activity recognition algorithm (3.3.5).

Small modifications of the simple activity recognition algorithm

However, we noticed that the amplitude of only one axis of the accelerometer that we mounted on the user's chest cannot be used to detect the descend on the slope. Therefore, we modified this algorithm slightly by considering two accelerometer axes and calculated the sum of the standard deviation of the x axis and y axis. To evaluate and determine the accuracy of this algorithms, we had to find the threshold value that clearly distinguishes with a high accuracy between stopping and riding.

4.4.1 Results

Snowboard beginners need a smaller threshold value

Figure 4.3 shows the output of the algorithm when applied to the sensor values of the accelerometer. The green curve shows the sum of the standard deviation of the x axis and the standard deviation of the y axis. Due to sensor problems, we evaluated only seven runs with overall 18 pauses from four different subjects. To evaluate the algorithm, we considered besides the several stops also the slow down phase before one pause and the acceleration after that. Experimental results allowed us to use a window size of 25 samples (0.5 sec) for the calculations of both standard deviations. We tested several threshold values and observed that for snowboard beginners a small threshold is suitable, whereas the threshold for advanced snowboarders can be higher. To calculate the accuracy of the algorithm, we chose an average threshold $T = 7$. Thus, the algorithm detected $77,7\%$ of all transitions from riding to

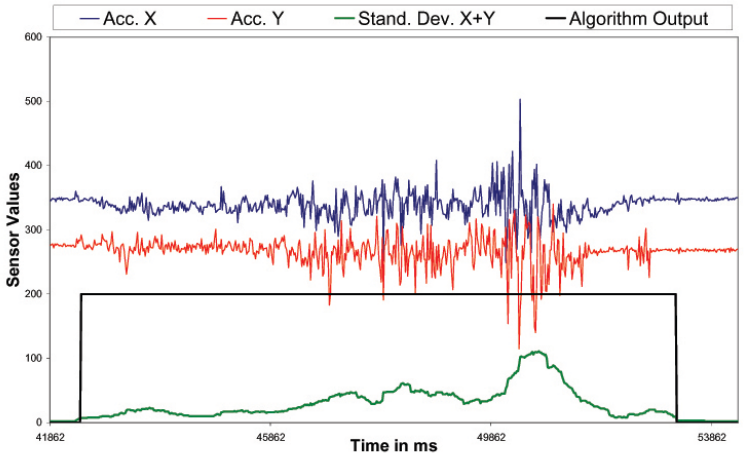

Figure 4.3: Algorithm output of the stop/go detection algorithm.

standing and 83% of all transitions between standing to riding. Hence, the algorithm detects movements with an overall accuracy of $80,5\%$.

4.4.2 Discussion

Difficult to achieve a high accuracy for everyone

We observed, that it is very had to adjust this algorithm in order to work equally well for all snowboarders. One reason for that, is the descending speed of different skilled snowboarders. While advanced snowboarders ride down the slope quickly, beginners tend to be careful and decrease their riding speed regularly.

Snowboard beginners often ride slowly

In addition, beginners cannot control the snowboard as well as advanced snowboarders. Thus, beginners cannot brake or accelerate properly, which leads to a wrong output of the algorithm, especially during the slow down phase. Figure 4.4 shows a typical error during the slow down phase of snowboard beginners.

Based on the observation, the best way for is to differentiate between snowboarder's skills before starting this algorithm.

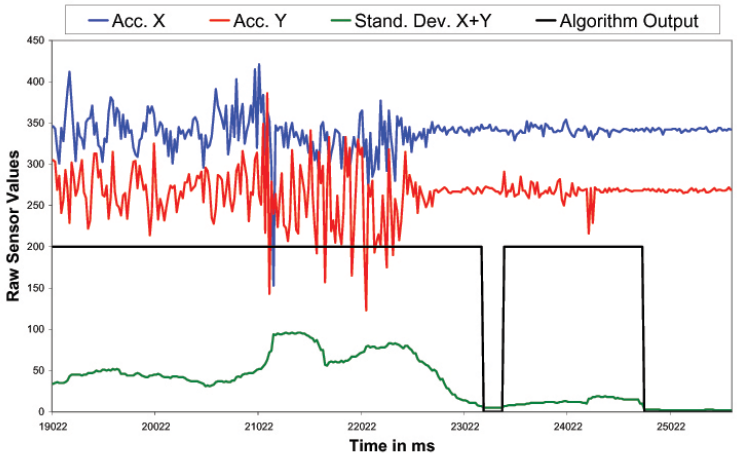

Figure 4.4: Detection error of the algorithm during the slow down phase.

4.5 Weight Distribution Algorithm

Evaluation is not possible without visual comparison

Wrong weight distribution during the ride is a common mistake of snowboard beginners. This algorithm detects the wrong weight distribution of the snowboarder during the descend. Evaluating the wrong weight distribution was not directly possible, because we did not have a visual comparison like in the other algorithm before. Therefore, the weight distribution of the snowboarder was not recognizable on the video footage. Thus, evaluating the best algorithm parameter was not possible with the same evaluation methods that we used for the turn/edge detection or the stop/go detection. NHowever, the evaluation of this algorithm should be possible with other evaluation methods.

