LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Sensor-Based User Interface Concepts for Continuous, Around-Device and Gestural Interaction on Mobile Devices

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik der
Ludwig-Maximilians-Universität München

vorgelegt von Diplom-Informatiker

Sven Kratz

München, den 06.06.2012

Erstgutachter: Prof. Dr. Michael Rohs
Zweitgutachter: Prof. Dr. Roderick Murray-Smith

Tag der mündlichen Prüfung: 10. September 2012

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Kratz, Sven

Name, Vorname

München, 06.06.2012

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2

Abstract

A generally observable trend of the past 10 years is that the amount of sensors embedded in mobile devices such as smart phones and tablets is rising steadily. Arguably, the available sensors are mostly underutilized by existing mobile user interfaces. In this dissertation, we explore sensor-based user interface concepts for mobile devices with the goal of making better use of the available sensing capabilities on mobile devices as well as gaining insights on the types of sensor technologies that could be added to future mobile devices. We are particularly interested how novel sensor technologies could be used to implement novel and engaging mobile user interface concepts.

We explore three particular areas of interest for research into sensor-based user interface concepts for mobile devices: *continuous interaction, around-device interaction* and *motion gestures*.

For continuous interaction, we explore the use of dynamic state-space systems to implement user interfaces based on a constant sensor data stream. In particular, we examine zoom automation in tilt-based map scrolling interfaces. We show that although fully automatic zooming is desirable in certain situations, adding a manual override capability of the zoom level (*Semi-Automatic Zooming*) will increase the usability of such a system, as shown through a decrease in task completion times and improved user ratings of user study. The presented work on continuous interaction also highlights how the sensors embedded in current mobile devices can be used to support complex interaction tasks.

We go on to introduce the concept of *Around-Device Interaction* (ADI). By extending the interactive area of the mobile device to its entire surface and the physical volume surrounding it we aim to show how the expressivity and possibilities of mobile input can be improved this way. We derive a design space for ADI and evaluate three prototypes in this context. *HoverFlow* is a prototype allowing coarse hand gesture recognition around a mobile device using only a simple set of sensors. *PalmSpace* a prototype exploring the use of depth cameras on mobile devices to track the user's hands in direct manipulation interfaces through spatial gestures. Lastly, the *iPhone Sandwich* is a prototype supporting dual-sided pressure-sensitive multi-touch interaction. Through the results of user studies, we show that ADI can lead to improved usability for mobile user interfaces. Furthermore, the work on ADI contributes suggestions for the types

of sensors could be incorporated in future mobile devices to expand the input capabilities of those devices.

In order to broaden the scope of uses for mobile accelerometer and gyroscope data, we conducted research on motion gesture recognition. With the aim of supporting practitioners and researchers in integrating motion gestures into their user interfaces at early development stages, we developed two motion gesture recognition algorithms, the *$3 Gesture Recognizer* and *Protractor 3D* that are easy to incorporate into existing projects, have good recognition rates and require a low amount of training data. To exemplify an application area for motion gestures, we present the results of a study on the feasibility and usability of *gesture-based authentication*.

With the goal of making it easier to connect meaningful functionality with gesture-based input, we developed *Mayhem*, a graphical end-user programming tool for users without prior programming skills. Mayhem can be used to for rapid prototyping of mobile gestural user interfaces.

The main contribution of this dissertation is the development of a number of novel user interface concepts for sensor-based interaction. They will help developers of mobile user interfaces make better use of the existing sensory capabilities of mobile devices. Furthermore, manufacturers of mobile device hardware obtain suggestions for the types of novel sensor technologies that are needed in order to expand the input capabilities of mobile devices. This allows the implementation of future mobile user interfaces with increased input capabilities, more expressiveness and improved usability.

Zusammenfassung

Ein allgemein zu beobachtender Trend in den letzten 10 Jahren zeigt, dass neue Generationen von mobilen Geräten wie Smartphones und Tablets mit einer stetig wachsenden Anzahl von Sensoren ausgestattet werden. Es ist anzunehmen, dass die Möglichkeiten der vorhandenen Sensoren meist nicht vollständig von aktuellen mobilen Benutzerschnittstellen ausgeschöpft werden. In dieser Dissertation erforschen wir sensorbasierte Konzepte für mobile Benutzungsschnittstellen mit dem Ziel, sowohl die derzeit vorhandenen Sensoren auf mobilen Geräten besser auszunutzen, als auch Erkenntnisse über die Arten von Sensortechnologien zu bekommen, mit der zukünftige mobile Geräte ausgestattet werden könnten. Wir interessieren uns besonders dafür, wie neuartige Sensortechnologien verwendet werden können, um neue und fesselnde Interaktionskonzepte für mobile Geräte zu entwickeln.

Wir behandeln folgende Themenbereiche, die von besonderem Interesse sind: *kontinuierliche Interaktion*, *Interaktion in der Umgebung* des mobilen Gerätes („Around-Device Interaction“) und *Bewegungsgesten*.

Bezüglich kontinuierlicher Interaktion erforschen wir die Verwendung von dynamischen Zustandssystemen, um Benutzungsschnittstellen zu implementieren, die auf einem konstantem Datenstrom eines oder mehrerer Sensoren beruhen. Insbesondere untersuchen wir Zoomautomatisierung in neigungsbasierten Benutzungsschnittstellen für die Kartennavigation auf mobilen Geräten. Wir zeigen, dass trotz der Vorteile von vollautomatischem Zoomen in gewissen Situationen, die Bereitstellung einer manuellen Eingriffsmöglichkeit der Zoomstufe („Semi-Automatic Zooming“) die Usability eines solches Systems erhöht. Dies belegen wir anhand der Ergebnisse einer Nutzerstudie, die einen Rückgang der Bearbeitungszeiten sowie verbesserte Bewertungen aufzeigen. Die präsentierten Beiträge im Bereich kontinuierlicher Interaktion zeigen wie die eingebetteten Sensoren in heutigen mobilen Geräten dazu verwendet werden können um komplexe Interaktionen zu unterstützen.

Wir befassen uns anschließend das Konzept der Interaktion in der Umgebung des Gerätes—„Around-Device Interaction “ (ADI). Durch die Erweiterung des Interaktionsbereiches des mobilen Gerätes auf dessen gesamte Oberfläche sowie dessen umgebenden physischen Raumes wollen wir zeigen, wie die Ausdruckskraft und Möglichkeiten von mobilen Eingabemethoden

verbessert werden können. Wir leiten einen Designraum für ADI ab und evaluieren in diesem Kontext drei Prototypen. *HoverFlow* ist ein Prototyp, der es unter Verwendung einer geringer Anzahl von einfachen Sensoren erlaubt, grobe Handgesten in der Umgebung eines mobilen Gerätes zu erkennen. *PalmSpace* ist ein Prototyp der dazu dient, die Verwendung von Tiefenkameras auf mobilen Geräten zu erforschen, indem die Hände des Benutzers zwecks Entwicklung von mobilen Benutzungsschnittstellen zur direkten Manipulation mittels räumlicher Handgesten erfasst werden. Das *iPhone Sandwich* ist ein Prototyp eines mobilen Gerätes, das beidseitige druckabhängige multi-touch Eingabe unterstützt. Durch die Präsentation von Ergebnissen von Nutzerstudien zeigen wir, dass ADI zu einer verbesserten Usability von mobilen Benutzungsschnittstellen führt. Des Weiteren führt unsere Arbeit im Bereich ADI zu neuen Erkenntnissen über die Arten von Sensoren, die in zukünftigen mobilen Geräten verbaut werden könnten und die neuen Eingabetechniken welche diese ermöglichen würden.

Um die Verwendungsmöglichkeiten von Beschleunigungs- und Gyroskopdaten von mobilen Geräten zu erweitern, haben wir die Erkennung von Bewegungsgesten erforscht. Das Ziel war, Praktiker und Forscher dabei zu unterstützen, Bewegungsgesten in möglichst frühen Entwicklungsstadien in deren Benutzungsschnittstellen integrieren zu können. Dazu haben wir zwei Erkennungsalgorithmen für Bewegungsgesten, *$3 Gesture Recognizer* und *Protractor 3D*, entwickelt, die einfach in bestehende Projekte zu integrieren sind, eine gute Erkennungsrate haben und nur eine geringe Anzahl an Trainingsdaten benötigen. Um ein mögliches Anwendungsgebiet von Bewegungsgesten zu untersuchen, haben wir uns mit gestengestützter Authentisierung befasst. Wir präsentieren die Ergebnisse einer Nutzerstudie, welche die Machbarkeit und Benutzbarkeit dieses Authentisierungskonzepts untersucht.

Mit dem Ziel, die Verknüpfung von Gesteneingaben mit bedeutender Ausgabefunktionalität zu ermöglichen, entwickelten wir *Mayhem*, ein grafisches Programmierwerkzeug für Endbenutzer, die keine vorherige Programmiererfahrung haben. Mayhem kann für schnelles Prototyping von gestenbasierten Benutzungsschnittstellen verwendet werden.

Der Hauptbeitrag dieser Dissertation ist die Entwicklung von mehreren neuartigen Konzepten für mobile sensorbasierte Benutzungsschnittstellen. Entwicklern von mobilen Benutzungsschnittstellen soll somit geholfen werden, die sensorischen Fähigkeiten von mobilen Geräten besser im Sinne der Benutzungsschnittstelle auszunutzen. Desweiteren bekommen Hersteller von mobilen Geräten Anregungen darüber, welche Arten von neuartigen Sensoren für die Eingabe auf mobilen Geräten verbaut werden können. Somit können zukünftige mobile Benutzungsschnittstellen mit neuartigen Eingabefähigkeiten, mehr Ausdruckskraft und eine höheren Usability implementiert werden.

Acknowledgements

First of all, I would like to deeply thank my advisor, Michael Rohs, for providing excellent guidance and placing his trust in me during the past four years. Michael was always enthusiastic about the projects I proposed and provided me with very valuable feedback and insights. I would also like to thank my secondary examiner, Roderick Murray-Smith for taking on this role. His systematic as well as technical approach to HCI has been a great inspiration to me.

I would like to thank Sebastian Moeller for on-boarding me at Deutsche Telekom Laboratories (now Telekom Innovation Laboratories), my first position in research, and supporting me throughout my years there. I would like to thank my colleagues at T-Labs for contributing to an awesome working atmosphere. Irene Huber for being a super-administrator that always kept things running smoothly at T-Labs. Georg Essl was a great collaborator and providing valuable advice. Jörg Müller was a welcome addition to our small HCI group at T-Labs, his sense of humor makes any situation bearable, and he has been a good friend and collaborator ever since he joined T-Labs. Niklas Kirschnick introduced me to the topic security and usability and it was great fun supervising theses together with him. I'd also like to thank my other colleagues at Telekom, Jens, Marcel, Michael N., Matthias G., Matthias R., Karim, Jan-Niklas, Matthias, Blazej, Tim, Ina and so many others that made the coffee breaks fun and the table soccer matches so challenging.

In the summers of 2010 and 2011, I completed two internships at the Microsoft Applied Sciences Group in Redmond, WA, USA. I would like to thank Paul Dietz for his excellent mentoring during my internship. Paul is an amazing person to be around, and not a day goes by without being able to learn something useful from him. I would also like to thank Steven Batiche for supporting us throughout the Mayhem project. I had great technical conversations with Matheen Siddiqui, Johnny Lee and Vivek Pradeep, and they really inspired me to learn more about Computer Vision. Cati Boulanger is one of the most creative persons I have worked with and I cannot wait to see her future projects. Jay Meistrich and Eli White were great collaborators and really taught me how to get the most out of Visual Studio. Lastly, I would like thank my awesome fellow interns, Albert, Chen, David and Jinha for making a great experience even better.

Following in the footsteps of my advisor, I moved to Munich in the beginning of 2011 and joined the Media Informatics Group. I would like to thank Andreas Butz and Heinrich Hussmann for adopting Michael and me into their group and providing such a great place to work and, above all, friendly work atmosphere. Franziska Schwamb is was helpful with the "red tape" of getting my position approved at the university and I would like to thank her for all her efforts. Rainer Fink keeps the Munich lab's IT infrastructure running smoothly and efficiently—he could always find a solution to my requests. Of course, I also met many more awesome colleagues here in Munich: Alex d. L., Alex W., Aurélien, Alina, Bettina, Doris, Emmanuel, Hendrik, Henri, Fabian, Max, Raphael, Sara and Sebastian. It has been a great time and I wish you all the best of success for the future.

In Berlin and Munich, I supervised a number of student theses. I would like to thank Kimmo Nurmisto, Daria Skripko, Ivo Brodien, Tilo Westermann, Dennis Guse, Benjamin Bähr, Valerie Kroner, Bernhard Slawik, Felix Reitberger and Jörg Moldenhauer for their hard work and their contributions to some of the papers we published together.

This whole endeavor would not have been possible without the support of my grandmother to whom I am eternally grateful.

Lastly, I would like to thank my loving wife for giving me much-needed moral support throughout the years.

Contents

List of Figures

List of Tables

Conventions

Throughout this dissertation we use the following conventions.

Definitions of technical terms or short excursus are set off in colored boxes.

Excursus:
Excursus are detailed discussions of a particular point in this dissertation, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

```
squares = [x**2 for x in range(100)]
```

Citations of published literature are given in parentheses.

(?)

Citations of online sources and websites are denoted with a W in superscript.

(openstreetmap.org, 2012)[W]

In addition, every web citation is adorned with the date of access and the subdirectory on the thesis DVD containing a mirror of the web site at the time of access, i.e, *Accessed: dd.mm.yy (url.foobar)*.

The entire dissertation is written in American English.

Chapter 1

Introduction

"The best scientist is open to experience and begins with romance—the idea that anything is possible."

—Ray Bradbury

hardware evolution has a significant influence on mobile user interfaces

In the past years, mobile devices, in particular smart phones and slate-sized devices, have undergone a rapid evolution. Improvements in CPU speed, memory capacity, screen resolution and sensory capabilities of the devices profoundly affect the development of mobile user interfaces. User interfaces for smart phones, for instance, have transformed from being mostly button-driven, as was the case just a few years ago, towards being mainly (multi-)touch-based today.

advanced sensors enable smarter and more effective user interfaces

At the same time, the extension of the sensory capabilities of mobile devices, such as incorporating GPS, accelerometers, gyroscopes, magnetometers and distance sensors, not only aids in creating the opportunities to implement a much wider variety of applications for mobile devices, but also enables the development of smarter and more effective mobile user interfaces than the ones available today.

a paradigm shift in mobile interaction is needed

Desktop computing is moving away from the traditional GUI model of interaction, towards natural user interfaces (NUI) (Wigdor and Wixon, 2011), where the underlying functions to be controlled are embedded in the actual physical appearance and behavior of the controlled UI elements. Looking at current mobile interfaces, this shift cannot be readily seen[1]. Although multi-touch displays and powerful sensors and processors are present on the latest mobile device, interacting with these doesn't fundamentally differ from interacting with the very first, mobile devices, for instance the IBM Simon (Section 1.1.4.1). For instance,

[1]We have discussed the possibilities for realizing NUI paradigms in mobile user interfaces in (Kratz et al., 2010b).

in most naïve touch-screen interfaces, physical buttons are directly replaced by virtual counterparts on the touch-screen.

This dissertation presents novel user interface technologies and paradigms for mobile user interfaces that leverage the advanced sensing technology and processing capabilities on modern mobile devices. By conducting user studies we try to show that these novel technologies can help the users complete their tasks more effectively and in a more engaging way than existing mobile user interfaces.

This dissertation addresses the aforementioned challenge by proposing new uses for existing mobile device sensors as well as evaluating previously unused sensor technologies for use in mobile interfaces. Before we discuss the research topics and contributions in this dissertation in more detail (Section 1.2), we need to summarize the evolution of mobile user interfaces and present a brief overview of the status quo of mobile interaction.

1.1 Milestones in the History of Mobile Interaction

mobile interaction research into new sensing technologies allows for improved input techniques

Mobile Interaction is a relatively young field within Computer Science. Due to small screen sizes, limited ways for the user to make inputs and because mobile devices are used on-the-go rather than in stationary settings, mobile user interfaces have especially high challenges to overcome to achieve good usability. One of these challenges is to develop and apply useful sensing technologies that allow the users to interact with their devices in ways that are both more enjoyable and expressive.

1.1.1 The Dynabook

miniaturization of computing devices begins in the 1970s

In the 1970s the focus of computing started shifting from mainframe systems to smaller computing systems. This marked the start of the micro computer revolution since the idea that having a computer "on one's desktop" was unforeseen at the time. Only few researchers considered even smaller, portable devices. Alan Kay was one of the first researchers who proposed a mobile computer, which he called the *Dynabook* (Kay, 1972b).

the dynabook: a universal tool for education

The Dynabook design was intended as a "global information utility" as well as an educational tool for "children of all ages". The Dynabook's size was about the size of a modern tablet PC. The design included a 512×512 pixel display supporting "dynamic graphics" with stylus input, audio recording and playback, a large local storage capacity (for the

Figure 1.1: Alan Kay's original DynaBook paper prototype. Adapted from original image by Flickr User Marcin Wichary CC PD.

time) and network connectivity. Kay discussed the option of providing either a physical keyboard (Figure 1.1) or letting the screen cover the entire surface of the device and using an on-screen keyboard for text entry.

It is interesting to note that it took almost 40 years until the first DynaBook-like devices, such as the Amazon Kindle or the Apple iPad became adopted widely.

1.1.2 Pads, Tabs and Live Boards

computers will disappear into the background

In his seminal work on Ubiquitous Computing (UbiComp), "The Computer for the 21st Century", Mark Weiser envisions a future where computing will disappear into the background (Weiser, 1991). His vision is that computing power will be very inexpensive and available everywhere, with "hundreds of devices in a single room".

Weiser accurately predicted the form factors of future mobile devices

The description of the devices, "badges", "pads", "tabs" and "live boards" that Weiser envisions will populate the world of UbiComp is of particular interest for mobile interaction, as most of these device classes have at present been fully commercially developed and are used in ways remarkably similar to what Weiser had envisioned in 1991.

Badge-type devices nowadays exist in a variety of forms, such as RFID[2]-enabled public transportation passes and identification documents where radio waves are used both as communication channel and

[2]radio frequency identification

as external power source, active price tags in supermarkets, or also interactive door panels. As Weiser envisioned, micro controllers have nowadays become so cheap that they can be incorporated in almost any device, with little or no size constraints. Special care, however, has to be taken when designing user interfaces for very small to minuscule devices. A recommended work on that topic is (Ni and Baudisch, 2009).

mobile phones and tabs are not yet the "scrap" devices Weiser had envisioned

Today's smart phones and tablets are a good approximation of Weiser's pads and tabs. One property of these devices, however, that Weiser (arguably) mis-predicted is that they aren't (yet) "scrap computers". Weiser may have underestimated the personal attachment users have to their mobile devices and also the mobile device's function as a status symbol. It may be more appealing to some people to own an expensive device than a whole bunch of "scrap" devices. Furthermore, mobile devices store most of the user's data locally, even though the level of connectivity to other devices and the internet matches if not surpasses the capabilities of devices envisioned in Weiser's article. With decreasing device costs and the shift of data storage and applications towards "Cloud Computing", Weiser's vision of throw-away, ubiquitous devices may yet become reality.

interactive surfaces are not yet ubiquitous

The final device class Weiser envisioned, live boards, have also appeared, albeit in the preferred form of interactive surfaces. While flat-screen displays may be considered ubiquitous in public spaces of developed countries, the majority of them are not interactive. Interactive surfaces, wall-mounted or in a tabletop form factor, are becoming increasingly popular but still are far from being Ubiquitous and are, for the most part, a novelty when seen or used by the general population.