Assumption on the detection accuracy

Nevertheless, we act on the assumption that the accuracy of this algorithm is as well as the accuracy of the turn/edge detection algorithm. The reason for that is that both algorithms are almost identical and only differ in the detected direction of the weight distribution (3.3.2). Additionally, informal user tests in the lab, which all were successful, confirmed the high accuracy of this algorithm.

4.6 Straight Knees Detection Algorithm

Evaluation is not possible without visual comparison

This algorithm detects the knee bending of the snowboarder. Some snowboard beginners bend their knees not enough, which is a common beginner mistake. The evaluation of this algorithm offered the same problems as the weight distribution algorithm. Due to the thick snowboarder clothes, we could not directly determine on the video footage whether the knees are bent enough or not. Therefore, we could not evaluate the algorithm and determine the detection accuracy with the same evaluation methods that we used for the other algorithms.

Assumption on the detection accuracy

However, informal tests in the lab were all successful. In addition, we observed at the recorded sensor data, that it is possible to detect straight knees during the descend on the snowboard. We instructed the subjects to straight their knees after each turn. Since it was not always possible to recognize the straight knees on the video footage, the sensor curves of both bend sensors shown in Figure 4.5 made it possible to detect straight knees while descending the slope.

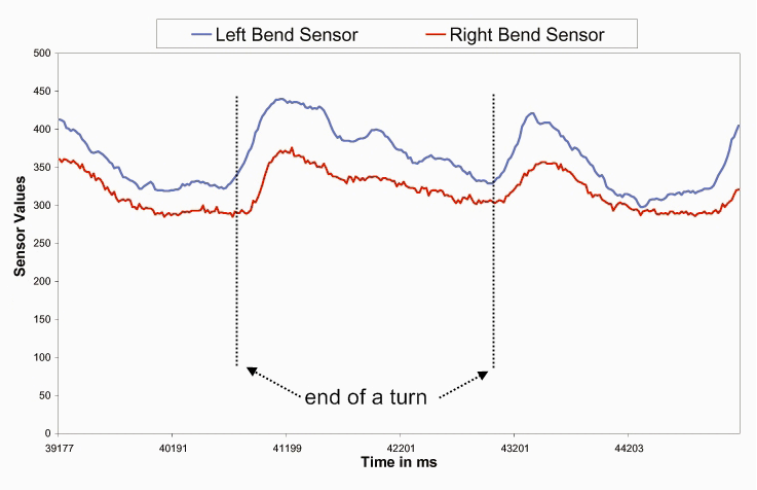

Figure 4.5: Sensor curves of both bend sensors during the descend on the slope. Lower sensor values correspond to bending knees.

Another aspect that allowed us to assume a high detection accuracy is the fact that the algorithm core of the bend detection algorithm is almost similar to the weight distribution algorithm. Thus, we suppose a similar high detection accuracy, but as long as we cannot proof this, we cannot determine this assumption exactly.

4.7 Summary and Discussion

Finally, we want summarize all algorithms with all the parameter values and detection accuracies that we developed and evaluated in the thesis. Table 4.1 shows an overview of all results of the algorithm evaluation. Since we could not evaluate the weight distribution as well as the straight knee detection, this table contains no values for the both algorithms. Nevertheless, we only know that a calibration of the sensors is needed before starting to work.

	Window Size (SMA)	Alpha (EMA)	Calibration	Detection Accuracy
Turn/Edge Detection (4.2)	25 samples(0.5 sec)	0.9	✓	$98,3\%$
Calibration-free Turn Detection (4.3)	-	0.9		$92,06\%$
Stop / Go Detection (4.4)	25 samples(0.5 sec)	0.0		$80,5\%$
Weight Distribution 4.5	-	-	✓	-
Straight Knees Detection 4.6	-	-	✓	-

Table 4.1: Summary of all algorithms, best parameter settings, and detection accuracies. (✓= needed, - = not verified)

The turn/edge algorithm detects with a respective accuracy the most of the turns and the riding edge of the snowboarder. The weight distribution algorithm should have similar accuracy than the turn/edge detection, because the functioning of both is equal. To determine the accuracy of the straight knees algorithm, we should find other ex-

perimental methods. One solution could be to attached or marked the outer side of the legs with stripes that could indicate knee bending on the video footage. To recognize wrong weight distribution on the video footage, such stripes could be attached on the front and the back of the legs. Although we dismissed the calibration-free turn detection algorithm as a result of lower accuracy than the turn/edge detection, this algorithm shows that achieving high detection accuracy without calibration is possible.

Chapter 5

Final Implementation: The First Snowboard Assistant

"Theory is knowledge that does not work. Practice is when everything works and you do not know why."