1.1.3 The Apple Newton

The Newton was a *Personal Digital Assistant* (PDA) developed by Apple in 1993. Although similar devices had been developed before, the Newton (Figure 1.2) was the device for which the term PDA was actually coined. The device was designed for stylus input and had sophisticated handwriting recognition implemented using Artificial Neural Networks (Yaeger et al., 1998). This same handwriting recognition technology, originally called *Rosetta* has been incorporated in the later Apple operating system OS X as *Inkwell* (Yaeger, 1996)[W].

Although the Newton was generally successful and very popular with its users, it was cancelled as a product in 1997 by Steve Jobs, when he returned to Apple (Isaacson, 2011).

Mobile Interaction on Interactive Surfaces:
A niche topic in mobile interaction research that is worth noting is mobile interaction on interactive surfaces. The work in this area explores the questions of how mobile devices can be used in conjunction with a surface computer.
The advantage of the mobile device is its quasi-constant connectivity and its role as the user's personal data storage while on the go. Interactive surfaces can be used to provide rich input and output capabilities to the mobile device, allowing the phone to run more complex applications using the interactive surface as a proxy. A number of interesting papers have been written on the subject, covering, for instance:

- using the camera image of mobile devices for pointing tasks on remote displays (Boring et al., 2010).
- using mobile phones as primary input devices on light-weight, or "unobtrusive" interactive surfaces (Kratz and Rohs, 2009b).
- conducting collaborative sharing tasks using mobile devices on an interactive tabletop surface (Kray et al., 2008).
- detecting devices on interactive surfaces (Echtler, 2008).
- augmenting mobile devices with external multi-touch detection when placed on a flat surface (Butler et al., 2008b).

Figure 1.2: The Apple Newton Message Pad 100. Adapted from original image by Flickr user Bruno Cordioli CC BY.

1.1.4 Mobile Phones

4.6 Billion mobile phones are in use throughout the world (CBS News, 2010)[W], making them truly pervasive devices. Mobile phone usage has penetrated all levels of society. Even the poorest people in developing

the mobile phone can be considered the most widely used computer in the world

countries have access to mobile phones. Thus, the mobile phone can be considered as the most widely used computer in the world.

The mobile phone has not only revolutionized the way people communicate in developed countries, but has also fundamentally changed the way people communicate and live in developing countries.

Mobile user interfaces have undergone a significant development since the introduction of the first commercially available hand-held mobile phone, the *Motorola DynaTAC* (Wikipedia, 2012)[W] in 1983.

Figure 1.3: The IBM Simon in its charging cradle. The large touch-screen with on-screen keyboard for dialing is visible.

1.1.4.1 The IBM Simon – An Early Touch-Screen Device

the IBM Simon was the first touch-screen-only mobile phone

Up to the introduction of the IBM Simon in 1993, mobile phones were mainly button-based, and their functionality rarely extended beyond placing and receiving phone calls. The IBM Simon, however, changed this notion radically. It incorporated a touch screen, spanning almost the entire device's side (Figure 1.3). Interaction with the Simon was done entirely through the touch screens, it did not feature physical buttons for dialing. The Simon was also amongst the first phones to feature several different applications, i.e., for managing contacts, displaying a calendar, or even playing games. The device was also one of the first smart phones as it had features that went beyond telephony, such as the capability of sending faxes or emails and saving data on a PCMCIA[3] memory card.

[3]Personal Computer Memory Card International Association.

1.1.4.2 Phone Cameras

camera phones led to new services and have a large social impact

The *Kyocera VP-2010* (Wikipedia, 2012)[W], introduced in Japan in 1997 was the world's first camera phone. The camera on this phone was front-facing, as its intended use was for video telephony. Manufacturers soon also equipped phones with rear-facing cameras for picture taking and video recording. The introduction of cameras on mobile phones was the basis for creating new services such as the Multimedia Messaging Service (MMS), designed to facilitate the sending of images from one mobile device to another. Camera phones have so far had a profound social impact. They have turned picture and video recording into a commodity—in any given situation, images can be taken quickly and (when required) discretely using a mobile phone. Because they are nearly impossible to ban, camera phones have in recent times been media recording devices. Together with social media services such as Facebook or Twitter, camera phones enabled protesters and observers to circumvent censorship during times of social uprisings against oppressive regimes, as witnessed during the "Arab Spring" in 2011 (Goodman, 2011).

visual codes enable phones to retrieve data from the physical world

Cameras have also enabled mobile phones to retrieve data from the physical world. Small snippets of data, such as URLs, can be encoded in the form of visual codes. Image processing algorithms running on mobile phones can thus decode the image contained on visual codes using camera images of the codes as input. Popular visual code formats readable by mobile phones are, for instance, the QR and EAN/UPC Codes (ISO/IEC, 2000b,c). More recent applications such as *Google Goggles* (Neven Sr and Neven, 2009);(Google Inc., 2012b)[W], allow marker-less recognition of specific types of everyday objects, such as book covers or even hardware appliances. There is ongoing work to expand the functionality Google Goggles to recognize biological artifacts such as plant leaves (PCWorld, 2009)[W].

A further use of the phone's camera is for Augmented Reality (AR) applications. Here, marker-based (Wagner and Schmalstieg, 2003b) or marker-less tracking (Lee and Höllerer, 2007) can be used to render virtual objects in the camera viewfinder's image, making interaction in a combination of the real and virtual world possible. Commercial applications, such as *Layar* have popularized the concept of "Augmented Reality Browsing" (Layar, 2012)[W], although one could argue whether if such "AR-Browsers" are true AR applications.

motion information can be extracted from camera image sequences

The camera feeds of mobile phones can also be used for direct input to the user interface. When it is moved through the air, it is possible

calculate a phone's movement direction using optical flow. This movement information can be utilized for pointing tasks on external displays (Ballagas et al., 2005b; Rohs), or for gesture recognition (Kratz, 2007b).

1.1.4.3 Multi-touch and Advanced Sensing Capabilities: the iPhone

Apple's iPhone introduced a number of new technologies to smart phones, which have since become standard in current devices. The iPhone was the first device incorporating a capacitive multi-touch screen, making the existing touch-screen devices, which relied primarily on resistive touch screens and stylus input, obsolete. By combining a database containing the geographic coordinates of WLAN access points, cell tower triangulation and GPS[4] unit, the iPhone was capable of determining its geographic location very quickly with a high precision. This allowed the development of a multitude of well-functioning location-based applications for the iOS[5] platform. Although the iPhone was not the first device to incorporate an accelerometer [6], it was the first phone that used accelerometer data extensively throughout the UI, for instance to rotate the user interface orientation to match the device's pose.

Accelerometers allow mobile phone applications to become aware of how the mobile phone is being moved. This motion information is used a wide range of applications ranging from sleep activity tracking[7] to establishing a connection between two devices [8].

Although this dissertation does not cover audio as a UI input/output channel in great detail, interaction through audio input is also a very important domain of mobile interaction. The iPhone 4S introduces Siri (Gruber et al., 2011), a novel audio-based assistant. Siri is an online speech-recognition engine coupled to an artificial intelligence backend. Siri features a high degree of integration with the iPhone's user interface. As such, Siri can be used to access and control most of the functions of the mobile device, such as sending SMS or email messages. A voice-based assistance feature such as Siri is particularly useful in situations when the user can't use the touch-screen, e.g. when driving an automobile.

[4] Global Positioning System.
[5] iPhone Operating System.
[6] The Nokia 5500 Sport was on of the first phones to feature a built-in accelerometer (Nokia Inc., 2012)[W].
[7] *Sleep Cycle Alarm Clock* (Maciek Drejak Labs AB, 2012)[W].
[8] *Bump* (Wikipedia, 2012)[W].

1.1.4.4 Future Input and Output Technologies for Mobile Phones

In this chapter, we have discussed a number technologies for input as well as output on mobile phones. We can, nevertheless, only speculate what types of input and output technologies will be included in future mobile phones. What is clear is that future mobile phones will likely be equipped with increasingly powerful I/O capabilities, i.e. sensors and output devices that are not yet present on current devices.

Whereas we try to envision future input and output technologies for mobile devices, we cannot claim, nor aim to give a complete overview of technologies to come. As such, the technologies discussed in the following represent the author's personal thoughts on what lies ahead in the domain of I/O for mobile user interfaces.

Pressure Sensing In answer to the question how multi-touch screens can be extended in functionality, pressure is a reasonable answer. The Android SDK actually already supports the reporting of "pressure" values (Google Inc., 2012a)[W]. These values are, however at present of limited use as the reported values correlate only roughly with the applied pressure since the reported value is actually the approximate touch area. As pressure increases, the touched area becomes greater. Unfortunately, the mapping between actual pressure and reported values is not only unclear but also inconsistent.

pressure-sensing could be added to future multi-touch devices

Existing multi-touch technologies supporting pressure input, however, could be adapted relatively easily for use on mobile devices. For instance, the UnMousePad (Rosenberg and Perlin, 2009) uses two sheets of Mylar printed with an array of line-shaped electrodes of force-sensing resistive (FSR) ink. The two sheets are aligned such that the electrodes from a grid. When pressure is applied to the UnMousePad, the grid nodes in the vicinity of the touch are compressed, creating an electrical contact. The electrical resistance of the contact points decrease as more pressure is applied. Thus the UnMousePad can detect multiple touches as well as their pressure.

FSR-ink-based resistive touch sensors could be adapted for use on mobile devices

A current issue prohibiting the use of resistive pressure-sensing touch pads on mobile devices, is that they need to be transparent in order to be fitted over a device's display. Current force-sensing inks are carbon-based and thus have a black, opaque color.

Pressure input on mobile devices will allow a higher degree of local degrees of freedom. For instance, critical device functions, such as deleting personal data could be made more difficult to activate by requiring more pressure to be applied for activation. We explore several aspects of pressure-based input on mobile devices in Section 4.5.

pressure input allows more local degrees of freedom

Autostereoscopic Displays A number of recent devices have appeared that feature (autostereoscopic) 3D displays. Examples are the *Nintendo 3DS* (Nintendo, 2012)[W] handheld game platform or the *HTC Evo 3D* (HTC, 2012)[W] smart phone.

The current autostereoscopic display technologies only have a fairly limited field of view where the 3D effect can be observed. Wedge optics could be a possible future technology that could be used to build improved autostereoscopic 3D displays (Large et al., 2010; Travis et al., 2009).

3D displays could have a number of potential benefits on mobile devices. 3D user interfaces for games or artificial reality applications would become more natural to use, due to added depth cues. Consumption of videos filmed in 3D would become possible. A larger, virtual display area could be realized using peephole techniques, e.g. in a way similar to the IllusionHole (Kitamura et al., 2001), if a way was found to enable effective eye tracking on mobile devices.

depth imaging will enable around-device interaction

Depth Imaging Depth-imaging cameras have recently become popular input devices for electronic entertainment devices such as the Microsoft Kinect (Microsoft Inc., 2012a)[W]. Currently, depth sensing cameras are still too bulky to be built into mobile devices. This miniaturization is, nevertheless, very likely to happen, and the possible applications of depth cameras in mobile devices are very exciting. Applications range from scanning of objects in 3D from mobile phones to gesture recognition to improved interaction in augmented reality applications.

HoverFlow (Section 4.3) explores a depth-based user interface that can detect coarse input gestures such as hand swipes across the device with different hand poses. In Section 4.4 we explore an around-device gesture-based interface that uses a depth camera as an input sensor. The results of the accompanying user demonstrate the usefulness of integrating depth cameras into future mobile devices.

1.2 Research Motivation and Relevant Contributions

In this dissertation we develop sensor-based user interface concepts for mobile devices in order to study the following overarching questions:

1. How can sensors on current mobile devices be used to improve the usability of mobile devices?

2. What types of new sensor technologies could be incorporated into future mobile devices and what novel mobile user interfaces could be realized using them? Will this increase the usability of mobile user interfaces compared to the state of the art?

3. How can we facilite the access for practitioners and researchers to relatively complex user interface concepts such as motion gesture recognition?

In order to address these issues, we focus on three research domains within mobile HCI, which we believe are relevant to answering the questions asked in the previous paragraphs: *continuous interaction with model-driven user interfaces* (Chapter 3), *around-device and sensor-based interaction* (Chapter 4) and mobile interaction based on *motion gestures* (Chapter 5). In the following, we describe and motivate our choice of these topics in more detail.

1.2.1 Continuous Interaction

state-space models allow automatic control of a large number of output parameters with only a few input parameters

User interfaces based on continuous interaction allow the user to control a non-discrete parameter through a continuous feedback loop. User input is usually entered into a dynamic model which then calculates the next state of the system. Thus, these types of models are also referred to State-Space models in the literature (Eslambolchilar and Murray-Smith, 2008c). State-Space models allow the creation of smart user interfaces that can, from a low number user input parameters, control a much higher number of output parameters. For example, one-dimensional tilt input fed into a state-space model can control the zoom level, scrolling speed and the current scroll position in a document for a 1D scrolling task (Cho et al., 2007).

state-space models usually follow a physical analogy to assist the user's the conceptual model

A usability drawback of controlling a large number output parameters from a low number of input parameters is that it can become unclear to the user how the system responds to his actions. Thus, state-space models usually follow some sort of physical analogy, which aims to make the system's behavior intuitively clear to the user. Therefore, it is very important to calibrate carefully the parameters of State-Space models in order to enable the users to obtain a clear mental model of how the State-Space model is reacting to their input.

In this work, we present a novel tilt-based map scrolling interface called *Semi-Automatic Zooming* (SAZ). SAZ extends a *Speed-Dependent Automatic Zooming* (SDAZ) approach based on a State-Space model with 2D scrolling and manual control of a base zoom level. Study results indicate that SAZ performs significantly better than SDAZ and could be an alternative to multi-touch for map navigation tasks.

1.2.2 Around-Device Interaction

a larger interaction volume allows for more input possibilities

Interaction in and on the physical space around the mobile device, or Around-Device Interaction (ADI), is an emerging research topic in the field (Butler et al., 2008b). The premise of ADI is to free the device's user interface from the physical constraints of the mobile device, and to utilize the space surrounding the device to provide richer input possibilities. This is desirable as the capabilities and processing power of mobile devices are evolving faster than their input possibilities. To have a larger physical interaction volume can allow for a richer and more powerful user experience and a better use of the capabilities and features found on mobile devices.

ADI interfaces allow implementations of interfaces for fine-grained control of user interface parameters and also enable the implementation of gestural interfaces using the space in the vicinity of the device for movement. Technologies supporting the implementation ADI interface include cameras, depth cameras and IR distance sensors.

This work presents the following contributions in the area of around-device interaction:

- *HoverFlow* (**?**) in Section 4.3. HoverFlow is a mobile user interface that explores the use of gesture input in the space above the mobile device's screen.

- The *iPhone Sandwich* (Essl et al., 2009), a prototype device that provides of pressure-based front and rear multi-touch input capabilities (Section 4.5).

- To study the benefits (if present) that rear-of-device input offers, we conducted research comparing front, rear and tilt-based input for 3D rotation tasks (Kratz and Rohs, 2010a), in Section 4.6.

- *PalmSpace* (Section 4.4) is a further ADI user interface prototype. Using a depth camera, PalmSpace tracks the user's hand pose in 3D. We study the usability of PalmSpace for rotation tasks.

1.2.3 Motion Gestures

Motion gestures are part of everyday life for most humans. Because of this, gestures can be more intuitive to learn than abstract commands. This in turn can lead to a reduction of cognitive load while operating a user interface.

Due to physiological differences, each user performs gestures differently. Thus, gestures can convey a biometric signature that is specific to the user performing them. Furthermore, gestures play a large role in subconscious communication between humans. Similarly, it may be possible to glean information about the user's emotional state from the way gestures are entered.

Many modern smart phones are equipped with accelerometers and gyroscopes. These sensors allow the implementation of user interfaces that use motion gestures as input. Motion gestures can be used in a number of applications ranging from gaming to security. Because motion data obtained from gyroscopes and accelerometers generally has a high noise content and is prone to variations due to changing user posture and outside conditions, developing algorithms for motion gesture recognition is challenging.

This dissertation contributes the following work in the area of motion gestures;

- The *3 Dollar Gesture Recognizer* (Kratz and Rohs, 2010b) is a lightweight, data-driven gesture recognizer for motion gestures that is simple to implement, requires little training data and does not rely on external toolkits.
- *Protractor3D* (Kratz and Rohs, 2011) improves upon the $3 Gesture Recognizer by applying a closed form solution to correctly match gesture and template rotation differences, thus significantly increasing the recognition rate as well as make in gesture recognition rotation-invariant.
- We present the results of an experiment in gesture-based authentication and show that it may be a promising technology for lightweight authentication in mobile devices.
- We present the results of a study that analyzes the impact of using gyroscope data in addition to accelerometer data for motion gesture recognition. We also look at different ways of combining the two data types.

1.2.4 Design Space of Sensor-Based Interaction

To be able to better discriminate between the sensor-based interfaces presented in this dissertation it is useful to classify them in the context of a design space.

Essl et al. (Essl and Rohs, 2007) presented a design space for sensing-

design space classifies input capabilities of sensor type

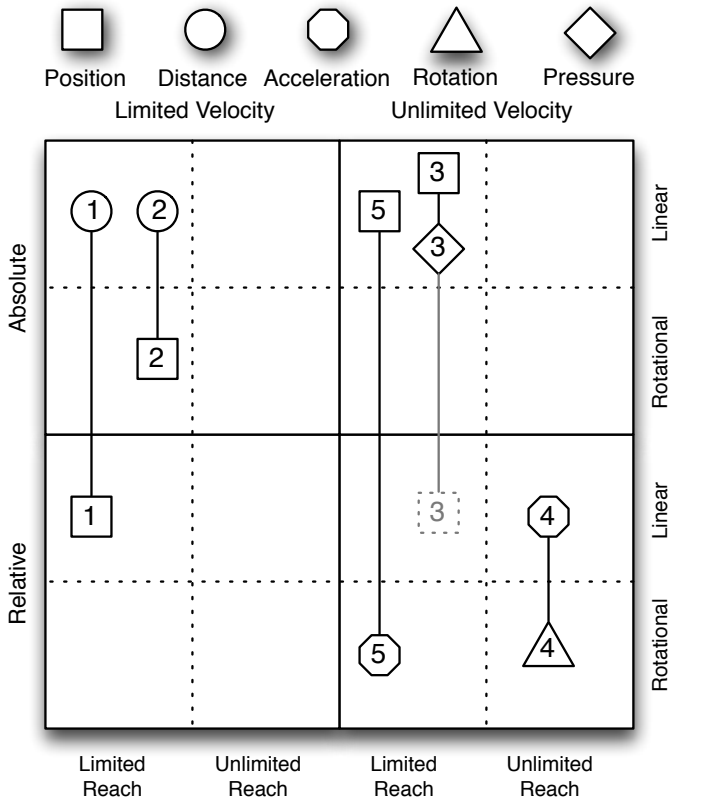


Figure 1.4: Classification of sensor-based user interfaces presented in this dissertation in Essl et al.'s Design space for sensor-based interaction.

based interaction with mobile devices in the context of musical performance. The basic idea of this design space, however is applicable well to the sensor-based mobile user interfaces we developed in this dissertation. The design space has four major subdivisions, *Absolute*, *Relative*, *Limited Velocity* and *Unlimited Velocity*, that relate to the type of sensor obtained and four minor subdivisions, *Rotational*, *Linear*, *Limited Velocity* and *Unlimited Velocity*, that relate to the type of user inputs that can be measured by the sensors. This design space is thus relatively useful for classifying the types of inputs which are possible by a given sensor-based user interface. To put the sensor-based interfaces developed in this dissertation into context using the design space, we modified the original measured inputs (*Position*, *Velocity* and *Acceleration*), by removing *Velocity* and adding *Distance*, *Rotation* and *Pressure*.

Figure 1.4 shows a classification of the sensor-based user interfaces

presented in this dissertation using Essl's design space. HoverFlow (1) uses an array of infrared distance sensors to obtain an absolute distance of the user's hand from the mobile device's display as well as the ability to sense the relative motion of the users' hand across the display. PalmSpace (2) uses a depth camera to obtain distance and rotation information of the user's hand. Pressure-based dual-sided multitouch (3) allows absolute position and pressure values as user inputs. Motion gestures and gesture-based authentication (4) can use a combination of accelerometers and gyroscopes, which sense acceleration and rotation, respectively. Finally, Semi-Automatic Zooming (5) uses tilt input for relative control of the map scroll speed as well as an absolute mapping of a touch screen slider value to the base zoom level of the map interface.

1.2.5 Classification of Contributions in SIGCHI HCI Curriculum

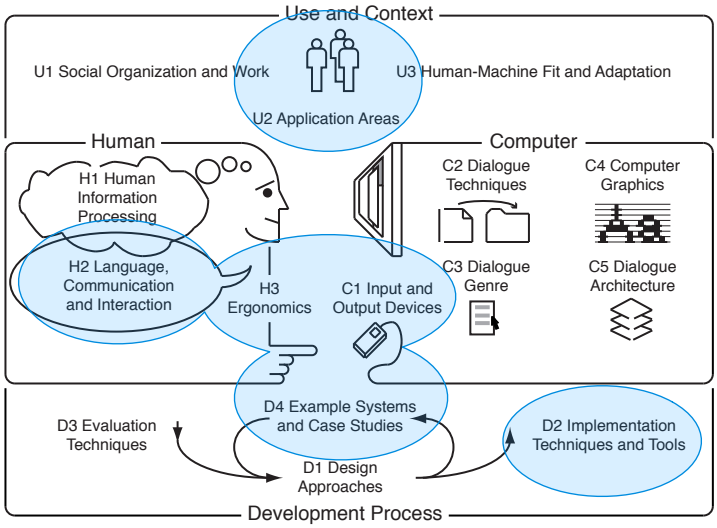


Figure 1.5: The contributions of this dissertation indicated on a graphical representation of the ACM SIGCHI classification system for HCI. The categories contributed to are: *U2, H3, C1, D2, D4*.

The work on this dissertation contributes to a range of fields within HCI. Looking specifically at the ACM[9] SIGCHI[10] Curriculum (Hewett et al., 2009)[W] (Figure 1.5), the contributions lie in the following areas:

- **U2. Application Areas** biometric, gesture-based authentication (GBA) for mobile devices (Kirschnick et al., 2010).
- **H3. Ergonomics** study on the properties of pressure-based input on mobile devices (Stewart et al., 2010)

[9]Association for Computing Machinery.
[10]The ACM Special Interest Group on Human-Computer Interaction.

- **C1. I/O Devices** development of gesture recognition algorithms (Kratz and Rohs, 2010b, 2011), semi-automatic zooming for map navigation (Kratz et al., 2010a), exploration of dual-sided (Kratz and Rohs, 2010a), pressure-based interaction by creating an iPhone Sandwich prototype (Essl et al., 2009).

- **D4. Example Systems** example applications for ADI (**?**Kratz et al., 2012a), tilt-based map navigation system (Kratz et al., 2010a).

- **D2. Implementation Techniques/Tools** end-user programming environment for rapid prototyping of gesture-based interfaces

1.3 Dissertation Structure

This dissertation is structured to emphasize the contributions in the three main areas of research we focused on that are relevant to the topic of sensor-based mobile interaction: continuous interaction using model-driven user interfaces, sensor-based and around-device interaction and, lastly, motion gestures. In the following, we outline the further structure of this dissertation.

In Chapter 2, we survey related work in order to obtain an overview of the state of the art in sensor-based mobile interaction. We highlight how the contributions in this dissertation build upon or extend related work in the field.

We present the results of our research on continuous interaction with model-driven user interfaces in Chapter 3. This chapter portrays how the sensors embedded in current mobile devices can be used to improve the usability of mobile user interfaces by allowing the development of user interface models that simultaneously control a larger range of user interface parameters than the user would be capable of doing manually. In addition, the results in this section support our argument that by using the built-in sensors of mobile devices more effectively, the usability of mobile user interfaces could be significantly improved in the future.

We examine future sensor technologies that could be incorporated into mobile devices in Chapter 4. We introduce the concept of around-device interaction (ADI) as a way of extending the input capabilities of mobile devices to encompass their entire surface as well as the physical space surrounding them. To evaluate the concept ADI, we describe three hardware prototypes, and present the results of user studies exemplifying the usability improvements for devices supporting ADI in comparison to conventional mobile user interfaces.

Motion gestures are a promising user interface input technique for mobile interaction. A difficulty for developers intending to make use of motion gestures in their applications is that in order to recognize non-trivial motion gestures, developers may need knowledge of machine learning techniques and need to resort to specialized libraries or toolkits. One of the goals of this dissertation is to facilite access to mobile gesture recognition for practitioners and developers. In Chapter 5 we thus present a set gesture recognizers that feature a low implementation effort and are easy to incorporate into mobile applications. The algorithms were developed to support rapid application development and prototyping. In this chapter we also cover a novel application for motion gestures, gesture-based authentication (GBA). Through the results of a user study we show this type of authentication is both feasible to implement on mobile devices and resistant to visual attack attempts. The chapter concludes by an analysis of the effects of combining accelerometer data and gyroscope data on the accuracy of motion gesture recognition.

With the goal of making gestural interaction available to the end user and supporting prototyping of mobile gestural interfaces, we present *Mayhem* in Chapter 6. Mayhem is an end-user programming environment that can be used to prototype gestural interactions for scripting tasks or home automation. We present a sample application case highlighting Mayhem's salient features.

In Chapter 7, we address open issues and enumerate possibilities for future continuation of the work presented in this dissertation.

We conclude this dissertation in Chapter 8 by summarizing and contextualizing the contributions and discussing their implications for the field of mobile human-computer interaction.

Chapter 2

Background and Related Work

In this chapter we survey previous work that is related to this thesis. It provides an overview of the state of the art in sensor-based interaction, in order for the reader to better classify the contributions presented in this work. The ordering of the related work sections in this chapter roughly follows the structure of the rest of the thesis.

2.1 Continuous and Model-Driven Interfaces

Speed-dependent automatic zooming (SDAZ) is the navigation technique that is the foundation of the work presented in Chapter 3 Continuous Interaction and State-Space Systems.

SDAZ was discussed by (Igarashi, 2000) in 2000 as a technique for navigating large documents. They conducted a preliminary user study comprising a 1D document scrolling task and a 2D map navigation task and compared SDAZ to traditional pan-and-zoom navigation. Whereas SDAZ was clearly preferred by the subjects in the 1D scrolling task, the preference for SDAZ in the map navigation was only slightly higher than traditional navigation. Moreover, the authors did not observe an improvement in task completion time using SDAZ in either the 1D or 2D tasks. The authors used a reciprocal function with the scrolling speed as denominator to automatically control the magnification level.

(Cockburn and Savage, 2003) conducted a similar study with a larger number of participants. They, too, compared traditional with SDAZ navigation (with a linear mapping from scroll speed to zoom level) for 1D document scrolling and 2D map scrolling. Interestingly, their results are significantly in favor of SDAZ, both in terms of task completion times and NASA TLX (Hart and Staveland, 1988a) workload assessments. In contrast to the work presented in this thesis, Cockburn

et al. used a 2D map display with fixed boundaries, which is unlike modern *slippy maps*, which we used for our studies. Slippy maps are used by most online map providers such as Google Maps, for instance. Slippy maps allow for infinite scrolling in latitude and longitude, with a seamless rollover at the map content boundaries.

In contrast to previous studies on SDAZ-based interfaces for map navigation, the map interface we used contains real map material that covers a substantial geographic area in a wide range of zoom levels. To our knowledge, such a realistic testbed has not been used for previous studies on mobile devices.

(Wallace et al., 2004) evaluated SDAZ scrolling speeds for a 1D text scrolling interface. Comfortable SDAZ scroll speeds are dependent on the assigned task (i.e., reading generally allows higher movement rates than looking at abstract data), visual perception, and the size and resolution of the device's display. For our research, we could not apply the results of that paper directly due to the different target domain (mobile map navigation as opposed to text document scrolling) and because their study was conducted using a 19 inch monitor connected to a desktop PC. We did however include some high-level practical advice, such as preferring an early zoom-out, in the interface for our study. Cockburn et al. conducted a similar study, comparing SDAZ and several variants of SDAZ, (van Wijk and Nuij, 2004), with traditional scroll bars in a 1D document scrolling task.

A paper with significant relevance to our work is *OrthoZoom Scroller* (Appert and Fekete, 2006), a 1D scrolling technique that uses one mouse axis to input the panning speed and the orthogonal axis to control the content's zoom (a similar technique, *GestureZoom*, had been previously presented by Patel et al. (Patel et al., 2004a)). Appert's results have shown that OrthoZoom can be twice as fast as SDAZ. The zoom-level slider in our SAZ prototype (Section 3.1.2.6) interface has a similar function as the orthogonal axis in OrthoZoom and can also be considered an orthogonal input dimension. In contrast to OrthoZoom, the slider in our implementation does not totally override the automatic zooming behavior but only sets the base zoom level for SDAZ to operate on.

A study comparing pen-based rate control for map scrolling using SDAZ, pen-pressure-based zooming and tilt-based zooming was conducted by (Büring et al., 2008). The authors conducted their study on a tablet PC with the content scaled to the screen resolutions of mobile devices. In contrast to our study, they did not use tilt for rate-control and their map content covered a much smaller geographic area, which is likely to have reduced the amount of zoom levels needed. In their

study, the application's display size has a significant effect on the semantic (i.e., unguided) navigation tasks, SDAZ significantly reduces the task completion times in navigation tasks using HALO (Baudisch and Rosenholtz, 2003) to guide the users and a non-significant increase in task completion times when SDAZ is used in unguided navigation tasks.

Murray-Smith et al. created a dynamic systems-based *state-space model* for SDAZ (Eslambolchilar and Murray-smith, 2004; Eslambolchilar and Murray-Smith, 2008c) and discussed its application in a tilt-based 1D text browsing interface for PDAs. In their approach, the SDAZ "camera" is modeled as a physical object in an environment simulating mass and friction, which is coupled to the user's input. Using this model allows the creation of automatic zooming behaviors that are modeled on physical analogies, which are likely to be more easily understood by the users. In addition, the State-Space Model allows developers to precisely tune the interface's behavior using a small set of parameters. For our contributions in Chapter 3 we adapted this previous work by extending the model proposed by Murray-Smith et al. to support 2D map scrolling with automatic zooming.

2.2 Sensor–Based Interaction and Around-Device Interaction

Interaction in the vicinity of a mobile device, *Around-Device Interaction* (ADI), has been discussed in previous work and is also the focus of ongoing research efforts. Our contributions in Chapter 4 were developed to further explore the design space of ADI and sensor-based interaction.

Our presentation of related work focuses on two important areas in the space of ADI: sensing input *in relation to the device* and sensing input *in the environment* (including the user's body).

2.2.1 Sensing Input in Relation to the Device

The Gesture Pendant (Starner et al., 2000) is a chest-worn device that consists of a wireless camera with IR illumination. This set-up was used to control a home automation system via in-air hand gestures. An additional use was monitoring for pathological tremors through analysis of the characteristics of the hand gestures.

(Cassinelli et al., 2005) developed a system for finger tracking in free space based on a steerable laser beam. This system is highly suited

for deployment in a mobile device since the laser, imaging sensor and micro mirror are very compact relatively low-cost components. The tracking and depth resolutions of the system are comparable to current depth imaging cameras.

MagiTact, *Abracadabra* and *Nenya* (Ketabdar et al., 2010b; Harrison and Hudson, 2009b; Ashbrook et al., 2011) are systems for around-device motion detection that use a magnetometer to sense the presence of a magnetic token worn on the user's finger. Magnetometers are present in many current mobile devices. However, it is difficult to derive precise 3D position information from these sensors. Furthermore, magnetometer-based techniques force users to use an input artifact equipped with a magnet, such as a magnetized ring or a stylus with a magnetic tip. Another limitation of the such systems is that only a single object can be tracked simultaneously.

SideSight (Butler et al., 2008b) used IR distance sensors to implement multi-touch input to the sides of a mobile device, when the device is placed on a flat surface.

2.2.2 Sensing Input in the Environment and on the Body

A number of projects have explored the concept of sensing input in the environment in vicinity of the user, and on the users themselves. The premise is that input areas are ubiquitous—we only need to find out how to sense user input on them. All of the following techniques can be incorporated into mobile devices.

Acoustic input via scratching on a mobile device was proposed by (Murray-Smith et al., 2008). (Harrison and Hudson, 2008) explored the use of arbitrary surfaces such as walls, tabletops or fabrics can be instrumented with a microphone. Using machine learning techniques, different gestures can be classified from the acoustic signature of the scratch.

Using the body as an antenna for ambient electromagnetic noise, (Cohn et al., 2011) could classify the location of touch inputs on surfaces near a source of electromagnetic noise, such as a wall outlet or light switch. Measurements of very low voltages on the user's body, which are caused by external electromagnetic noise, were used as input features for classification. The scope of this concept was further extended to detect free-space body gestures in environments with electromagnetic noise (Cohn et al., 2012). Moving parts of the body alters its properties as an antenna, which in turn modifies the voltages measured due to electromagnetic noise. Although the techniques discussed previously look promising, they are currently only experimental, and it is

unclear how they would perform in environments with high amounts of electromagnetic noise or highly varying amounts of noise, e.g., in large buildings, industrial settings or in electrified public transport vehicles.

SkinPut (Harrison et al., 2010) uses the transmission of sound via the body's skin and the skeletomuscular system to localize on-body input events. Harrison developed a sensing device worn on the upper arm that could locate touches at arbitrary positions on the user's arm. The device used a set of piezo sensors tuned to a set of resonant frequencies transmitted from touches on the user's arm through body tissue and the skin.

OmniTouch (Harrison et al., 2011) uses a shoulder-mounted projector and depth camera combination to enable touch interaction on arbitrary everyday surfaces. The depth camera is used to automatically acquire surfaces for output, detect touch events and to warp the projected image to approximate the orientation of the projection surface.

2.3 Rear-of-Device Input

HybridTouch (Sugimoto and Hiroki, 2006) was one of the first mobile device prototypes with two-sided input capabilities. The authors also developed dual-sided interaction techniques for map navigation.

Rear-of-device input was further explored by (Wigdor et al., 2007c). The prototype Wigdor et al. developed consisted of a display with two resistive touch screens, one on the front and one on the back. In addition, a camera was mounted at a distance from the rear touch screen, in order to detect the presence of the user's hands such that a "shadow" image of the hand could be displayed to the user. This shadow image could also be used to represent a hover state for rear-of-device input. In case of missing visual feedback of the hand or finger position when interacting behind interactive surfaces, Wigdor demonstrated that minimum target sizes of around 4.5 cm are required for users to reliably acquire them (Wigdor et al., 2006).

Baudisch et al. explored rear-of-device input on very small devices (Baudisch and Chu, 2009). Display-based techniques for compensating occlusion due to the *fat finger problem* (Siek et al., 2005), e.g., Shift (Vogel and Baudisch, 2007), aren't applicable below certain screen sizes due to the missing screen space for showing additional information such as callouts. Baudisch et al. show that with rear-of-device interaction, complex tasks can still be accomplished, even on devices with a screen size smaller than one inch.

Wobbrock et al. conducted an extensive study input performance for front-of-device and back-of-device interaction (Wobbrock et al., 2008). The authors specifically look at performance of Fitts' Law performance, feedback mechanisms for back-of-device interaction and compare stroke-based text entry performance in front and on the rear of the device.

2.4 Pressure-Based Input

There is a substantial body of work on pressure-based input with pens and styli. (Mizobuchi et al., 2005; Ramos et al., 2004) show that users do not keep precise pressure levels well without addition al feedback. Ramos et al. suggest pressure widgets as a form of visual feedback to improve performance of pressure-based input. Pen pressure has been used to improve target selection tasks (Ren et al., 2007). They used continuous pressure to control the size of a circular cursor area as well as the zoom level for small targets.

In contrast to touch, it is easier to simultaneously apply pressure and move a pen on a screen. When using direct touch, friction between the finger and the touch surface quickly increases with pressure, such that moving at the same time becomes difficult. *Zliding* (Ramos and Balakrishnan, 2005) combines sliding for scrolling and pen pressure for zooming. (Ramos and Balakrishnan, 2007a) proposed *pressure marks*, which are pen strokes with continuously changing pressure, as input for graphical user interfaces.

Pressure input on handheld devices directly with the fingers has also been explored. (Harrison et al., 1998) devised the idea of using pressure for embodied interaction with devices. *Gummi* (Schwesig et al., 2004) used bending to control gradual transitions between views, transparency and zooming. (Scott et al., 2009) explored force gestures for mobile devices, such as bending, compressing, squeezing and stretching.

(McCallum et al., 2009) present a mobile text input technique where each key can sense three different pressure values: a soft press invokes the first, a medium press the second and a firm press the third character mapped to the key.

For large multi-touch surfaces (Benko et al., 2006) propose mapping the on-screen width of the finger image (in the case of video-based multi-touch surfaces) to previous click selections and to improve precision of selection. Indeed, similar "simulated" pressure readings are obtained from size of the finger impression in mobile capacitive

touch screens, and can be accessed on Android devices (Google Inc., 2012a)[W] and (unofficially) on iOS devices (Stackoverflow user KennyTM, 2010)[W]. Of course, these approximations are not as pressure sensors such as force-sensing resistors (FSRs).

2.4.1 Mapping of Pressure to Input Values

Previous work has discussed different kinds of discretization and transfer functions to process raw sensor input in order to achieve better user performance. (Cechanowicz et al., 2007a) explored different discretization functions for a pressure-sensitive mouse and found that a quadratic mapping centered at the lower range works best. In (Ramos and Balakrishnan, 2005) a parabolic-sigmoid transfer function was used. (Shi et al., 2008) found that a fish-eye discretization function was superior to other mappings.

2.4.2 Feedback for Pressure Input

Most pressure-based input techniques rely on continuous visual feedback. Tactile and multimodal feedback play an important role in pen-based interfaces (Lee et al., 2004; Liao et al., 2006). (Rekimoto and Schwesig, 2006) implemented a three pressure-level button ("not pressed", "pressed lightly" and "pressed hard"). The button provided tactile feedback upon corssing the boundaries of these levels. The effectiveness of tactile feedback in mobile devices has been explored in (Brewster et al., 2007b; Hoggan et al., 2008; Luk et al., 2006). *EarPod* (Zhao et al., 2007) investigated auditory feedback for eyes-free interactions on mobile devices.

2.5 Gestural Input in Mobile Interaction

In Chapter 5 we present our work on mobile interaction using motion gestures. Since previous work on 2D stroke recognition is highly relevant, we will discuss it first, followed by work on motion gestures, which are usually captured in 3D from tri-axis accelerometers and gyroscopes.

2.5.1 2D Pen and Touch Gestures

The *$1 Gesture Recognizer* (Wobbrock and Wilson, 2007) is a recognizer for pen or touch strokes on touch-sensitive screens. The emphasis of the $1 Gesture Recognizer lies on simplicity and ease-of-implementation. It does not require any external toolkits and recognizes input strokes robustly. The algorithm is so simple that its pseudocode completely contained in the paper. Multistroke recognition was added the $1 Gesture Recognizer by (Anthony and Wobbrock, 2010), and adds several useful optimizations to the basic concept. In Section 5.2, we discuss the *$3 Gesture Recognizer*, an algorithm that builds on the concepts of the $1 Gesture Recognizer to allow the recognition of 3D motion gestures.

Protractor (Li, 2010b) is a gesture recognition algorithm for touch screens. It is a template-based recognizer that resamples the gesture trace to get a vector with a fixed dimension, which is then translated to the origin and normalized. For each comparison with a template a rotation is performed that optimally aligns the input and template gestures. This is done in an efficient way using a closed-form analytic approach, in contrast to the *$1 Gesture Recognizer*. *Protractor3D* extends this concept to use a closed-form solution for optimal rotational alignment for 3D motion gestures (see Section 5.3).