—Hermann Hesse

After developing and evaluating the detection algorithm, we started to implement the first version of the wearable *Snowboard Assistant* proposed by Guggenmos [2007]. Since most human movements are below 18 Hz and even below 10 Hz in quite standing (Mathie et al. [2004], Brunelli et al. [2006]), we chose a sampling rate of 20 - 25 Hz for mistake detection.

5.1 First Version of a Wearable Snowboard Assistant

Beginners should practice only one exercise at once

Interviews with snowboard instructors yielded that beginners should practice only one exercise at once. Receiving too much feedback, while preforming more exercises at

one time, might lead to confusion.

Inspired by prior work, we designed our software in such a way, that users are able to train or analyze one exercise at once. In addition, we wanted that users are able to change or analyzes the exercises at runtime without restarting the system.

Visual feedback support for the calibration

One problem, which could be classified out of the prior work was that the calibration often yielded in wrong calibration values. The reason for that was that snowboard beginners were not able to distribute their weight equally on the board during the calibration phase. Thick clothes, the snowboard shoes, and the unfamiliar situation on the snowboard made it difficult to stand properly in the basic stance during the calibration phase,. Additionally, the snowboarder received no feedback in order to correct his posture. Based on this problem, we implemented a function which provides visual feedback and supports the user to distribute his weight equally. Figure 5.1 displays visual calibration aid that is provided by our application.

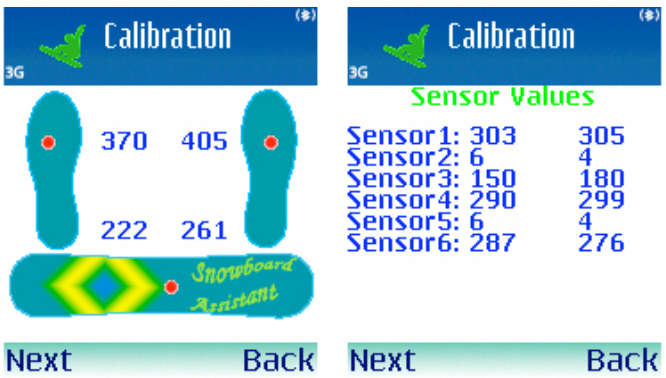

Figure 5.1: Snowboard Assistant: Calibration mode. The user can switch between (a) the visual weight distribution feedback and (b) the raw sensor values. The red circles indicate the weight distribution on the snowboard as well as the weight distribution for each foot.

After having calibrated all sensors, the user is able to choose among four different functionalities:

- Riding Edge Detection and Turn Counter
- Stop and Go Detection
- Weight Distribution
- Bending Knees

5.1.1 Turn/Edge Detection

Using the weight distribution algorithm to detect the riding edge

This function analyzes the descend of the snowboarder and counts the turns and the times of the different snowboard edges which the user needs to descend the slope. The core of this function is the front back weight distribution algorithm (3.3.2), which is used to determine the riding edge. Figure 5.2 shows the display output of the algorithm. Before descending the slope, the snowboarder only needs

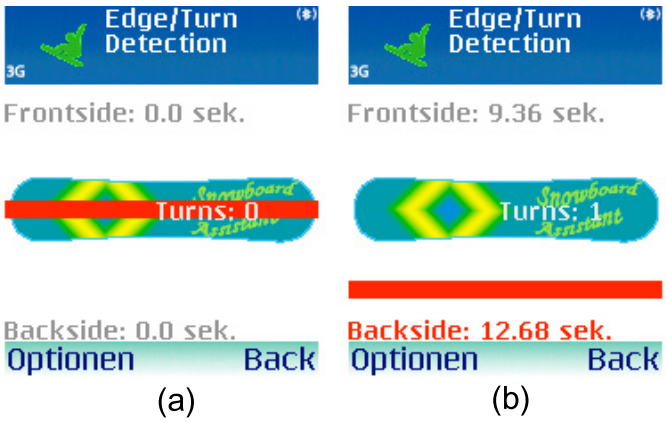

Figure 5.2: Snowboard Assistant: Edge Detection and Turn Counter function. It shows the starting screen with reseted values (a) and the visual output of the application during the run (b). The thick red line indicates the riding edge. The turn counter is shown on the snowboard.

Simple usage of the function

to press the start button. After he arrives at the end of the slope, the used can stop and analyze his ride by reading the information on the mobile phones display. If the user wants to repeat this performance again, he only needs to press the start button again. Thus, the old results will be deleted and the application restart the analysis of the ride.

The default settings for this algorithms are the parameter settings that achieved the highest accuracy during our evaluation. Nevertheless, the user can modify all predefined algorithm parameters to adapt the algorithm more to his own needs.

5.1.2 Stop and Go Detection

Using the simple activity recognition to detect movements

This function calculates the riding time and the rest period of the rider, while descending the slope. Additionally, the user can verify how often he took a rest. Basically, the function uses the simple activity recognition algorithm (3.3.5) to detect movements of the snowboarder. The functionality is similar to the edge detection, and allows the user to perform more exercises in a row. Additionally, the user can modify predefined algorithm parameters to adapt the algorithm more to his own needs. Figure 5.3 shows the display output of the function.

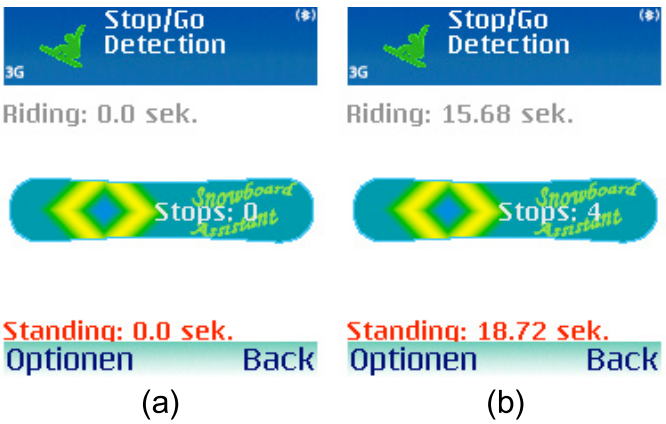

Figure 5.3: Snowboard Assistant: Stop/Go Detection. It shows the starting screen with reseted values (a) and the visual output of the application during the run (b). In addition to both timers, the counter is displayed on the snowboard that indicates the amount of stops during the ride.

5.1.3 Weight Distribution

Provide tactile feedback on wrong weight distribution

This function uses haptic feedback to inform the user about wrong weight distribution. Therefore, we used the left right distribution algorithm (3.3.2), to determine the weight distribution on both feet. Again, the handling of this function is conform to the other functions.

Since this function supports the user with tactile feedback, we provided no visual information on the phone display. This function triggers on wrong movements specific vibration motors that the snowboarder can chose before each performance. Similar to the other functions, the user can also modify each default parameter.

5.1.4 Bending Knees

Provide tactile feedback on straight knees

The last function analyzes knee bending and uses tactile feedback to inform the user, when he does not sufficiently bend his knees. Similar to the weight distribution function, we provided no visual output and the user can also modify each parameter between the exercises.

A small sample run of this application is shown in the appendix of the thesis (Appendix B).

5.2 Implementation Challenges

Challenges of application programming for mobile devices

In contrast to desktop computers, implementing such applications for mobile devices with low computational power is quite different. We realized, that we quickly reached the limits of the mobile phone as well as the programming language Java ME.

While developing our algorithms, we tested them on the mobile phone at the same time. Therefore, we implemented a test application, which helps us to identify the challenges of programming such mobile applications in Java ME:

- **Timer Accuracy -** Time is an important aspect in mistake detection applications, which is essential for synchronized working in milliseconds intervals. However, the timer of Java ME is not so accurate and tends to slight variances. Based on the fact that we only used one *SensAct* box, we observed no significant influence of the algorithms' output. This might change, when streaming from more then one box. Here, it is necessary to synchronize the input streams in order to detect mistakes accurately.

- **Time-consuming Operations -** Before implementing applications that work in milliseconds intervals, it is important to know, which operations of the application consume the most time. We identified several time-consuming and computational intensive operations, such as object instantiating or several drawing operations. Based on this knowledge, we could avoid or relocate such events to parts of the program that are not so much time depended.

- **Garbage Collector -** In contrast to C/C++, Java uses the garbage collector for deallocating unused objects and variables. This automatic memory management system is controlled by the Java Virtual Machine, which can influence the algorithms' run. Although Java ME offers no possibility to prevent the garbage collector, we could anyway influence the garbage collector schedule. Java ME offers a function that the allows the user to recommend a garbage collector activation. We used this function in parts of the program that are not so depend on time.

- **Java Predefined Functions -** Java offers lots of predefined function that can be used to implement different applications. Working in milliseconds intervals does not allow to use each of this function. For instance, to gain sensor values as integers out of the input stream, we realized that the predefined Java function is not suitable. Therefore, we have to find and implemented a workaround for this problem.

- **Java Virtual Machine** Java programs were compiled into bytecode that is interpreted by the Java Virtual Machine. This leads to the fact that Java programs has

a poor performance in contrast to applications written in programming languages such as C. Thus, our applications became unstable and crashed while streaming with more then 30 Hz. The reason for that was often an overflow in the input stream buffer. Additionally we observed that while increasing the amount of connected *SensAct* boxes, the sampling rate that allows stable functionality decreases.

Chapter 6

Summary and Future Work

"The future influences the present just as much as the past."

—Friedrich Nietzsche

In this chapter we want to summarize the results of our work and focus on identified and solved problems as well as on open challenges that should be solved in the near future.