Giving continuous feedback and also feedforward during gesture stroke input can be important, firstly because novice users can more effectively learn the gesture stroke vocabulary, and secondly because they get the opportunity to correct their inputs. *OctoPocus* (Bau, 2008) is a gestural input system for menu selection that provides both feedback and feedforward, and the authors show that OctoPocus performs favorably compared to a conventional hierarchical menu.

2.5.1.1 Use of Motion Gestures in Mobile User Interfaces

RexPlorer (Ballagas et al., 2007) is a location-aware mobile game that allows tourists to explore the city of Regensburg, Germany. The interaction is based on movement gestures: The user can cast different types of "spells, to solve a riddle and learn more about the city". RexPlorer uses very simple u-shaped gestures and an ad-hoc recognition scheme (Kratz, 2007b). The motivation for using gestures with simple shapes was to guarantee the gesture recognition when using noisy data for input. Motion information for gesture entry was obtained from 2D motion estimation of images from the mobile device's built-in camera, thus the system can only recognize motions based on 2D shapes.

(Kela et al., 2006) developed an accelerometer-based input mechanism for interaction in a smart workplace for designers and engineers.

DoubleFlip (Ruiz and Li, 2011) addresses the problem of false positive gestures by employing a simple delimiter gesture (the "double flip" gesture) for entering gesture input mode. The criteria for choosing the "double flip" gesture were that (1) the gesture had to be easy to perform and (2) the gesture had to be sufficiently distinct from everyday movement. To confirm the latter the authors investigated the false positive rate for a large corpus of everyday movement data.

(Hinckley and Song, 2011) explored how to combine touch input with motion gestures. This kind of multi-modal interaction opens up a design space that allows very expressive kinds of mobile interaction. In particular, the accelerometer can be used to gain additional information about touch events, i.e., "soft touch" or "hard touch".

2.5.1.2 Vocabularies for Interaction via Motion Gestures

Li et al. (Ruiz et al., 2011) applied the approach of Wobbrock et al. (Wobbrock et al., 2009) to mobile gestures. The idea is to confront users with a desired task goal (such as deleting items) and let users choose which gesture they find the best match for the goal. (Kuehnel et al., 2011) followed the same approach for gestural interaction in smart-home environments.

2.5.1.3 Gesture-Based Authentication

In Section 5.4 we present work on gesture-based authentication. There have been several previous publications on this topic.

(Patel et al., 2004b) created a gesture-based authentication scheme to authenticate mobile devices on public terminals via shaking. When the user selects a device to authenticate with, the device generates a gesture input prompt. If the user performs the gesture as requested, the device is authenticated an the user can interact with the public terminal via the authenticated mobile device.

(Farella et al., 2006) studied gesture-based authentication using an accelerometer-equipped device. The authors particularly focused on dimensionality reduction, comparing the results of PCA[1] and LLE[2] with

[1] principal component analysis (Jolliffe, 2002).
[2] locally linear embedding (Saul and Roweis, 2000).

a *k*-nearest neighbor classifier. In their study, PCA achieved better results than LLE. The overall results of the study lead the authors to the conclusion that gesture-based authentication is feasible given a small user population size. However, the authors did not evaluate the resistance of their system to visual attacks.

(Mayrhofer and Gellersen, 2007) developed a scheme for device-to-device authentication based on synchronous shaking. Acceleration data from the devices' accelerometers was used as a basis to generate and authenticate cryptographic keys. This general idea has also been developed further and commercialized in form of the *Bump* application (Wikipedia, 2012)[W], which allows the exchange of contact information between to smart phones by bumping them together.

Liu et al. evaluated gesture-based authentication, based on the *uWave* system by the same authors (Liu et al., 2009b,a). Their results show that movement gestures can be remembered at least as well as standard pass phrases. Furthermore, they tested the robustness of their algorithm towards visual disclosure of the gesture entry and found that visual disclosure significantly raises the false positive rate.

2.5.2 Recognition Systems for Mobile Motion Gestures

uWave (Liu et al., 2009b) is a motion gesture recognizer based on Dynamic Time Warping. The accuracy obtained by their system reached 98.6%. The paper suggests a template adaptation system, where a weak matching templates are periodically replaced by the current gesture input, in order to counter subtle changes in the user's gesture entry technique over time.

(Schlömer et al., 2008) presented an HMM-based classifier for gestures input via a WiiMote. The accuracy achieved is, however, does not appear to be greater than that of simpler, template-based approaches.

(Hoffman and Varcholik, 2010) compared a linear as well as an AdaBoost classifier for motion gestures input via a *Wii Remote*[3]. They analyzed the effects of adding rotational features to the for recognition by using data from the gyroscopes on the *WiiMote MotionPlus*[4]. Although our work in Section 5.5 partially replicates the study of the effects of gyroscope data, the sensor setup of the motion plus is different from that typical of mobile phones, as it has one two-axis gyroscope and one

[3]The WiiMote is a game controller manufactured by Nintendo that incorporates a 3-axis accelerometer (Lee, 2008).

[4]The MotionPlus is an attachment for the WiiMote that adds gyroscope to the WiiMote. (Wii Brew, 2012)[W] provides extensive information on sensor information provided by the MotionPlus.

single axis gyroscope. Mobile phones usually contain a single three-axis gyroscope. Moreover, we also focus on methods for combining accelerometer data with gyroscope data for algorithms that do not support feature dimensionalities larger than 3.

2.5.3 Development Tools for Gesture-Based Interfaces

In Chapter 6, we present *Mayhem*. Although Mayhem is not explicitly designed as a development tool for gesture-based interaction, it can be used to rapidly connect the output of gesture recognizers to complex functionality, eliminating the need for additional software development to attach functionality to gesture recognizers. This can be very useful, e.g., when prototyping gesture-based user interfaces. In the following, we discuss two related systems that, in contrast to *Mayhem*, focus more on developing the actual gesture vocabularies and gestures used for input.

SATIN (Hong and Landay, 2007) was a development toolkit for pen-based user interfaces. Built on top of Java Swing (Fowler, 1998)[W], SATIN tried to help developers of such interfaces by providing a scene graph, rotation and scaling support for virtual objects and also stroke recognizers and interpreters for pen-based strokes.

MAGIC (Ashbrook, 2010a) is a development tool for the design of gesture-based interfaces. It specifically addresses developers that have no knowledge of pattern recognition or machine learning. Additionally, Ashbrook and Starner recorded an "Everyday Gesture Library" that allows developers to test how well their gesture designs can be disambiguated from everyday movements.

Chapter 3

Continuous Interaction and State-Space Systems

"Any sufficiently advanced technology is indistinguishable from magic."

—Arthur C.Clarke

A number of mobile interaction tasks require the user to repeatedly perform the same input gesture in order for him to accomplish the tasks. Examples of this are mostly found when the user has to navigate an extensive set of data, i.e. navigating a lengthy list of contacts or scrolling through a long web page. A further example of such tasks is map navigation. Here, the space the user is navigating is extremely expansive, and scrolling takes place in two dimensions and at a much larger scale than scrolling in lists or multi-page documents.

Concretely, multi-touch user interfaces exhibit a weakness when users are tasked with navigating large data sets such as maps, forcing the users to use a high number of "flick" inputs to scroll as well as "pinch" inputs to zoom in or out of the map.

continuous input and automated control of certain UI parameters may lead to lower effort and better usability

Our motivation for the UI concepts and user studies presented in this chapter is to use the sensing capabilities found on existing mobile devices as well as dynamic UI models based on state-space systems to reduce the number of input repetitions needed for mobile map navigation, in order to increase the usability. Our two main strategies to accomplish this goal are to use continuous input vs. discrete input and to allow the UI to assist in controlling the zoom level and scrolling speed, using a state-space Speed-Dependent Automatic Zooming (SDAZ) model. The motivation behind automating a number of important user interface parameters is that by reducing the amount

of inputs the user needs to make, his effort is reduced, thus freeing some his mental capacity to allow him to better focus on the goal he wants to achieve by performing the navigation task.

In Section 3.1 we introduce Semi-Automatic Zooming (SAZ). SAZ uses tilt information obtained from the mobile device's accelerometer in order to control zoom, scroll speed and scroll direction. SAZ builds upon SDAZ by adding the capability to override automatic zooming at any time through a slider that controls the base zoom level.

Motivated by the results obtained from our user evaluation of SAZ, we evalute if automatic zooming can be integrated into a standard multi-touch interface in a more subtle way. This approach aims to combine the advantages of these two techniques in a single map navigation interface. Section 3.2 presents a technical description of such a system and the results of preliminary we conducted to evaluate it.

3.1 Semi-Automatic Zooming for Mobile Map Navigation

Semi-Automatic Zooming (SAZ) is a novel interaction technique for mobile map navigation (Kratz et al., 2010a). SAZ gives the user the ability to manually control the zoom level of an SDAZ interface, while retaining the automatic zooming characteristics of that interface at times when the user is not explicitly controlling the zoom level.

We conducted a user study using a realistic mobile "slippy map" that covers a large geographic area at a high resolution[1]. In the study, we compare SAZ with existing map UI techniques, such as multi-touch and Speed-Dependent Automatic Zooming (SDAZ). For the SDAZ implementation, which is also underlying to SAZ, we extended a dynamic state-space model to accept 2D tilt input for scroll rate and zoom level control.

SAZ significantly improves usability over SDAZ

The results obtained in our study reveal that SAZ has a significantly improved usability, in terms of task completion times and subjective ratings, over SDAZ and that, barring a steeper initial lerning curve, SAZ is comparable in performance and usability to a standard multi-touch map interface. Thus, the study indicates that SAZ can serve as an alternative input technique to multi-touch for mobile map interfaces, especially for such devices that are not equipped with multi-touch input, or as a companion technique to touch-based map navigation techniques.

[1]The software implementation of SAZ and the user study was conducted by Ivo Brodien in his Diploma Thesis (Brodien, 2009).

3.1.1 Introduction

Mobile map applications are becoming increasingly popular on current smartphones. This trend is likely to continue into the future, with the newest devices adding additional navigation-related features such as magnetic compasses.

interacting with mobile maps requires a high number of individual panning and zooming steps

We believe, however, that navigation interfaces for mobile maps can be significantly improved. Mobile maps provide expansive and simultaneously dense data on a wide range of scales, which makes displaying all data relevant to a specific navigation task difficult due to the small screen size on mobile devices. Using current mobile map interfaces, finding (and selecting) points of interest on mobile maps requires the user to perform numerous manual zooming and panning steps. This problem is exacerbated by the lack of a precise pointing device, such as the mouse on desktop PCs, on touch-screen-based mobile devices.

Speed-Dependent Automatic Zooming (SDAZ) (Igarashi, 2000) is a rate-based scrolling technique, in which the application automatically controls the content's zoom level depending on the scroll speed. The motivation behind automatic zooming is to preserve the optimal visual flow rate of the content by zooming out when the scroll rate increases. Also, switching between panning and zooming can increase mental load and user frustration. Thus, SDAZ is beneficial because it relieves users of an additional control dimension, the need to manually adjust the zoom level and allows simultaneous zooming and panning of the user interface. However, past studies have shown that SDAZ is not always comfortable for the users, although the results suggested that it improves rate-based scrolling (Cockburn et al., 2005b). As far as we know, we have conducted the first study to evaluate SDAZ interfaces for mobile maps using 2D tilt input.

undershooting and overshooting of targets is a major problem of SDAZ-based interfaces

We examine if dynamic rate-based control with automatic zooming as implemented in SDAZ is potentially a beneficial technique for map navigation tasks in mobile map applications. However, for precise selection tasks (i.e. selecting a point of interest on the map), SDAZ can become uncomfortable to use, as users are forced to slow down to precisely pinpoint their target. This can precipitate imprecise increases in zoom at higher zoom levels, in which users "lose" their targets from the interface's view. Such conditions unnecessarily force the users to resume scrolling in order to regain an overview of their location by zooming out. This target over- or undershoot (or "hunting") problem (Igarashi, 2000; Wallace, 2003) significantly adds to task execution times when using SDAZ.

adding a manual override of the zoom level is desired and may improve the user experience of SDAZ

In this section we aim to show that exclusively coupling the content's zoom level to the scroll rate is an unfair constraint, and that especially in the case of mobile map interfaces, the user is likely to benefit from zooming out the content without having to scroll it, as map navigation tasks require users to view and to classify larger features of the map (in a zoomed-out state) in order to locate the point of interest they are interested in. Thus, we propose a novel SDAZ-based mobile map navigation technique, which we refer to as Semi-Automatic-Zooming (SAZ). The idea of SAZ is to use a tilt-based SDAZ interface for basic behavior, with added manual zoom level control, i.e. in the form of a slider widget mapped directly to the displayed content's zoom level. Using manual zoom control, users can keep the zoom level constant without the need to scroll the map. On the other hand, when scrolling to distant points of interest, users are relieved of the need to explicitly control the zoom level.

3.1.2 Implementation and Design of the Mobile UI

In order to study tilt-based mobile map interfaces, we implemented a custom mobile map application on the Apple iPhone 3GS. Our application has three navigation modes: a "standard" multi-touch based-map interface, a tilt-based SDAZ interface and an SAZ-based interface.

3.1.2.1 Slippy Map

the interface used OpenStreetMap map data with Route-Me as the frontend

We used the *Route-Me* toolkit[2] to implement the slippy map interface used in our test application. Route-Me allows the use of custom tile sources, and allowed us to generate a custom map tile database which was stored directly on the mobile device in the map application's folder. The map tiles for the database were generated[3] from *OpenStreetMap* geo data which is available for download at no cost from (openstreetmap.org, 2012)[W]. Our approach using an on-device map tile database has the advantage of removing any network latency due to map tile retrieval, and allows us to scroll and zoom the map dynamically at a maximum refresh rate of 20 fps on an iPhone 3GS.

the map data covered Central Europe in high detail

The map tile database contains tiles for the world map from Open-

[2]Route-Me (Michael Tyson and collaborators, 2012)[W] is an open source replacement map viewer for iOS. The project allows full customizability of all the aspects of rendering the map on the device. The most important feature for us in this case was the ability to use a local map tile database. This allowed fast map rendering without lag from network transfers.

[3]The raw geo data (in XML format) was imported to a PostgreSQL database with geoinformation system extensions (OSGeo Project, 2012)[W]. The map tiles were then rendered using the *Mapnik* map renderer (Pavlenko, 2012)[W]. A Python script then inserted the individual map tiles into a SQLite data base for use by the mobile map application.

StreetMap zoom levels 3 (19567.88 meters per pixel[4] (m/px)) to 10 (152.87 m/px) and a detailed map of Central Europe is covered from zoom levels 11 (76.44 m/px) to 15 (4.78 m/px) which is detailed enough for identifying street names. In total, about 3.5 million map tiles are stored in the database, which has a size of almost 12 GB.

3.1.2.2 Touch Interface

The multi-touch-based interface is modeled after the user interface of the iPhone's on-board map application. Scrolling is controlled by dragging the finger across the map. The pixel distance between the start and end of dragging is directly mapped to the map canvas. A drag motion of n pixels scrolls the map by n pixels in the specified direction. Zoom is controlled by two-finger gestures. A "pinch-in" gesture zooms out the map and a "spread-out" gesture zooms in the map. Zooming is implemented by multiplying the current zoom level with the ratio of the starting and ending distance measured between the touch points of the pinch and spread gestures.

3.1.2.3 State-Space Model for SDAZ

We chose to implement SDAZ using the dynamic systems approach as described in (Eslambolchilar and Murray-Smith, 2008c) because this is the current state of the art in SDAZ implementations and has been widely published. As the authors only specify a model appropriate for 1D scrolling tasks, we extended their model to be able to support 2D scrolling tasks with tilt input.

To do this we had to extend the model update matrix to incorporate 2D tilt input based on accelerometer data and movement on a 2D plane, by adding two additional rows representing the scroll speed and position of the additional movement dimension:

$$\dot{X} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-R_h}{M} & 0 & 0 \\ 0 & 0 & 0 & \frac{-R_h}{M} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{A}{M} & 0 & 0 \\ 0 & \frac{A}{M} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \tag{3.1}$$

$(x_1, \ldots, x_5)^\intercal$ represent the model's state

$\dot{X}$ represents the changes of the state-space vector with $(x_1, x_2, x_3, x_4, x_5)^\intercal$ representing *PositionX*, *PositionY*, *SpeedX*, *SpeedY* and *Zoom*. M, R_h and A are parameters of the state-space model. M represents a mass in kg, R_h is the vertical resistance and A is a scaling constant. u_x, u_y are the tilt values obtained from the device's acceleration sensor, which is sampled at 20 Hz. The tilt values are used to update the state-space model as shown in Equation 3.1. The tilt values are computed using the arctangent of the acceleration values of the x (a_x) and y (a_y) axis divided by the acceleration value of the z axis (a_z), respectively:

$$\begin{aligned} u_x &= atan(a_x/a_z) \\ u_y &= atan(a_y/a_z) \end{aligned} \tag{3.2}$$

Before calculating the tilt values, we applied a basic low-pass filter in order to decrease noise and lower the impact of unwanted motion. The low-pass filter however cannot have too much influence on the values because otherwise sudden corrective scrolling movements would be too sluggish. Because of the two-dimensional input (tilt in x and y axis) we also had to adapt the change the zoom level update of our system in comparison to the one described by Murray-Smith. In Equation 3.1 the zoom update is represented by the u_z component. The zoom level update u_z is dependent upon the speed component sc and the current acceleration a_z in the z axis (the component of the earth's acceleration g in the acceleration value measured by the phone). We calculate u_z as follows:

$$u_z = (B/M)sc + (C/M)(1 + a_z) \tag{3.3}$$

our method of controlling the zoom updates allows the system to react more quickly to device tilt changes

We chose to implement the u_z update in this way in order to allow the device tilt to be the dominating input for controlling the zoom level. B and C are scaling factors, and M is the mass value defined for the model. As device tilt increases, a_z decreases. When the device is completely level $a_z = -1$. On the iPhone, $a_z < 0$ when the screen faces upwards. An interesting property of Equation 3.3, is that we can adjust the influence of both tilt (a_Z) and speed (sc) on the interface zoom. This way, we can, for example, set up the interface zoom to respond faster to device tilt changes. This allows the user to get a better overview of the direction he wants to navigate to by increasing the device's tilt to zoom out, before a higher scroll rate gradually sets in.

$$sc = max(abs(\dot{x}_3), abs(\dot{x}_4)) \tag{3.4}$$

[4]The meter per pixel values approximate map scales at Central European latitudes.

3.1.2.4 Parameter Selection for the State-Space Model

parameters control physical properties used by the state-space model

To allow for the camera to zoom continuously between zoom levels 3 and 15, and to fine-tune the coupling between scroll speed, tilt and zoom change, we had to adjust the parameters $M, R_h, A, B, C,$. In effect, these parameters model the physical properties of the virtual camera used in the SDAZ visualization. These properties are mass (M, in kg) and horizontal movement friction (R_h, in kg/s). A, B, C are used as scaling factors for the output values provided by the model.

we obtained working parameter settings analytically

In order to find values for the parameters, we sampled actual accelerometer data and ran an off-line simulation using *MatLab* (The MathWorks Inc., 2012)[W]. By analyzing plots of position, speed and location produced by the state-space model and different parameter settings, we found an initial set of values for the parameters. Following that, the parameter values were tested by expert users in the map navigation interface implemented for our study. For map navigation, feedback provided by expert users suggested that we use model settings that make the map zoom out quickly upon tilt input. This enables the user to orientate himself more easily, such that she can find the correct movement direction towards his target point of interest. We had to slightly adjust the model parameters obtained via our offline analysis and used the following settings in our user study:

$$M = 50, R_h = 5, A = 1.5, B = 100, C = 30 \tag{3.5}$$

3.1.2.5 Visual Feedback and Diving Mode

Because SDAZ requires the user to center the screen on the exact location of the target point of interest before zooming in to see its details, we added a box around our interface's map center, as shown in Figure 3.1. In the following we will refer to this box as the "map center box".