6.1 Summary and Contributions

In this thesis we developed a mobile sensor/actuator platform that is usable in different application domains. Therefore, we developed three different versions of the *SensAct box* and four different detection algorithms. We developed a software library that allows easy and rapid development of mobile phone applications for our systems. Finally, we evaluated our hardware platform in the domain of snowboarding and conducted a user study to determine the accuracy of our detection algorithms. Considering the requirements for the hardware and software (1.1—"Goals and Requirements") we reached the following results:

Hardware Setup We developed a mobile and robust hardware box that allows easy connection of sensors and actuators. After evaluating the box, we identified problems and build an improved version.
Finally, we build four *SensAct boxes*:

- two of the first version boxes
- one first improved version box
- one final version box

Thus, we established a small set of hardware boxes that can be used for prototype development.

Software Implementation We implemented different detection algorithms that recognize specific gestures and postures. In addition, the structure of the implementation allows to combine more of these algorithms in order to detect more complex movements.
We implemented a software library in Java ME that consists of useful procedures, functions and algorithms. This library eases the development and increase the development pace of new prototypes. Finally, to help the designer with the prototype development, we implemented two supporting mobile phone applications (3.3.7 , 3.3.7)that offer different functions for sensors and actuators.

Snowboarding Domain We evaluated our system in the snowboarding domain and investigated most of our library algorithms. Additionally, we implemented an algorithm only for snowboarding, which shows the applicability of our system besides the library algorithms. To evaluate our algorithms we recorded sensor data of eight subjects that we later used to determine the detection accuracy, by using *iSense*.

Final Implementation To complete our thesis, we developed the first version of a *Snowboard Assistant*. Therefore, we considered the results that were investigated by Guggenmos and implementing a supporting feature for calibrate the sensors. Based on the conducted interviews with snowboarders, the application offers only one functionality at once in order to not confuse the user with too much feedback at once.

This application detects two common snowboard beginner mistakes and provides tactile feedback. Furthermore, the application analyses the descend of a snowboarder by counting turns or the riding time.

6.2 Future Work

Although our work can be already used as a toolkit for developing mobile wearable prototypes, there are still lots of challenges that should be solved in the near future.

Hardware Setup The *SensAct box* was sufficient for evaluation in the snowboarding domain. The system was not obtrusive and did not disturb the snowboarders during the descend. Nevertheless, while using the system, users must wear a bum bag. Additionally, mounting the sensors and actuators with subsequent calibration often takes over half an hour. Basically, the most common problem was the displacement of the sensors while dressing the snowboarder.
One solution would be to use LilyPad Arduinos (Buechley and Eisenberg [2008]) and integrated the hardware completely into clothes with snap buttons as connectors for sensors and actuators similar as used in the project EduWear (2.4.5).

Upper Body Posture To detect the back posture, we simply used an accelerometer and measured the smoothed value of one accelerometer axis. This can be used to detect static postures. Once the user started to move, the algorithm would calculate wrong results. The reason for that is the signal noise produced by the accelerometer value amplitude caused by the user's movements. Therefore, this algorithm cannot be used to detect wrong upper body postures while snowboarding. One solution is to used a similar system to detect upper body postures as proposed by Mattmann et al. [2007]. This system uses novel strain-sensitive sensors (Silveira et al. [2006]) and distinguishes between 27 different upper body postures.

Set of Sensors/Actuators In this thesis we attached connectors to a set of different sensors such as accelerometers, force sensors, and bend sensors and allowed an easy way to connect them to the *SensAct box*. Besides the sensors, our platform offered different actuators such as LEDs and vibration motors. Additionally, the user can use the mobile phone speaker and display as actuator.
To establish the creation of more complex prototypes, the system should provide a higher variety of different sensors and actuators.

Evaluation Methods We were not able to evaluate with our methods the detection accuracy of the straight knee detection algorithm (3.3.3)as well as the weight distribution algorithm 3.3.2). Therefore, we must investigate new evaluation methods to proof all the algorithms that are not directly verifiable by using video footage. These could be a visual aid on the clothes of the snowboarder, such as stripes on the legs, that makes it possible to identify bend knees or the weight distribution on the video footage.

Feedback Patterns The feedback pattern library contains only simple tactile feedback patterns such as simple ON/OFF or a sinusoidal PWM output. The next step is to investigate different feedback patterns that users can intuitively understand. Finally, these intuitive patterns can be used to extend the pattern library.

Detection Algorithms The algorithm library consists of four different algorithms, but we demonstrated that it is easy to create own algorithms (4.3). However, the library should be extended to provide a higher variety of algorithms that can be used for prototyping. The new algorithm should follow the structure of the existing ones in order to be usable in combination with other algorithms.

Calibration Problem One drawback of our algorithms is the calibration phase before they started to work. Most of our algorithms need accurate calibration values in order to detect mistakes with a high accuracy. Based on the fact that the calibration process is error-prone and sometimes results in wrong calibration val-

ues, using calibration-free algorithms would be one solution in order to keep the mistake detection accuracy on a high level. In the thesis we developed a calibration-free detection algorithm that recognizes turns with an adequate accuracy. We observed that it is possible to work without calibrating the sensors. Thus, calibration-free algorithms could overcome the calibration problem. However, the algorithm that we developed is too domain specific and we do not know whether other algorithms exists that can be generalizable on different application domains.