When using SDAZ, it is crucial for the users to know exactly how they are influencing the interface's rate control, in order for them to predict its future behavior. In our SDAZ interface, rate control is achieved by tilting the mobile device.

a visual feedback mechanism indicates the device's tilt

Because we found that users had difficulty to exactly judge the device's tilt, we implemented a visual feedback mechanism to visualize the current tilt angles. Visualization of the tilt in the x and y axis is achieved by drawing a small yellow square ("tilt indicator") in the area enclosed by the map center box (Figure 3.1 (a)) . If the device is level, the red dot remains in the center of this area, and indicates to the user that no

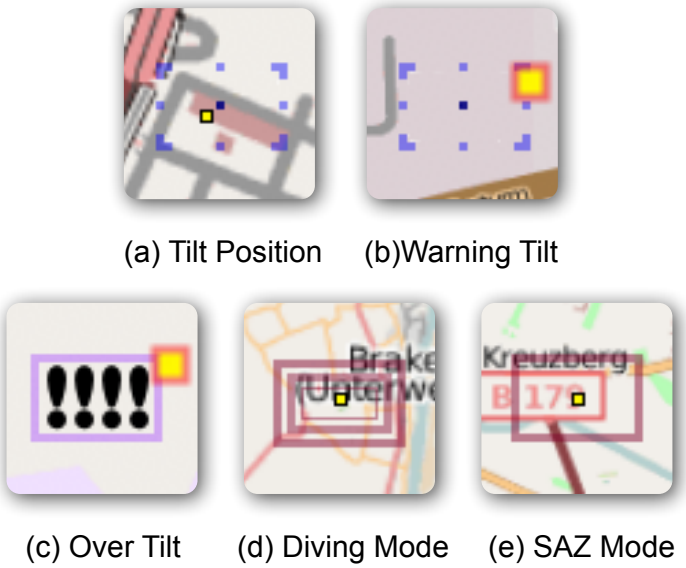


Figure 3.1: Visual feedback indicators for SDAZ and SAZ: (a) shows the visual feedback in default mode, with the tilt indicator displayed as a yellow square. (b) shows the warning-tilt indicator, when the tilt of the device approaches the over-tilt region. (c) indicates that the device has been tilted too far and that the interface is in over-tilt mode. (d) is the feedback displayed when in diving mode. (e) shows the default visual feedback given in SAZ mode. (Kratz et al., 2010a).

scrolling is taking place. If the device is tilted in a certain direction, the red dot moves outwards from the center proportionally to the degree of tilting, thus indicating the current scroll rate setting in the direction represented by the dot's offset with respect to the map center.

visual warning against over-tilting

To prevent operation of the interface at tilt angles larger than 30°, at which the map would not be clearly visible to the user anymore, we introduced feedback for an "over-tilt" mode, which freezes the interface and displays a warning rectangle around the map center box (Figure 3.1 (c)). When the device's tilt is within 10% of the boundary after which over-tilt is likely to occur ($\geq$ 27°), the tilt indicator is enlarged and surrounded by a red border, in order to warn the user that the over-tilt condition is imminent (Figure 3.1 (b)).

diving mode to prevent target overshooting

A common problem with SDAZ interfaces is the overshooting of targets, or "hunting effect" (Igarashi, 2000; Wallace, 2003). Because our interface simulates physical properties, such as friction and inertia, it is possible to overshoot the target, which makes the user compensate by tilting the device in the opposing direction. This, in turn increases the zoom level of the interface requiring further adjustment. A solution to this problem is to introduce a "diving mode" (Eslambolchilar and

Murray-Smith, 2008c). In our implementation, diving mode is initiated when the device is kept level for a time threshold of half a second.

When diving mode is activated, the interface smoothly zooms the map to the maximum zoom level. The map center box changes its shape (Figure 3.1 (d)) to indicate to the user, that the interface is in diving mode.

3.1.2.6 Semi-Automatic Zooming (SAZ) Interface

SAZ allows the user to control the zoom level manually with a sider

The idea of Semi-Automatic Zooming is to give the user manual control over zooming when he desires it. In our case, this manual control is implemented as a slider (or "SAZ slider"). As shown in Figure 3.2, the SAZ slider is placed on the right-hand edge of the screen. This placement of the slider is optimized for use with a right-handed user's thumb, but could just as easily be designed for left-handed use by moving the slider to the left edge of the screen.

(a) SAZ - Automatic Zoom Control

(b) SAZ - Manual Zoom Control

Figure 3.2: Placement and visualization of the zoom level slider in the SAZ map interface: (a) shows the SAZ interface in automatic zooming mode (no touch on slider), (b) shows the manual zooming mode (finger on slider thumb controls the zoom level). (Kratz et al., 2010a).

(a) Landmark Marker (b) Route Markers (c) Route Segment Update

Figure 3.3: Display items shown during the user study trials. (a) shows one of the markers as displayed in the *find landmark* task. (b) shows the markers to be crossed in the *follow route* task. (c) shows how the last route segment is updated (color change from yellow to orange) when the user crosses into the next route segment in the *follow route* task. (Kratz et al., 2010a).

the slider's thumb indicates the current zoom level when it is not being manipulated

The slider's thumb is mapped linearly to the interface's zoom level, and automatically updates its position to reflect the current zoom level when manual zooming is not being performed by the user. The user can manually take control of the zoom level by touching the slider with his thumb (Figure 3.2 (b)). This is reflected in a change of color change of the slider area from transparent gray to magenta.

The movement of the thumb is mapped in the following way: moving the slider up increases the zoom level, moving it down decreases the zoom level. Holding the slider's thumb at a constant position holds the zoom level constant. As the thumb's movement range is mapped directly to the range of zoom levels, the user can access all map zoom levels by a single movement with his thumb. Automatic zoom control is resumed when the slider's thumb has been released by the user. Automatic zoom control is indicated by the slider's color changing from magenta to transparent gray (Figure 3.2 (a)).

Scroll rate control is not affected by use of the slider, and tilt input to scroll remains enabled during use of the SAZ slider. One of the major advantages of using an underlying model-based SDAZ interface is that we can keep the visual scroll speed constant irrespective of manual zoom changes input with the SAZ slider. Due to the availability of manual zoom control, dive-mode has been disabled in SAZ.

Several features of the visual feedback displayed in SDAZ mode are retained in SAZ mode. Here, the tilt indicator is surrounded by a red box (Figure 3.1 (e)). SAZ mode also shows over-tilt and over-tilt warning feedback.

3.1.3 User Study

The aim of our experiment was to to compare task execution time, task load and user satisfaction results for the Touch, SDAZ and SAZ map in-

terfaces. Additionally, we compared two different task types, *find landmark* and *follow route*. Each task type is divided in two sub tasks with two different map scales *large* (regional to country scale) and *small* (city scale).

3.1.3.1 Participants

We invited a total of 13 right-handed participants, 8 male and 5 female. The average age was between 24 and 28 years. The majority of the participants did not have prior experience with either the iPhone or mobile map navigation. Participants were given monetary compensation after completing the study.

3.1.3.2 Experimental Design and Tasks

The experiment used a repeated measures, within-participant factorial $3 \times 2 \times 2$ design. Factors were input method (*multi-touch, SDAZ, SAZ*), task type (*find landmark, follow route*) and map scale (*large, small*).

users must find the exact location of a point of interest in the *find landmark* task, whereas in *follow route*, they must follow a prominent thoroughfare

The goal of the *find landmark* task was to analyze the usefulness of the interface control mode for searching for the exact location of a point of interest. The *find landmark* task was designed to represent finding the exact location of a certain point of interest, such as a restaurant, train station or popular sightseeing destination in an actual mobile map application. In a deployed mobile map application, the *follow route* task would be analogous to following a subway line map, for example.

In *find landmark* the task of the user was to locate and select a number of landmarks by navigating to them and selecting them by tapping on them. The landmark markers (Figure 3.3 (a)) were only visible from zoom level 11 onwards in the *large* map types and from zoom level 13 in the *small* map types, due to the smaller geographic area which resulted in a lower need to zoom out to lower zoom levels. Selection of the landmarks was only possible when fully zoomed in. The marker display constraints for the *find landmark* task were implemented to force the users to precisely locate and select the landmarks. During the trials, the participants were given a paper overview map on which the location and sequence of the landmarks for the current trial was shown.

The task in *follow route* was to follow a given route using the mobile interface. Each route consists of a number of waypoints (Figure 3.1 (b)) on that had to be crossed in sequence. The waypoints are connected by the route shown in yellow. When a waypoint is crossed, the color of the previous segment of the route is changed to red, to indicate the

direction of the next waypoint, as is shown in Figure 3.1 (c). Waypoints could only be crossed from zoom levels 13 to 15. The aim of this was to make the test subject follow the route as closely as possible.

Each task was tested on large and small map scales. Our idea was that regional-scale (150-500 km map diameter) map areas would force the user to zoom out to a lower zoom level as compared to the same task on a city-sized map scale (25-100 km map diameter). By having two types of map scale, we could measure the performance of the input techniques over a wider range of zoom levels and geographic areas. The *follow route* task was designed to evaluate the panning functionality whereas in the *landmark finder* task participants had to make additional use of the zooming functionality (Büring et al., 2008).

Each task was tested on large and small map scales. Our idea was that regional-scale (150-500 km map diameter) map areas would force the user to zoom out further compared to the same task on a city-sized map scale (25-100 km map diameter). By having two types of map scales, we could measure the performance of the input techniques over a wider range of zoom levels and geographic areas. The *follow route* task was designed to evaluate the panning functionality whereas in the *landmark finder* task participants had to make additional use of the zooming functionality (Büring et al., 2008).

3.1.3.3 Apparatus

We evaluated SAZ, SDAZ and multi-touch using a custom-built application running on an iPhone 3GS. In order to prevent lag from loading map tiles, all map tiles were stored in a database on the device. The map data contained the entire world map from zoom levels 3 to 10 (zoom levels 1 and 2 were not used) and a detailed map (zoom levels 11 to 15) covered the boundaries of Central Europe.

For the *find landmark* task, we provided the users with a printed overview map on paper, on which the location of the landmarks was indicated. We chose to provide a paper map in order to disambiguate our results from the participants' geographic knowledge.

3.1.3.4 Procedure and Dependent Measures

The study began with the participants filling out a general questionnaire on their gender, age and experience with smart phones and mobile maps.

the participants played a labyrinth game for warm-up

As a warm-up exercise, the participants were required to play three levels on a tilt-based labyrinth game (Illusion Labs, 2010)[W]. The goal of this exercise was to help the user learn the effects of tilting the device when using a tilt-based interface. The Labyrinth program gave the users an indication of the lag (if any), sensitivity and precision of the built-in acceleration sensor. Furthermore, the movement of the ball when the device was tilted was analogous to the movement direction of the camera in the SDAZ and SAZ interfaces. This may have helped to adjust the users' mental model when using the SAZ and SDAZ map interfaces in the actual user study.

We counterbalanced the order of input technique to prevent learning effects. For each input technique, we tested two task variants, *find landmark* and *follow route* as well as two map scales (large, small) per subtask, which in total resulted in 12 trials (one trial per condition) per user.

We measured task completion time as a quantitative performance measure. Task duration was measured directly on the test apparatus. Task duration gives us a good indication of the performance of the input technique, although as can (also) be seen from our results, task duration does not always correlate with satisfaction and ease of use.

Further qualitative measures were *task load* (measured using the NASA TLX questionnaire (Hart and Staveland, 1988a)) by input method and *satisfaction, learnability, usefulness* and *ease of use* (measured with the USE questionnaire (Lund, 1998)) and their respective ranking for each input method.

A NASA TLX questionnaire was provided to the test subjects after each trial, to record the direct impact each test condition had on the user's task load. Users were required to fill out the USE questionnaires after completing all the trials for each input method. Apart from the USE questions, participants were asked to provide negative or positive comments about the input technique, if they had any. After completion of the study, the users had to give a ranking of the USE variables on a further questionnaire.

3.1.4 Study Results

In the following, we analyze the quantitative results for average task completion time, as well as qualitative results for NASA TLX and USE questionnaires.

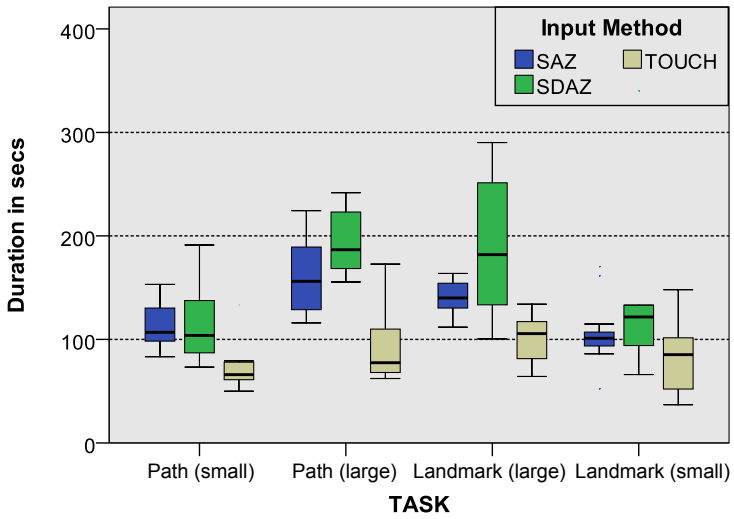


Figure 3.4: Boxplot of task completion time by input method and subtask. (Kratz et al., 2010a).

3.1.4.1 Task Completion Time

Table 3.1 shows the average task completion times ordered by subtask and input technique. The task completion time of SAZ is significantly lower than SDAZ in all subtasks, and multi-touch was the faster technique across all subtasks.

Figure 3.4 shows the boxplots of the task duration vs. input method and subtask. A univariate ANOVA yields a combined significant effect of input method $\times$ task type on the task duration ($F(5,155) = 13,756, p < 0.001$). However task type (*find landmark* or *follow route*) alone did not have a significant effect ($F(1,155) = 0.004, p = 0.948$). In contrast, input method showed a significant effect on task completion ($F(2,155) = 33.363, p < 0.001$). There was no significant input method $\times$ task type interaction.

3.1.4.2 NASA TLX

To analyze the effects of input method(multi-touch, SAZ or SDAZ) and task type (*find landmark* or *follow route*) on the NASA TLX ratings of *Effort, Frustration, Performance, Mental Demand, Physical Demand* and *Temporal Demand*, we conducted a multivariate ANOVA.

The following measures showed a significant difference:

- *Effort:* $F(5,150) = 3.27, p = 0.008$
- *Performance:* $F(5,150) = 2.64, p = 0.03$

Subtask	Input Technique	Mean (s)	Std. Dev. (s)
Path (Small)			
	SAZ	113.1	21.5
	SDAZ	117.4	35.9
	Multi-Touch	74.3	24.6
Path (Large)			
	SAZ	163.3	37.0
	SDAZ	200.4	42.7
	Multi-Touch	91.5	32.6
Landmark (small)			
	SAZ	106.8	30.3
	SDAZ	139.8	75.3
	Multi-Touch	83.9	37.5
Landmark (large)			
	SAZ	140.2	27.4
	SDAZ	192.3	65.0
	Multi-Touch	100.1	22.9

Table 3.1: Average task completion times by subtask and input technique.

- *Mental Demand:* $F(5,150) = 2,95, p = 0.01$

Input technique had a significant individual effect on the following measures:

- *Effort:* $F(2,150) = 7.48, p < 0.001$
- *Frustration:* $F(2,150) = 4.19, p = 0.02$
- *Performance:* $F(2,150) = 4.68, p = 0.01$

There was no significant interaction of input method $\times$ task type.

The effects of different input techniques on *Effort*, *Frustration* and *Performance* are as expected, due to the different characteristics of the input techniques. Mental, physical and temporal demand do not appear to be significantly affected by the input technique. The sole significant individual effect of task type was *Mental Demand* ($F(1,150) = 9.611, p < 0.01$), which can be explained by the difference in difficulty of our tasks. *Follow route* appears to be significantly easier than locating and finding a target on the mobile map, even with the target marked on a paper map for guidance.

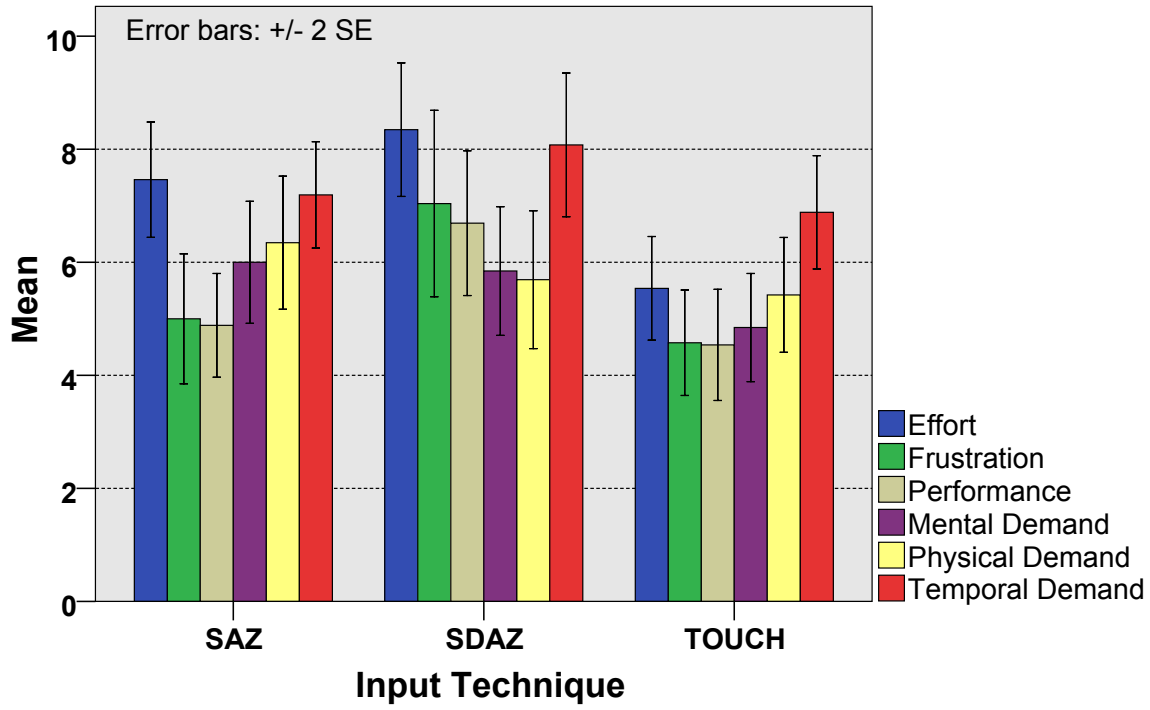


Figure 3.5: Results of NASA TLX questionnaires by input technique. (Kratz et al., 2010a).

To analyze the individual differences of the input techniques, we conducted a Sidak post-hoc analysis. For *Effort* there is a significant difference between multi-touch and SAZ (MD[5] $= 1.92, p = 0.032$) as well as multi-touch and SDAZ ($MD = -2.81, p = 0.001$), although there is no significant difference between SDAZ and SAZ. For *Frustration*, multi-touch differs significantly from SDAZ ($MD = -2.46, p = 0.023$), whereas there is no significant difference between SAZ and multi-touch. There is a borderline significant difference in *Performance* between SAZ and SDAZ ($MD = -1.81, p = 0.053$) and between multi-touch and SDAZ ($MD = -2.15, p = 0.15$). Interestingly, there is no significant difference in *Performance* between SAZ and SDAZ. Conforming to the results of the multivariate ANOVA, *Mental Demand, Physical Demand* and *Temporal Demand* had no significant effects.