Haptic Communication The connection library offers only the possibility to connect one mobile phone with up to seven *SensAct boxes*. However, the system does not allow connections between two mobile phones. By providing the possibility of connection between two phones, we could establish a communication protocol between two sensor / actuator platforms. Since there a many researches on haptic interpersonal communication (Brave and Dahley [1997], Rovers and van Essen [2004], Chang et al. [2002], the system could be also used as a toolkit for wireless haptic interpersonal communication prototypes.

Algorithm Complexity Due to the fact that mobile phones become more and more powerful, we should focus on more complex algorithms. Complex algorithms are able to distinguish between different activities without changing the hardware configuration (Ravi et al. [2005]). Thus, the system would allow to create more complex prototypes by increasing the amount of basic movements that can be detected.

Finally, after realizing all these challenges the last question would be how much designers can benefit from this toolkit while developing new wearable computing systems.

Appendix A

Software Library Documentation

- mcg.arduino.io
  - mcg.arduino.io.ArduinoComm()
    - public void connectToArduino(String address);
    - public void sendToArduino(String text);
    - public void cancel();
    - public boolean isConnected();
    - public byte receiveFromArduino();
    - public void startStreaming();
    - public void stopStreaming();
    - public void calibrateSnapshot();
    - public void calibrateAverage();
    - public void startMotorDigital(int num);
    - public void stopMotorDigital(int num);
    - public void startMotorAnalog(int num, int val);
    - public void stopMotorAnalog(int num);
    - public void skipBytes(int num);
    - public void readSensorValues(int[] sensorValues);

- mcg.arduino.move
  - mcg.arduino.move.BendBack(int windowSize, int threshold, int connector, int outputFlagPos);
    - public void calculate(int[] sensorValues, int[] flagArray);
  - mcg.arduino.move.BendKnee(int windowSize, int threshold, int connector, int outputFlagPos);
    - public void calculate(int[] sensorValues, int[] flagArray);
  - mcg.arduino.move.FrontBackWeightDistribution (int windowSize, int threshold, int connectorLeft, int connectorRight, int outputFlagPos);
    - public void calculate(int[] sensorValues, int[] flagArray);
  - mcg.arduino.LeftRightWeightDistribution (int windowSize, int threshold, int connectorLeft, int connectorRight, int outputFlagPos);
    - public void calculate(int[] sensorValues, int[] flagArray);
  - mcg.arduino.StopGoDetection (int windowSize, int threshold, int connector, int outputFlagPos);
    - public void calculate(int[] sensorValues, int[] flagArray);

- `mcg.arduino.pattern`
  - `mcg.arduino.pattern.FeedbackPattern (ArduinoComm aComm, int motorNum, int patternNum);`
    - `public void run();`

Mobile phones must support MIDP 2.0 and CLDC 1.1 to run applications that use this software library.

Appendix B

First Snowboard Assistant: Sample run

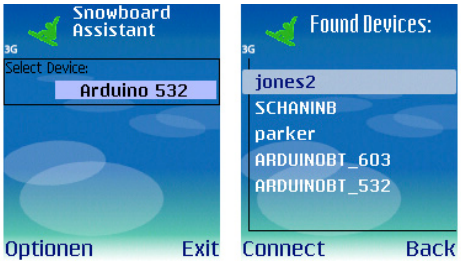

Figure B.1: Before starting to work, the user can choose among a set of Arduinos to establish a connection with a *SensAct box*. Besides the predefined set of *SensAct boxes*, the user can search for new Bluetooth devices. By pressing the *Connect* button, the application goes to the next step.

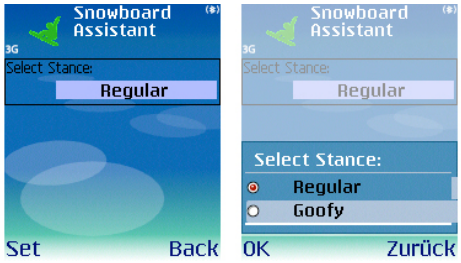

Figure B.2: After connecting to a *SensAct box* the user needs to specify his stance on the snowboard. This is important for the weight distribution algorithm as well as for the edge / turn detection. By pressing the *Set* button, the application goes one step and allows to calibrate the sensors.

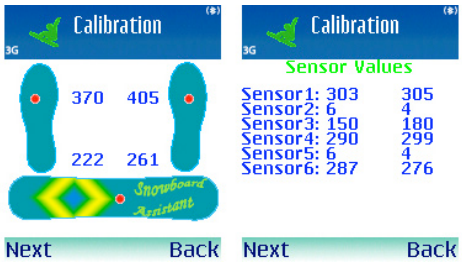

Figure B.3: After selecting a stance, the user needs to calibrate the sensors. Therefore, he can switch between the visual help screen that displays the weight distribution or the raw sensor values.

Figure B.4: After calibrating the sensors, the user can choose between the four functions. By following the interviews with snowboard instructors, only one function can be used at once.

Figure B.5: The Edge / Turn detection displays the current riding edge as well as the riding time. Furthermore, the user can adjust the algorithm parameters before starting with an exercise.

Figure B.6: The Stop / Go detection displays the riding time as well the standing time. Furthermore, the user can adjust the algorithm parameters before starting with an exercise.

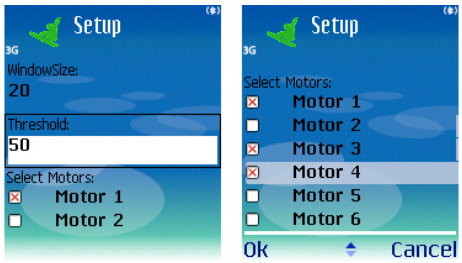

Figure B.7: Although the weight distribution provides no visual output, the user can adjust all algorithm parameters and specify multiple actuators for the tactile feedback.

Figure B.8: The bend knee detection offers the possibility to adjust all algorithm parameters. In addition to the parameters, the user can specify multiple actuators for each leg.

Appendix E

Smoothing filters

E.1 Simple Moving Average (SMA)

The Simple Moving Average (*SMA*) works similar to a first-order low-pass filter and reduces signals with high frequencies. In other words, *SMA* reduces the sensor noise and smooths the sensor curve.

SMA simply calculates the arithmetic mean of the previous n sensor values $val(s)$. Therefore, the SMA value $SMA(s)$ at time t is

$$\begin{aligned} SMA(s)_t &= \frac{val(s)_t + val(s)_{t-1} \cdots + val(s)_{t-n+1}}{n} \\ &= \frac{\sum_{i=0}^{i=n-1} val(s)_{t-i}}{n} \end{aligned}$$

The size of the n previous data samples describes the degree of the filtering.

E.2 Exponential Moving Average (EMA)

The Exponential Moving Average (*EMA*) also works similar to a first-order low-pass filter. The *EMA* is calculated by using the prior *EMA* value and the actual sensor value $val(s)$ at time t.

$$EMA(s)_t = \alpha * EMA(s)_{t-1} + (1 - \alpha) * val(s)_t$$

, whereas $\alpha \in [0; 1]$ and $EMA(s)_0 = val(s)_0$

The smoothing factor $alpha$ describes the degree of the filtering. When using a high $alpha$ value, the previous data sample $EMA(s)_{t-1}$ becomes more relevant. In contrast a low $alpha$ factor emphasizes the actual value $val(s)_t$.

Bibliography

Gregory D. Abowd, Liviu Iftode, and Helena Mitchell. Guest editors' introduction: The smart phone–a first platform for pervasive computing. *IEEE Pervasive Computing*, 4(2):18–19, 2005. ISSN 1536-1268. doi: http://dx.doi.org/10.1109/MPRV.2005.32.

Jonghun Baek, Geehyuk Lee, Wonbae Park, and Byoung-Ju Yun. Accelerometer signal processing for user activity detection. In *KES*, pages 610–617, 2004.

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

Michael Barr. Introduction to pulse width modulation. *Embedded Systems Programming*, pages 103–104, 2001.

Scott Brave and Andrew Dahley. intouch: a medium for haptic interpersonal communication. In *CHI '97: CHI '97 extended abstracts on Human factors in computing systems*, pages 363–364, New York, NY, USA, 1997. ACM. ISBN 0-89791-926-2. doi: http://doi.acm.org/10.1145/1120212.1120435.

Davide Brunelli, Elisabetta Farella, Laura Rocchi, Marco Dozza, Lorenzo Chiari, and Luca Benini. Bio-feedback system for rehabilitation based on a wireless body area network. In *PERCOMW '06: Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops*, page 527, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2520-2. doi: http://dx.doi.org/10.1109/PERCOMW.2006.27.

Leah Buechley. A construction kit for electronic textiles. In *ISWC*, pages 83–90, Montreux, Switzerland, 2006.

Leah Buechley and Michael Eisenberg. The lilypad arduino: Toward wearable engineering for everyone. *IEEE Pervasive Computing*, 7(2):12–15, 2008. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2008.38.

S. Cardin, F. Vexo, and D. Thalmann. Wearable System for Mobility Improvement of Visually Impaired People. *Visual computer journal*, vol. 23:p 109 – 118, 2006. doi: NA.

Angela Chang, Sile O'Modhrain, Rob Jacob, Eric Gunther, and Hiroshi Ishii. Comtouch: design of a vibrotactile communication device. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 312–320, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. doi: http://doi.acm.org/10.1145/778712.778755.

Tanzeem Choudhury, Gaetano Borriello, Sunny Consolvo, Dirk Hähnel, Beverly Harrison, Bruce Hemingway, Jeffrey Hightower, Predrag V. Klasnja, Karl Koscher, Anthony LaMarca, James A. Landay, Louis LeGrand, Jonathan Lester, Ali Rahimi, Adam Rea, and Danny Wyatt. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2):32–41, 2008.