Figure 3.5 shows a plot of the average ratings of the NASA TLX measures by input technique. In terms of user-rated *frustration*, there was no significant difference between SAZ ($M = 5.0, SD = 4.14$) and multi-touch ($M = 4.58, SD = 3.36$). SDAZ was rated significantly higher in *frustration* with the worst average rating of 7.04 ($SD = 5.94$). The user rated the *effort* worst for SDAZ ($M = 8.35, SD = 4.26$) with no signif-

[5] MD stands for mean difference, the difference of means between conditions in pairwise comparisons.

icant difference compared to SAZ ($M = 6.44, SD = 3.68$). Multi-touch was rated significantly better for *effort* with an average of 5.54 ($SD = 3.3$) A notable NASA TLX result is the (borderline) significant increase in user-rated *performance* of SAZ over SDAZ ($M = 4.88, SD = 3.3$ vs. $M = 6.69, SD = 4.6$; lower is better), however multi-touch remains best ($M = 4.54, SD = 3.55$).

3.1.4.3 USE Questionnaire

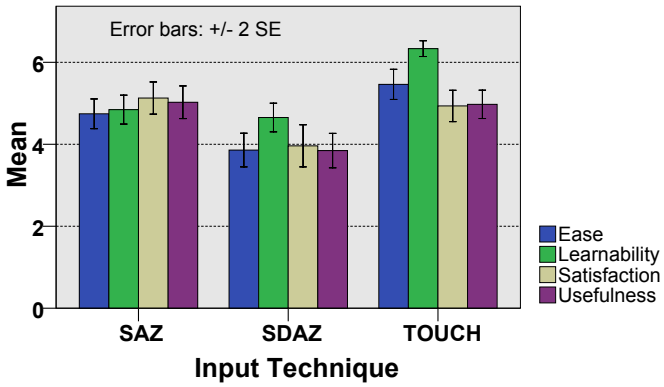


Figure 3.6: Results of USE Questionnaire by input technique (higher is better). (Kratz et al., 2010a).

The mean ratings given for each input technique on the USE Questionnaire are shown in Figure 3.6. We conducted a multivariate ANOVA on the results of the USE Questionnaire. Dependent measures were *Ease*, *Learnability*, *Satisfaction* and *Usefulness*. Task type and input method were chosen as factors. There were significant differences between all measures. The individual F-values are as follows:

- *Ease:* $F(2, 155) = 7.214, p < 0.001$
- *Learnability:* $F(2, 155) = 14.097, p < 0.001$
- *Satisfaction:* $F(2, 155) = 3.286, p = 0.008$
- *Usefulness:* $F(2, 155) = 4.839, p < 0.001$.

Task type had no significant effect on any of the measures whereas input technique had a significant effect on all measures:

- *Ease:* $F(2, 155) = 17.506, p < 0.001$
- *Learnability:* $F(2, 155) = 35.131, p < 0.001$
- *Satisfaction:* $F(2, 155) = 8.157, p < 0.001$
- *Usefulness:* $F(2, 155) = 11.567, p < 0.001$.

3.1.4.4 USE Rankings

Figure 3.7 shows the results of the USE questionnaire rankings. Sidak post-hoc analysis for the USE Questionnaire rankings shows that for *Satisfaction* and *Usefulness*, SAZ ranks at least as well as multi-touch. For the other USE measures, SAZ consistently received a higher rating than SDAZ and SAZ was rated significantly higher than SDAZ in *Ease*. The difference in *Learnability* between SDAZ and SAZ was not significant, as these two techniques seem harder for unskilled users to master than multi-touch.

The detailed results for the Sidak analysis are as follows: for *Ease* there were significant differences between SAZ and SDAZ ($MD = 0.88, p = 0.004$), between multi-touch and SAZ ($MD = 0.72, p = 0.027$) and also between touch and SDAZ ($MD = 0.88, p < 0.001$). Multi-touch was rated best ($M = 6.56, SD = 1.48$), followed by SAZ ($M = 4.74, SD = 1.31$) and SDAZ ($M = 3.85, SD = 1.48$). *Learnability* had the highest rating for multi-touch ($M = 6.34, SD = 0.69$) with a significant difference compared to SAZ ($MD = 1.49, p < 0.001$) and SDAZ ($MD = 1.68, p < 0.001$). SAZ had the second best rating ($M = 4.85SD = 1.27$), with SDAZ worst ($M = 4.65, SD = 1.26$). There was no significant difference in *Learnability* between SAZ and SDAZ. SAZ had the best rating for *Satisfaction* ($M = 5.13, SD = 1.42$), with multi-touch second ($M = 4.93, SD = 1.38$), although the difference was not significant. SDAZ had the lowest rating ($M = 3.96, SD = 1.85$) with significant differences compared to both SAZ ($MD = 1.67, p = 0.001$) and multi-touch ($MD = 0.97, p = 0.006$). SAZ also had the highest rating for usefulness ($M = 5.03, SD = 1.44$) followed by multi-touch ($M = 4.97, SD = 1.25$), although also in the case the differences were not significant. SDAZ received the lowest rating ($M = 3.85, SD = 1.51$) that was significantly different compared to both SAZ ($MD = 1.18, p < 0.001$) and multi-touch ($MD = 1.13, p < 0.001$).

Figure 3.7 reflects clearly the borderline differences in *Satisfaction* and *Usefulness* rankings for the input techniques. On average, SAZ ranks first in *Satisfaction* ($M = 1.55, SD = 0.55$) and *Usefulness* ($M = 1.67, SD = 0.60$), with multi-touch (*Satisfaction:* $M = 2.05, SD = 0.67$, *Usefulness:* $M = 1.79, SD = 0.69$) second and SDAZ third (*Satisfaction:* $M = 2.39, SD = 0.55$, *Usefulness:* $M = 2.53, SD = 0.60$) . Multi-touch ranks first in *Ease* ($M = 1.38, SD = 0.60$) and *Learnability* ($M = 1.34, SD = 0.55$), with SAZ (*Ease:* $M = 2.089, SD = 0.60$, *Learnability:* $M = 2.19, SD = 0.60$) second and SDAZ third (*Ease:* $M = 2.52, SD = 0.63$, *Learnability:* $M = 2.46, SD = 0.63$).

The low *Ease* and *Learnability* ratings received by SAZ, which are not significantly different from those of SDAZ, indicate that this technique

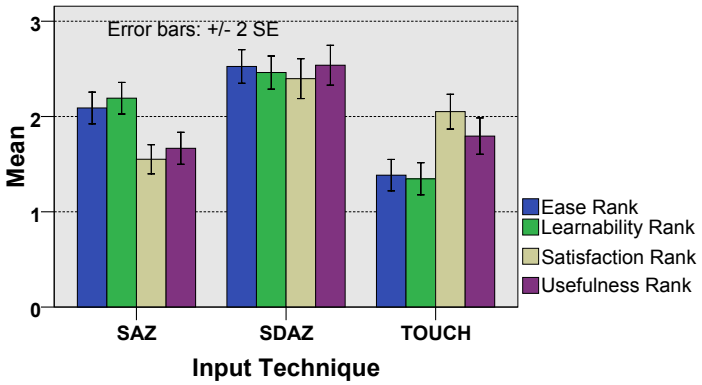


Figure 3.7: USE Questionnaire ranking results. (Kratz et al., 2010a).

is harder to learn for novice users. However, SAZ was always rated better than SDAZ and the ratings for *Satisfaction* and *Usefulness* indicate that the users perceived SAZ as being more satisfying to use and more useful than multi-touch. We believe that the ratings for *Satisfaction* and *Usefulness* would have been even more favored towards SAZ if the users had had more experience with that technique.

3.1.5 User Feedback

At the end of the USE Questionnaires for each input method, the users were asked to give positive or negative feedback, if they felt inclined to do so.

In the case of SDAZ, some users complained about "unwanted loss of control" due to the coupling of tilt with zooming and scrolling. However, users also noted some positive aspects of SDAZ, such as "map always visible" (i.e. no occlusion), "quick zooming" and "fun" to use.

Regarding multi-touch, the users liked the "precise control" they had over the map view. Frequent negative issues with multi-touch were "screen is covered" (i.e. occlusion), "slow" and "sticky fingers". It is interesting to note that multi-touch was perceived as being slower although this is contradicted by the experimental results. This perception might be due to the mental load associated with switching between zooming and panning the interface (both is not possible simultaneously using multi-touch).

The users liked the "quick zooming" of SAZ and found this input method "fun to use". They also noted that, in contrast to multi-touch, SAZ makes "one-handed interaction possible". However, the users recommended the slider to be designed in a way that it can be "locked"

at a certain zoom position without the requirement of maintaing touch contact with the slider. Also, the users commented that they would "get better with more practice" when using the SAZ interface.

The user feedback we obtained indicates that although multi-touch had the best experimental results, some users did perceive multi-touch as being a slow technique and had the impression of being faster while using SAZ. Feedback provided by the users also suggests that of the main advantages of using an SDAZ or SAZ interface is that these techniques are occlusion-free, which was appreciated by many of the users. SAZ was generally preferred over SDAZ due to the availability of manual zoom control.

3.1.6 Conclusion

a fully-functional map viewer with real map data was used in the study

In order to conduct our study comparing SDAZ, SAZ and multi-touch for mobile map navigation, we had to implement a high-performance dynamic map interface. Because we wanted to conduct our study under the most realistic conditions possible, we chose to implement our interfaces directly on a mobile device, and to use actual real map material at a high detail level covering a large geographic area with a wide range of scales. This has not been achieved in previous work in the field.

Rather than choosing an arbitrary mapping between scroll rate and zoom level, we chose to use the state-space model by (Eslambolchilar and Murray-Smith, 2008c), as we think that this model provides a more realistic usage experience due to its sophisticated physical modeling of the scrolling behavior. We had to extend the existing model to enable the support of 2D tilt input for scroll rate control.

SAZ is superior to SDAZ and performs at levels comparable to multi-touch

Our study demonstrates that for mobile map navigation, SAZ is a superior input technique compared to SDAZ. Not only was the task duration of SAZ significantly lower, SAZ was rated better in both the NASA TLX and the USE questionnaires. Notably, the results from NASA TLX and the USE Questionnaire indicate that SAZ performs at least as well as multi-touch. The user feedback and *Satisfaction* and *Usefulness* ratings from the USE Questionnaire indicate that SAZ was seen as increasing the user's productivity and effectiveness while at the same time being fun to use.

SAZ's steeper learning curve and the users' familiarity with touch could have biased the results

A possible reason that the study results are not even more in favor of SAZ is that the users had no previous experience with automatic zooming or tilt-based interfaces. Also, we have to take note that the iPhone's support for multi-touch is very mature, as this device was developed for supporting multi-touch as primary means of interaction.

This may have contributed towards biasing the study results towards multi-touch.

multi-touch was perceived as being the slowest technique

In this context, it is interesting to note that informal post-experiment discussions, several participants noted that they had perceived multi-touch as the slowest technique. In addition, several users mentioned occlusion problems when using the multi-touch interface, which weren't present when using SDAZ or SAZ. Because of the very mature implementation of multi-touch on the iPhone, it may be useful to re-evaluate SAZ and SDAZ in comparison to a non multi-touch mobile map interface such as the one found on mobile devices running Android. In this case, SAZ has the potential to be at a clear advantage.

SDAZ is a useful technique for devices not capable of multi-touch

From a more global perspective, the results of this work demonstrate that tilt-based SAZ is an efficient input technique for navigating large information spaces on mobile devices, and has the potential to be a very useful input technique for future devices that are not equipped with multi-touch-based interfaces.

3.1.7 Further Work

From the user feedback, we gained some valuable insight into the design of future SAZ interfaces. The users appreciated the manual zoom control provided by the SAZ slider, although it was apparently lacking a feature allowing the users to lock the zoom level without having to touch the slider's thumb.

In the future, it may be useful to analyze how much the users made use of the SAZ slider in individual task scenarios, i.e. in *find landmark* or *follow route*. In addition, we believe that in certain scenarios involving a search for a specific point of interest, such as in the *find landmark* task in our study, there may be certain phases of the task in which the users prefer to use automatic zooming and other phases where manual zoom control is preferred. We wish to analyze the phases of SAZ slider usage in more detail in future user studies.

3.2 Flick-and-Zoom: Touch-Based Automatic Zooming for Mobile Map Navigation

The results in Section 3.1.5 indicate that although Semi-Automatic Zooming (SAZ) is liked by the users, tilt-based navigation can be difficult to master without practice, in contrast to standard multi-touch interfaces. The idea for *Flick-To-Zoom* is to combine the advantages of

touch-based navigation with an automatic zooming model, and to re-evaluate this technique in comparison with SAZ. We[6] thus decided to evaluate a combination of State-Space automatic zooming with a touch based map interface, in order to combine the positives aspects of SAZ with a multi-touch interface.

3.2.1 Flick-and-Zoom Navigation

Flick-and-Zoom reduces the number of touch inputs

The basic idea behind Flick-And-Zoom is to use flick gestures to impart "momentum" to the virtual camera viewing the map, in the desired direction of scrolling. This similar to the naïve physical scrolling behavior of the 1D selection wheel widgets found in iOS[7]. We assumed that adding automatic zooming via flick gestures to a map navigation interface for mobile device will reduce the number of required inputs (touches) as well as decrease task completion times when compared to SAZ and touch without automatic zooming.

As discussed in the following Section in more detail, the speed of the flick gesture can either be input directly to the state-space model as an absolute scroll speed value, or in an additive variant, each flick can add more scrolling speed, in turn forcing the automatic zooming engine to adjust the zoom value.

3.2.2 Multi-Flick Gestures

For our implementation, we needed to decide which mapping to use from (multi-flick-) input to the scrolling speed and direction. (Aliakseyeu et al., 2008) evaluate a number of possible mappings for multi-flick gestures.

- **Multi-Flick-Add (MFA):** with this mapping, the scroll rate is influenced by the scroll direction as well as the scroll velocity. Successive flicks can thus be used to increase or decrease the velocity, successively. Figure 3.8 (a).

- **Multi-Flick-Standard (MFS):** in MFS, a direct mapping is performed from the flick speed to the scroll speed. In contrast to MFA, the scroll speed can not be additively increased with multiple flicks. Rather, the scroll speed and direction is always defined by the speed of the last flick. Figure 3.8 (b).

[6] Implementation and user study executed as a project thesis by Valerie Kroner (Kroner, 2011).

[7] For an excellent overview of how naïve physics can be applied to user interfaces see (Jacob et al., 2008b).

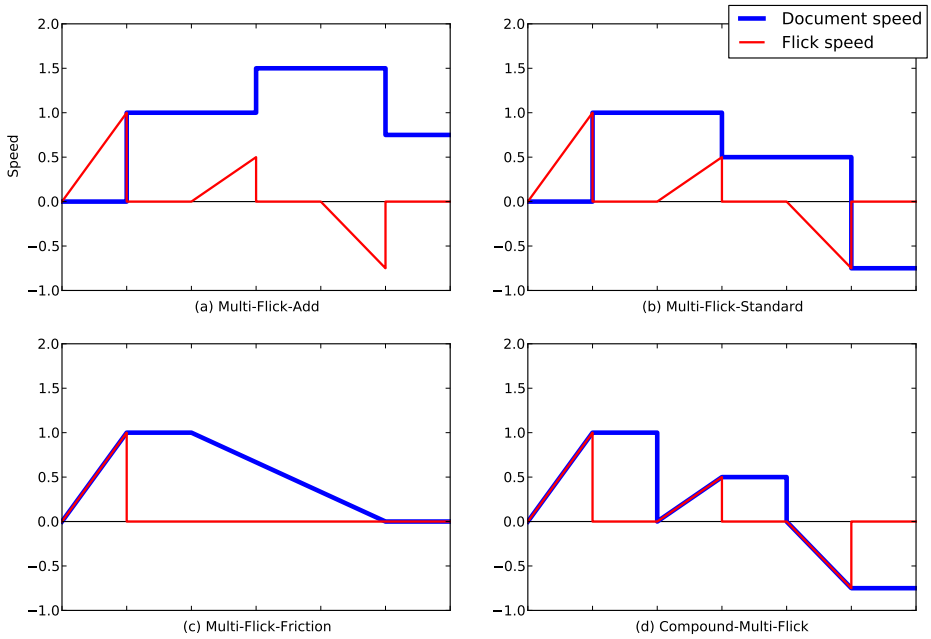


Figure 3.8: Mapping functions for the multi-flick techniques analyzed by (Aliakseyeu et al., 2008).

- **Multi-Flick-Friction (MFF):** the mapping from flick speed to velocity is essentially the same as in MFS. However, friction is added, which leads to a decrease of the scrolling speed over time. Figure 3.8 (c).

- **Compound-Multi-Flick (CMF):** in CMF, the document at first follows the displacement of the stylus or finger on the touch screen. When the stylus or finger is lifted from the screen, scrolling continues as if a MFS flick had been performed. Once the finger or stylus is set back down on the screen, scrolling ends. The advantage of this technique is that it allows precision displacement tasks when the stylus or finger is moved slowly over the screen as well as flick gestures, when the input artifact is lifted from the screen. Figure 3.8 (d).

Although we initially considered using MFA for our navigation technique, we soon realized that for 2D task MFA can be complicated to use as flicks in opposing directions can no longer directly be subtracted or added to each, as in 1D, due to difference in flick direction afforded by the 2D input space. Without quantization of the flick direction, i.e. into 8 quadrants, it is effectively too difficult for the user to properly control scrolling using MFA.

we chose CMF as input mapping for flick gestures

We thus opted to use CMF, as it provides instant feedback of the flick direction to the user (in contrast to MFS). Feeding CMF flicks into our

state-space model for SDAZ automatically adds friction to the scroll tasks, so our scroll behavior could be classified as *Compound-Multi-Flick-Friction (CMFF)*.

3.2.3 Adjustments to the SDAZ Implementation

We used the implementation of the state-space model from Section 3.1.2 as a basis for Flick-and-Zoom. Because tilt was previously used to input velocity changes, we had to adjust the State-Space model to use flick gestures as input. We also had to adjust the friction parameters in order to obtain an acceptable automatic zooming behavior from the system. The modified model is defined in Equation 3.6

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -B/M & 0 & 0 \\ 0 & 0 & 0 & -B/M & 0 \\ 0 & 0 & -B/M & -B/M & -B/M \end{pmatrix}}_{S} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ A/M & 0 & 0 \\ 0 & A/M & 0 \\ 0 & 0 & C/M \end{pmatrix}}_{U} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \tag{3.6}$$

$(\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{x}_4, \dot{x}_5)^\intercal$ is the update vector for the state-space model. The current state of the system is represented by $(x_1, x_2, x_3, x_4, x_5)^\intercal$. The speed update components u_x and u_y represent the update components of the flick movement in the x and y axes, respectively. Equation 3.8 describes how the values for u_x and u_y calculated. u_z is the zoom update, (see Equation 3.9). The parameters used for the modeling of physical behavior are M, the mass in kg, and scaling factors A, B, and C. A can be used to explicitly regulate the maximum zoom level. B regulates the amount of friction applied to the model during each state update. C controls the responsiveness of the zoom behavior. Based on experimentation and expert evaluation, we chose the following values for the parameters:

$$M = 30, A = 1.5, B = 9, C = 30 \tag{3.7}$$

In our previous SAZ implementation, the zoom friction update u_z is calculated explicitly (Equation 3.3, p.36). For Flick-and-Zoom, we incorporated the zoom friction directly in the bottom rows of the matrices S and U (compare Equation 3.6).

Instead of relying on the acceleration values for updates, Flick-and-Zoom obtains speed in x and y axes as inputs for each update step.

We calculate these speeds as follows: we calculate the movement extents of the flick gestures $(\Delta_{height}, \Delta_{width})$ by adding up the successive touch locations of the movement. The speed update components u_x, u_y are then calculated as follows:

$$u_x = \frac{s}{cos(atan(\frac{\Delta_{height}}{\Delta_{width}})) \cdot t_{flick}^2}$$
$$u_y = \frac{s}{sin(atan(\frac{\Delta_{height}}{\Delta_{width}})) \cdot t_{flick}^2} \tag{3.8}$$

where t_{flick} is the flick duration and s is scalar of the values for input into the state-space model. For our this particular implementation we obtained an acceptably working value of 1.5×10^5 for s through trial-and-error. We use an inverse-quadratic mapping of t_{flick} to mitigate the effect of long-duration flicks as they can be very similar to normal movements on the touch screen.

The zoom update u_z is calculated as the magnitude of (u_x, u_y):

$$u_z = \sqrt{u_x^2 + u_y^2} \tag{3.9}$$

If the user moves their finger over the touch screen for a very long duration, very large Δ_{height} and Δ_{width} values can result, which would lead to undefined behavior during the flick update. For this reason we introduced a threshold T_{upper} for flick duration time. If T_{upper} is exceeded, the application interprets the movement as a normal translation and not a flick. In addition, we also defined a threshold for the minimum flick duration T_{lower} to filter out minimal movements, for instance introduced by sensing noise in the touch screen when the finger is resting on the display in a stationary position. For the same reason we also only consider flicks with a minimum displacement of 10 pixels. We thus summarize the conditions for processing a movement as a flick as follows:

$$T_{upper} = 3000ms \wedge T_{lower} = 50ms \wedge |\Delta_{height}| > 10px \wedge |\Delta_{width}| > 10px \tag{3.10}$$

3.2.4 Preliminary Evaluation

We evaluated the usability of Flick-and-Zoom by comparing it to SAZ and Multi-Touch for map navigation in a preliminary user study.

Hypothesis We assumed that a combination of SAZ with Flick-And-Zoom, i.e retaining the manual slider from SAZ (Section 3.1.2.6) in the Flick-And-Zoom interface would be more natural and easier to use than SAZ (**H1**). The qualitative measurements for **H1** were obtained using the USE questionnaire (Lund, 1998). We also assumed that a combination of SAZ with Flick-and-Zoom would achieve a lower task completion time a normal multi-touch map interface (**H2**).

Test Subjects We invited a total of four test subjects to take part in the study. The subjects were between 25 and 27 years old. Two test subjects were Media Informatics students, one subject was an industrial engineer and another was a usability engineer. All test subjects owned a smart phone with a touch screen and three of them had previously used an iPad. However, none of the subjects had previously used map applications on tablet PCs. Two of the test subjects used map applications on their smart phones.

Apparatus We used a first-generation iPad, on which we implemented Flick-and-Zoom by extending the original iPhone implementation which was used for SAZ (Section 3.1.2) and modifying it to work on an iPad.

3.2.4.1 Design of the User Study

Input Techniques We tested four different input techniques for map navigation:

(A) **FLICK**: A combination of Speed Dependent Autmatic Zooming (SDAZ) and flick input (i.e. Flick-and-Zoom).

(B) **FLICK-SAZ**: A combination of Semi-Automatic Zooming (SAZ) and flick input.

(C) **SAZ**: Semi-Automatic Zooming (from Section 3.1).

(D) **TOUCH**: The standard multi-touch map interface found on the iPhone.

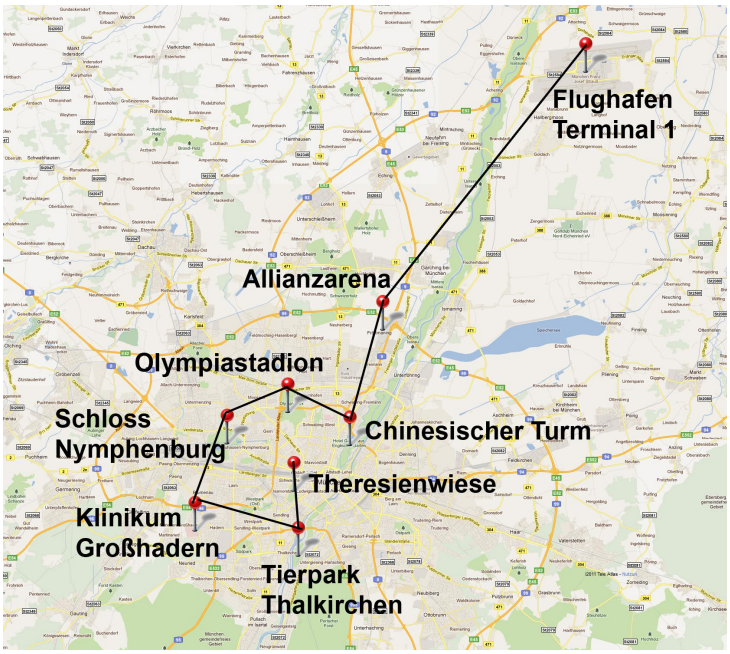


Figure 3.9: The poster containing the POIs and the suggested route which was provided to the test subjects during the study. Most of the POIs are famous tourist destinations in Munich, Germany. (Kroner, 2011).

Tasks The task the users had to perform were of the *LandmarkFinder* type used in Section 3.1.3.2. The users had to use the map interface to find particular points of interest (POIs) located on the map of Munich, Germany.

8 POIs in Munich are marked with a red map marker. Most of the POIs are famous tourist attractions in the city. Table 3.2.4.1 shows the specific POIs used in the study. The starting point for each task is the Munich Central Train Station Trials are initiated by touching the starting marker. It then changes its color to blue and the remaining number of POIs to touch are displayed on the screen.

we mitigated the effects of prior cartographic knowledge

To mitigate the effect of persons who are very knowledgeable about the city map, all test subjects could use for guidance a printed out map of Munich containing all the POIs. In addition, the test subjects had access to a poster (Figure 3.9) of a possible navigation route through all POIs, which showed the markers and with a line connecting them, suggesting the possible route. Most of the test subjects adhered to the suggested route.

a warmup trial was performed for each technique to acclimatize the test subjects

In order for users to get acquainted with each input technique, the users were asked to complete a warmup trial consisting of only three POIs before commencing the main trial for that input technique.

Table 3.2: Points of interest used in the warmup exercise as well as the main part of the user study.

	Warmup	Study
Starting Point	Central Station	Central Station
POI 1	Hirschgarten	Theresienwiese
POI 2	Deutsches Museum	Tierpark Thalkirchen
POI 3	Riem Arcaden	Klinikum Großhadern
POI 4	-	Schloss Nymphenburg
POI 5	-	Olympiastadion
POI 6	-	Chinesischer Turm
POI 7	-	Allianzarena
POI 8	-	Flughafen Terminal 1

Trials We chose a within-subjects design for the user study. Thus all participants performed the study with each of the input techniques. The order of the input techniques was randomized in order to counter learning effects. The ordering of input techniques was ABCD, CDAB, DCBA and BADC for users 1,2,3 and 4, respectively.

A USE questionnaire was presented to the users at the end of each trial. After completion of all four techniques, the users were presented with a final questionnaire asking them to rank the techniques according to the questions found on the USE questionnaire.

Adjustments to the Application Prior to the Experiment We conducted a pretest with two Media Computing students prior to the main experiments. As a result, we reduced the geographic area of the map data to Munich and its surroundings with a radius of approximately 150km. This was done in order to prevent test subjects with low cartographic familiarity of Munich from completely losing their orientation on the map.

minimum zoom level was limited to level 12

We also introduced limits to the zoom level. Because the POIs used in the study are located in and around Munich, we limited the minimum zoom level to Google Maps level 12. At this zoom level the entire geographic area covered by the experiment's tasks, Munich and its surroundings, are visible completely on the iPad's display.

At very high zoom levels (i.e. 18-19), the display becomes too detailed for orientation via geographical features. Such zoom levels are more suited to gaining specific data from the map such as street names or the outlines of individual buildings. Because the POIs are clearly visible at lower zoom labels we did not want to overload the test subjects with unnecessary information, so we restricted the maximum zoom level to 17.

the maximum zoom level was set to 17 in order to omit details not necessary for task completion

3.2.5 Results

We evaluated the quantitative data obtained from the log files of the iPad application and also qualitative data obtained from questionnaires given to the test subject as well as comments the test subjects made during the experiment.

3.2.5.1 Quantitative Results

The average task execution times for both the experiment and the warmup exercise are shown in Figure 3.10. On average, TOUCH and SAZ performed better than FLICK or FLICK-SAZ. Due to the low number of users, however, generalizations are difficult. For instance, FLICK was the fastest technique for two of the test subjects. In general, the task execution times varied more for the flick techniques, lying between 3.2-4.96 minutes for FLICK and 2.86-4.94 minutes for FLICK-SAZ compared to 1.65-2.23 minutes for TOUCH and 2.87-3.61 minutes SAZ. Regarding the two FLICK techniques, FLICK-SAZ was the faster technique for 3 participants.

In the warmup exercise, TOUCH had the lowest average task completion time. FLICK-SAZ performed better than FLICK or SAZ, which can be explained by the higher learning curve inherent to SAZ (Section 3.1.6).

When looking at the number of screen touches counted during the experiment, we can observe that FLICK and FLICK-SAZ do reduce the number of average screen touches from 121 for TOUCH to 78 and 74, for FLICK and FLICK-SAZ, respectively (Figure 3.11). This amounts to a reduction of almost 40%. Because SAZ is mainly tilt-based, it we only counted an average of 11 touches for that technique.

FLICK and FLICK-SAZ lead to a reduction in screen touches compared to TOUCH

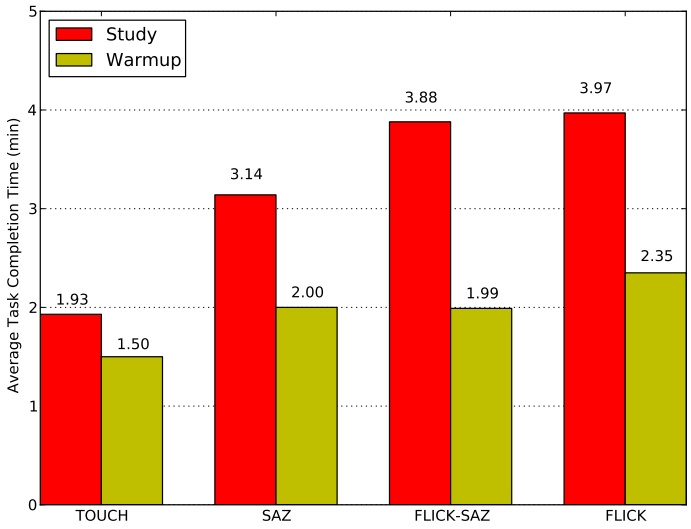


Figure 3.10: The average task completion times in minutes for each input technique during the experiment and during the warmup exercise.

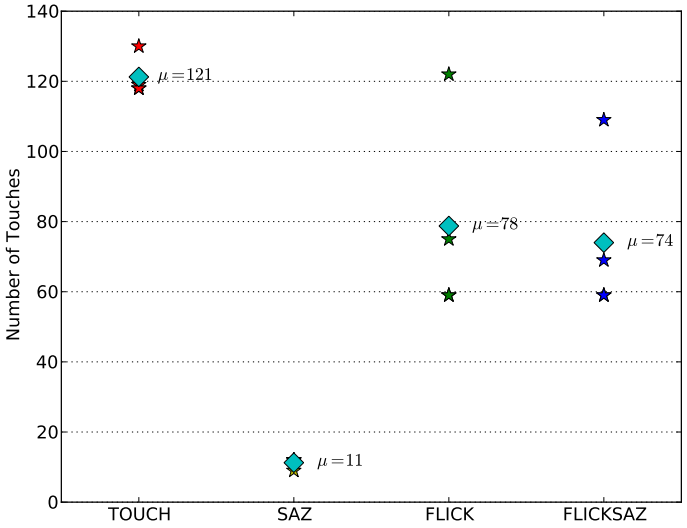


Figure 3.11: The number of touch events measured for each input technique. The diamonds represent the average number of touches over all participants for the respective input technique.

3.2.5.2 Qualitative Results

Figure 3.12 shows the average ratings and rankings for the USE questionnaire. TOUCH was rated and ranked best, followed by SAZ and the flick-based techniques for *Satisfaction, Usability and Learnability*. However, FLICK and FLICK-SAZ were rated better for *Usefulness* than SAZ. This may be due to tilt-based techniques being less familiar to users than ones based on touch.

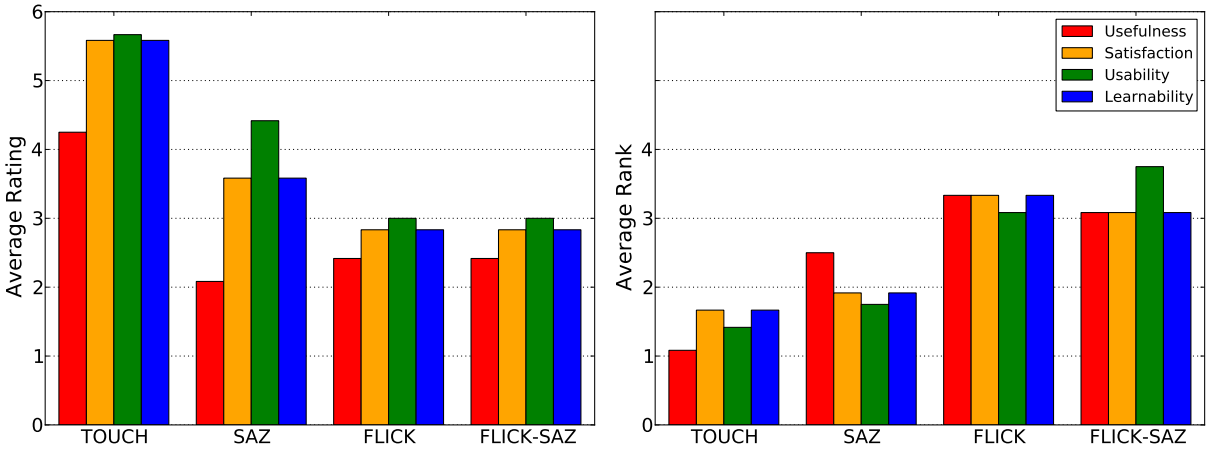


Figure 3.12: *Left:* the average USE Questionnaire ratings by input technique. *Right:* the average USE Questionnaire rank by input technique. Error bars are omitted due to the low number of study participants (N=4).

3.2.6 User Feedback

During the user study, the test subjects were encouraged to express their thoughts, impressions and actions orally. In the following, we summarize the most common items of spoken user feedback:

- All users preferred TOUCH as this is the technique that they were the most familiar with, since this technique is standard on most multi-touch mobiles. Because of their knowledge of how the interface would react, the users felt that they had more control over the map and that they were more effective. However, the users did criticize the large number of inputs required to reach each target. Also, several users mentioned that repeated pinch-to-zoom input was “annoying and slow”.

- Most users mentioned that the tilt-based navigation using SAZ was easy to learn and fun. Due to the direct translation of the tilt movement to scroll direction and zoom level, this technique was easy to grasp for most users. By contrast, the users found that the zoom-out and zoom-in speeds were too fast. Although the users mostly liked SAZ during the warmup exercise, some users got annoyed during the main task and found that “flying on the map” was fun but not a real alternative to a standard user interface, since it wouldn’t be appropriate in certain everyday situations, i.e. in business settings.

- FLICK and FLICK-SAZ obtained positive mentions for providing a good overview of the map when only the coarse direction of the target is known. The advantage versus TOUCH is that the map

keeps moving once scrolling has been initiated, and that the map can be seen scrolling without occlusion from the user's hand. Additionally, all test subjects liked the possibility to stop scrolling and zoom in fully using a single touch on the map's surface. However, the users would also have liked the ability to stop scrolling while maintaining the current zoom level, which could be easily incorporated into the existing application.

- Initially, all users did not feel that FLICK-SAZ was intuitive. With time, however, all subjects developed skills to interact using this method and mentioned that the slider would allow them to get an overview of the map position and the scrolling direction. The standard interaction sequence that evolved led to the users performing a fast flick in the desired direction and then controlling the slider to select the preferred zoom level. The scroll direction was then corrected by inputing additional flicks. The ability to "skip" to the target location directly by tapping on it was used often. The ability to manually override the automatic zooming using the slider was mentioned favorably.

3.2.7 Discussion and Future Work

In this section we compared Semi-Automatic Zooming (Section 3.1) with additional flick-based techniques for (Semi-) Automatic Zooming. The assumption was that flick-based input would be easier to use by the users than SAZ.

For the flick-based technique FLICK, we used flicks as inputs to a state-space SDAZ model to control zooming. The second technique, FLICK-SAZ, was based on SAZ, but used flick input to control the scrolling direction, as well as slider for manually overriding the zoom level.

(**H1**) and (**H2**) could not be validated—adjustments to the implementation may be needed

Against our expectations, FLICK and FLICK-SAZ performed worse than SAZ and the standard multi-touch interface TOUCH (**H1**). Although the flick-based techniques led to a reduction in the number of touch events (Figure 3.11) we observed neither a reduced task completion time nor a better rating or ranking in the USE questionnaire for these techniques (**H2**). A possible reason for this are weaknesses in the implementation of FLICK and FLICK-SAZ as the users criticized the speed and duration of the user interface behavior in response to flick gestures. Some users found the response too quick, some too slow. It seems likely that further calibration of the mapping of flick input to scrolling is needed. Also, the speed updates u_x and u_y (Equation 3.8) needs to be modified to incorporate the length of the flick gestures as well as its execution time.

Furthermore, the preliminary study presented in this section needs to be repeated as a full user study with the suggested improvements. Because of the low number of participants (N=4), the results presented here can only be considered indicative and not statistically significant.

a combination of FLICK and TOUCH should be considered for future studies

The user feedback also suggests that dynamic zooming behavior may not be needed at all times. A further technique that should be studied in the future is a combination of FLICK and TOUCH. In this variant, the UI would exhibit normal scrolling behavior for slow and short touches, allowing precision manipulation. For faster and longer touches, i.e. flicks, the UI would scroll using speed-dependent automatic zooming, as in FLICK. This hybrid technique would thus combine the familiarity of TOUCH with the advantages of FLICK.

3.3 Summary

In this Chapter, we introduced SAZ and Flick-and-Zoom, two input techniques designed to help users navigate mobile map interfaces more effectively. Our initial goals of reducing the number of user inputs have been achieved by SAZ and Flick-and-Zoom, but our expectations on their usability gains have only been partially met.

SAZ is a viable technique whenever multi-touch is not available or when provided for expert use

We have shown that SAZ significantly outperforms SDAZ, but does not outperform multi-touch. One of the problems is that tilt-based continuous input is not intuitive for every user and also that it has a significant learning curve. We suggest conducting a longitudinal study to analyze this learning behavior. SAZ could certainly outperform a multi-touch when used by an expert, as has been demonstrated by some of our test subjects. What should be noted is that SAZ is a preferred technique for devices that do not have a multi-touch enabled screen, i.e. not supporting *pinch-to-zoom*.

In order to evaluate if automatic zooming for mobile maps can be realized without tilt-based input (which can be difficult for some users to master), we implemented a flick-based (Semi-)Automatic Zooming approach, *Flick-and-Zoom*. The preliminary user study we conducted to evaluate this technique showed that there still are some issues that need to be addressed, such as the calibration of the parameters for the state-space model.

Flick-and-Zoom received promising comments but requires further refinement and a full-scale user study

The results of the preliminary user study of Flick-and-Zoom should be regarded as indicative. We obtained numerous suggestions for improvements of future versions of this technique, and we still need to conduct a full-scale user study of this technique. As with SAZ, our results indicate a steeper learning curve using Flick-and-Zoom. This

may be a general property of all user interfaces using automatic zooming, because the users need to generate a suitable mental model of the system in order understand the effects their inputs—a *Gulf of Evaluation* in Norman's sense (Norman, 2002). The positive feedback obtained by the study participants does make us confident that we will in the future be able to develop a Flick-and-Zoom-based mobile map navigation interface that outperforms the current standard UI for map navigation.