Lucy Dunne, Pauline Walsh, Barry Smyth, and Brian Caulfield. Design and evaluation of a wearable optical sensor for monitoring seated spinal posture. *iswc*, 0:65–68, 2006. ISSN 1550-4816. doi: http://doi.ieeecomputersociety.org/10.1109/ISWC.2006.286345.

Lucy Dunne, Pauline Walsh, Barry Smyth, and Brian Caulfield. A system for wearable monitoring of seated posture in computer users. *4th International Workshop on Wearable and Implantable Body Sensor Networks*, pages 203–207, 2007.

Christian Guggenmos. Towards a wearable snowboarding assistant. Master's thesis, RWTH Aachen University, Aachen, Germany, December 2007.

G. A. Hansson, P. Asterland, N. G. Holmer, and S. Skerfving. Validity and reliability of triaxial accelerometers for inclinometry in posture analysis. *V. Med Biol Eng Comput*, 39(4):405ñ–413, 2001.

Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–154, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: http://doi.acm.org/10.1145/1240624.1240646.

James F. Knight, Huw W. Bristow, Stamatina Anastopoulou, Chris Baber, Anthony Schwirtz, and Theodoros N. Arvanitis. Uses of accelerometer data collected from a wearable system. *Personal Ubiquitous Comput.*, 11(2):117–132, 2007. ISSN 1617-4909. doi: http://dx.doi.org/10.1007/s00779-006-0070-y.

K.S.C. Kuang, W.J. Cantwell, and P.J. Scully. An evaluation of a novel plastic optical fibre sensor for axial strain and bend measurements. *Measurement Science and Technology*, 13:1523–1534, 2002.

Kai Kunze, Michael Barry, Ernst A. Heinz, Paul Lukowicz, Dennis Majoe, and Jeurg Gutknecht. Towards recognizing tai chi - an initial experiment. In *Proceedings of the Third International Forum on Applied Wearable Computing*, Berlin, Germany, April 2006.

R. Lindeman and J. Cutler. Controller design for a wearable, near-field haptic display, 2003. URL `citeseer.ist.psu.edu/lindeman03controller.html`.

J. Mantyjarvi, J. Himberg, and T. Seppanen. Recognizing human motion with multiple acceleration sensors. volume 2, pages 747–752 vol.2, 2001. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=973004`.

Merryn J. Mathie, Adelle C. F. Coster, Nigel H. Lovell, and Branko G. Celler. Accelerometry: providing an integrated, practical method for long-term, ambulatory monitoring of human movement. *Physiological Measurement*, 25:R1–R20(1), 2004.

Corinne Mattmann, Oliver Amft, Holger Harms, Gerhard Tröster, and Frank Clemens. Recognizing upper body postures using textile strain sensors. In *ISWC 2007: Proceedings of the 11th IEEE International Symposium on Wearable Computers*, pages 29–36. IEEE Press, October 2007.

Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7(1): 3–28, 2000. URL `citeseer.ist.psu.edu/article/myers00past.html`.

Nuria Oliver and Fernando Flores-Mangas. *HealthGear: A Real-time Wearable System for Monitoring and Analyzing Physiological Signals*. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2547-4. doi: http://dx.doi.org/10.1109/BSN.2006.27.

Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 1541–1546. AAAI Press / The MIT Press, 2005. URL `http://dblp.uni-trier.de/db/conf/aaai/aaai2005.html#RaviDML05`.

A.F. Rovers and H.A. van Essen. Him: a framework for haptic instant messaging. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1313–1316, New York, NY, USA, 2004. ACM. ISBN 1-58113-703-6. doi: http://doi.acm.org/10.1145/985921.986052.

Michael Schrage. Cultures of prototyping. pages 191–213, 1996. doi: http://doi.acm.org/10.1145/229868.230045.

I. Silveira, F. Clemens, C. Bergmann, and T. Graule. Novel strain-sensitive sensors for application in human physiology. *NanoEurope*, pages 12 –14, September 2006.

Emmanuel Munguia Tapia, Stephen S. Intille, Louis Lopez, and Kent Larson. The design of a portable kit of wireless sensors for naturalistic data collection. In *Pervasive*, pages 117–134, 2006.

Jhun-Ying Yang, Yen-Ping Chen, Gwo-Yun Lee, Shun-Nan Liou, and Jeen-Shing Wang. Activity recognition using one triaxial accelerometer: A neuro-fuzzy classifier with feature reduction. In Lizhuang Ma, Matthias Rauterberg, and Ryohei Nakatsu, editors, *ICEC*, volume 4740 of *Lecture Notes in Computer Science*, pages 395–400. Springer, 2007. ISBN 978-3-540-74872-4. URL `http://dblp.uni-trier.de/db/conf/iwec/icec2007.html#YangCLLW07`.

Index

Typeset May 24, 2008