In summary, the results of our exploration of mobile map navigation interfaces based on continuous input and a state-space models that automatically control certain parameters such as scroll speed and zoom level, show that the performance of multi-touch has been surpassed by neither SAZ nor Flick-and-Zoom. However, on devices lacking multi-touch, techniques such as SAZ can offer a real advantage over normal user interfaces (i.e., which use buttons and digital joysticks for navigation). Tilt-based techniques seem to have a steeper learning curve than touch, but our results indicate that users tend to perform very well once they have understood the concept.

Chapter 4

Around-Device and Sensor-Based Interaction

"Basically, an input device is a transducer from the physical properties of the world into logical parameters of an application."

—Ron Baecker and Bill Buxton

novel user interface concepts need to be developed for sensor–equipped mobile devices

A generally observable trend over the past 10 years is that new generations of high–end mobile devices are being equipped with more types of embedded sensors than previous generations. In the previous chapter, we looked at possibilities of using the existing sensors in mobile devices to develop novel mobile user interfaces. In this chapter, we explore several types of additional sensor technologies that can be used to implement novel mobile user interfaces. This exploration enables us to draw conclusions about what types of sensors could be used in future mobile devices in order to improve the usability of those devices and to enable novel interaction concepts.

This chapter is structured as follows. Section 4.1 introduces the concept of *Around-Device Interaction*, a sensor–driven approach to expanding the interaction capabilities of mobile devices. The design space afforded by Around-Device Interaction is detailed further in Section 4.2. *HoverFlow*, discussed in Section 4.3, is a prototype featuring Around-Device Interaction that uses a small number of simple distance sensors. *PalmSpace* extends the concept of HoverFlow, allowing more fine-grained interaction using a depth camera (Section 4.4). In Sections 4.5 and 4.6 we examine pressure-enabled dual-sided multi-touch interaction using the *iPhone Sandwich*, a prototype that we developed.

4.1 Around-Device Interaction

The idea behind Around-Device Interaction (ADI) is to expand a mobile device's interactive space on and beyond the device's physical boundaries to permit richer and more expressive forms of interaction. There has been relatively little previous work in Around-Device Interaction (ADI) for mobile devices. ADI is of particular interest for mobile interaction, as the size of mobile devices—and thus their interaction possibilities—is limited.

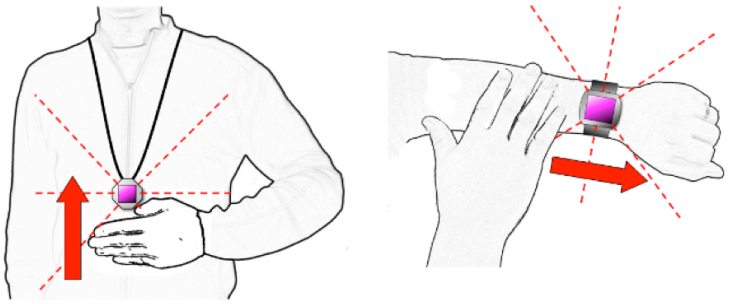


Figure 4.1: Interacting with very small devices via coarse gestures. The gestures are detected by an array of proximity sensors extending in radial direction from the device. (**?**).

sensors enable expanding the interaction space beyond the physical boundary of the device

Using sensors, the interaction space of small mobile devices can be extended beyond the physical boundary of mobile devices to include the full 3D space around them. Around-device interaction can be a beneficial addition to standard interface elements of mobile devices, such as keypads or touch screens. This is particularly attractive for very small devices, such as wristwatches, wireless headsets, and future types of wearable devices such as digital jewelry (Figure 4.1). With these kinds of devices, it is extremely difficult or even impossible to operate small buttons and touch screens. The space beyond the device, however, can easily be used, no matter how small the device may be. Such wearable devices can also serve as easily accessible controllers for appliances in the environment or for wireless communication applications.

In a smart home environment, for example, a gesture tracked by the device could dim the light or control the volume of the entertainment system. In mobile use scenarios, an incoming call could casually be forwarded to the voice mailbox or an incoming message could be acknowledged using different gestures. For mobile phones or tablet PCs—whether handheld, placed on a table, or placed in a cradle in the car—ADI could open up a range of 3D interaction possibilities. Coarse movement-based gestures could control tablet applications, such as turning pages in an electronic book. In a calendar application moving to the next day or month could be controlled by specific gestures, such

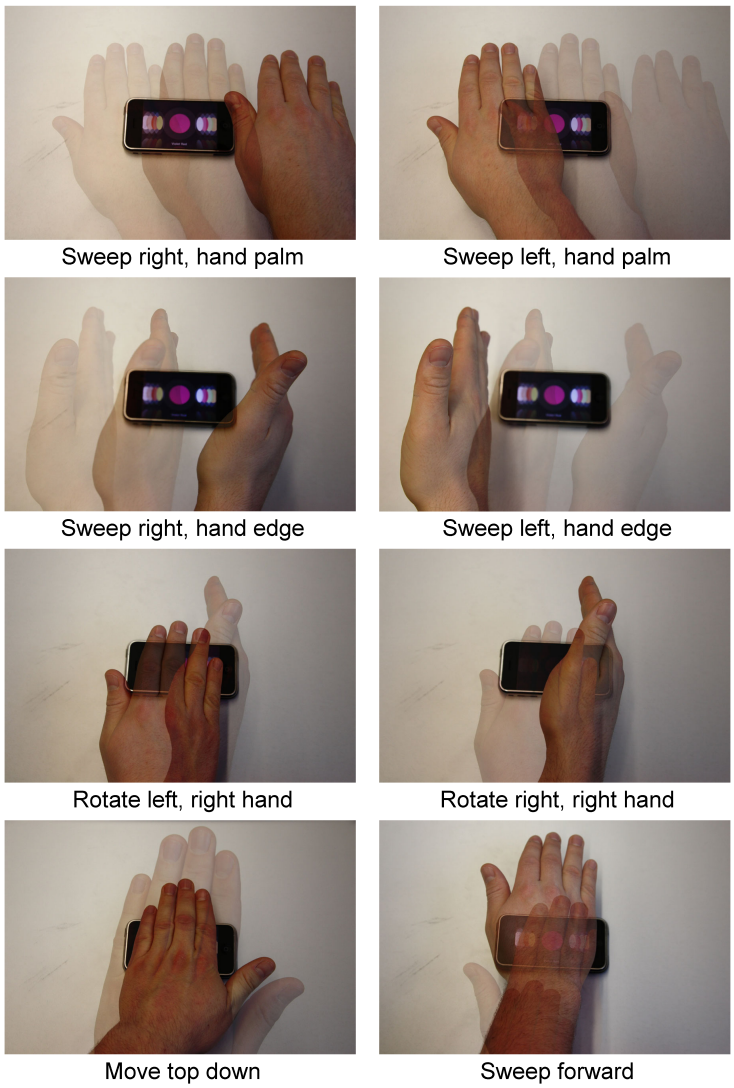


Figure 4.2: An overview of the hand and finger gestures that can be recognized by the HoverFlow prototype (see Section 4.3). (**?**).

as sweeping with the palm or with the edge of the hand, respectively (Figure 4.2).

Such coarse gestures do not require the activation of a user interface widget and can be executed without visual feedback. This is especially beneficial for devices for which command selection via visual feedback is difficult, because the device is not in the line of sight, such as digital jewelry or wireless headsets. More fine-grained gestures could have a natural spatial mapping to 3D objects on the screen. Moving the hand closer to the device or rotating the hand could be mapped to zooming along the z-axis or rotating 3D objects. In order to mitigate occlusion, such gestures do not necessarily have to be performed on top of the device display. If infrared proximity sensors are used, they can be ori-

ented in such a way, that the interaction does not occlude visibility of application objects on the screen.

4.2 The Design Space of Around-Device Interaction

with basic sensors, ADI can be used for coarse input at a distance

Using simple sensors, ADI allows for quick and coarse interaction with the devices, in cases where the desired actions are so simple that fine-grained interaction with the device's keypad or touch-screen is not necessary, or use of the mobile device at a distance is required.

For example, simple hand gestures may present an alternative to clicking the back, forward or reload buttons in the device's web browser. Similar functionality could be implemented to control playback of songs or movies with the device's media player. The detection of coarse hand gestures can also be beneficial in cases where the user needs to deliver a very quick input to the device, such as muting the device or answering a call. A simple hand gesture here is presumably quicker than getting the device's screen into focus, locating the appropriate button and coordinating the button press.

Although gestural input breaks the metaphor of direct manipulation (Butler et al., 2008b) when the gestures are symbolic, quick hand gestures may be particularly useful for tasks of an immediate and direct nature. Also, situations where visual interaction is not preferable, for example when driving vehicles, may benefit from interfaces that allow the input of simple commands using rough hand gestures.

using more advanced sensors, the detection of complex gestures becomes possible

Given a high enough sensing precision, the entry of non-symbolic gestures becomes feasible as certain physical characteristics such as the rotation of the user's palm (see Section 4.4) can be obtained from the sensor data. In this case, the precision and variance of the data is low enough to allow direct mapping of certain gesture characteristics to properties of user interface elements.

In the following, we shall characterize some of the elements of the design space of ADI-based interfaces. We use the term "design space" in a very broad sense, including elements that we deem to be important to the fidelity, usability and development of ADI-based interfaces.

4.2.1 Sensors

ADI is highly dependent on sensors that allow the mobile device to locate the position of the user or her hands and fingers. In the following,

we describe four key sensor technologies that can be used to implement ADI-based user interfaces: distance sensors, depth imaging, pressure sensing and magnetic tracking. Table 4.1 summarizes the advantages and disadvantages of the sensor types discussed in this section when used for ADI.

4.2.1.1 IR Distance

IR distance sensors are already embedded in most touch-screen smart phones for presence detection

IR distance sensors are a popular choice to sense user proximity in mobile interfaces. They are built in to most touch-screen based smart phones to sense if the device is pressed against the user's head in order to shut down the touch screen backlight during telephone calls. The advantage of IR distance sensors over ultrasound range finders, for example, is that multiple IR distance sensors working in unison show much less interference than ultrasound sensors. On the other hand, the coverage area of IR sensors is usually narrower than that of their ultrasound counterparts.

Obviously, the fidelity of ADI increases with the number of sensors. Technological miniaturization may in the future allow for the development of very small sensors. Placed in significant numbers on the device, such miniature sensors would allow mobile device to gain a relatively high-resolution "image" of its surroundings. Covering the device in printed organic distance sensor circuitry has also been envisioned (Butler et al., 2008b).

However, since energy consumption on mobile devices must be kept at a minimum, each increase of the amount of sensors will come at a cost. Not only do the sensors themselves consume energy, but with an increasing number of sensors that are mounted on a mobile device, the device's CPU will have to become active more often to process data due to the increased supply, rather than remaining in a standby mode.

novel interaction concepts can be realized by clever placement of IR distance sensors

Sensor placement is thus an important design decision. Sensors should be placed on locations on the device that allow them to optimally track the features (e.g., hands) of the user that are used for interaction with the device. As demonstrated by Butler et al., one possible useful placement of IR distance sensors is on the edges of the device facing outwards. This allows the device to track the presence of the user's fingers when the device is placed on a flat surface. *HoverFlow* (Section 4.3) demonstrates a set-up using sensors facing upwards from the device allow it to track the motion of the user's hand using information from only six IR distance sensors. An even more significant advantage of HoverFlow-like interfaces is that they do not require the device to be

placed on a flat surface in order to operate correctly. *LucidTouch* by (Wigdor et al., 2007c) demonstrated a technique enabling a mobile device to track the presence of the user's fingers to the rear of the device, based on IR distance sensors placed on the device's sides.

4.2.1.2 Depth Imaging

Depth imaging cameras, or "depth cams" provide a 2D array of pixels that each contain a depth value corresponding to the distance of the object in the scene from the imager of the depth cam. In comparison to IR distance sensors, which we discussed above, depth cameras provide a much larger amount of information with a higher precision.

depth cameras will enable complex around-device interactions

Although depth cameras are currently too large to be incorporated into mobile devices, it is plausible that in the future, miniaturized depth cameras could be incorporated into mobile devices. On mobile devices, they will enable much more complex interactions in the around-device interaction space. Depth cameras on mobile devices will allow very rich around-device gestures, as depth cams cannot only be used to accurately sense the position of the hand relative to the mobile device, but also discern between different hand poses, i.e. clenched, open, and also individual the positions of individual fingers. Users will thus be able to perform gestures with finger-level granularity. A limitation of depth cameras is that they can only observe the front-facing surface of objects within their view. This may limit the usefulness of depth cameras to recognize gestures where one or several of the user's fingers are occluded and thus cannot be observed in the depth image.

In Section 4.4 we present *PalmSpace*, with which we aim to prototype a possible use of mobile depth cameras for around-device interaction. By tracking position of the user's outstretched palm, we implemented a direct-manipulation interface for rotating 3D content on a mobile device.

4.2.1.3 Rear-of-Device Interaction and Pressure

one of the benefits of rear-of-device interaction is that it avoids occlusion

It is beneficial to make the largest possible area of mobile device interactive. This is especially important for very small devices, where a fingertip will occlude most of the screen's content. (Baudisch and Chu, 2009) demonstrated the utility of having a touch-sensitive surface on the device's rear, allowing relatively complex interactions, such as playing a 3D shooting game, on a device with a very small form-factor.

The *iPhone Sandwich* prototype (Section 4.5) we developed adds pressure-sensing to back-of-device interaction. The advantage of

adding pressure is that the users can command a very high amount of local degrees of freedom using variations of pressure at one or more points on the device's front or rear side. Furthermore a number of user interface metaphors become possible with pressure input, such as pinching an object to deform it, or requiring a higher amount of pressure for critical operations such as delete commands.

the iPhone Sandwich supports a large number of stroke and pressure gestures

Back-of device interaction allows for a large number of possible gestures to be performed by the user. Figure 4.3 shows a basic taxonomy containing a number of possible gestures using a device that supports back-of device interaction. The taxonomy also includes gestures that are based on pressure input. Most of the dual-sided gestures consist of one or more strokes where one touch on either side of the device is held stationary. This is because, in our experience, it is difficult to coordinate strokes on both sides of the device at the same time. Thus, the only dual-sided gestures with simultaneous strokes on both sides of the device feature strokes that either move in the same direction or in opposite direction (see Figure 4.3—"Simultaneous Finger Movement on Front and Back"). The taxonomy shows three useful types of pressure gestures. Pressure can be applied at the same point on the back and front of the device (*pinch*), or be coupled with a movement, either moving away from a central point or towards a central point (see Figure 4.3, *Pressure*). Pressure can also be used in conjunction with taps on the touch screens, which allows for many different combinations, as shown in Figure 4.3, *Taps and Pressure*.

The taxonomy shown in Figure 4.3 is not meant to be exhaustive, but aims to illustrate the numerous possibilities offered to developers of gesture-based mobile user interfaces when rear-of-device input in addition to pressure sensing is available. However, future user studies will need to be conducted in order to analyze the usability characteristics of gestures we propose.

4.2.1.4 Magnetic Tracking

Magnetometers can be used to sense the direction and intensity of magnetic fields, i.e. as emitted from magnets held in the vicinity of the device. Since magnetometers are present on most high-end smart phones and external magnets do not consume additional power, magnetic tracking is a useful technique for implementing around-device interaction.

There has been significant related work in this area. Abracadabra (Harrison and Hudson, 2009b) uses a magnetometer and a ring-mounted

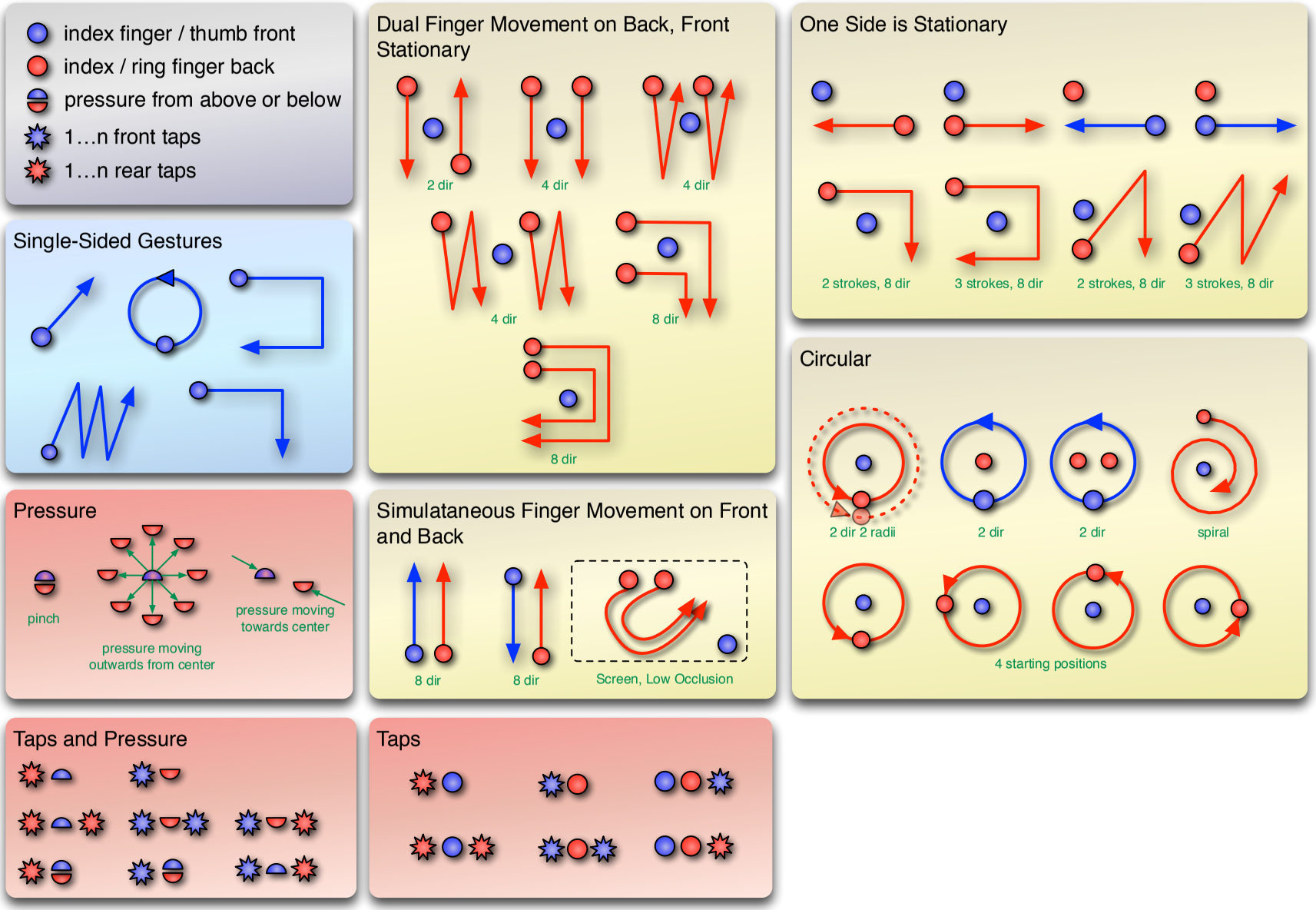


Figure 4.3: A taxonomy containing a number of possible dual-sided multi-touch gestures with support for pressure input.

magnet to enable around-device interaction for very small devices. Harrison states an effective range of 10 cm for their sensor, dependent on the strength of the magnet used. This paper describes methods of implementing selections via in-air click gestures (using the dipolar properties of magnets) as well as simple gesture recognition. Harrison summarizes the advantages afforded by around-device interaction as follows:

> "By extending the input area to many times the size of the device's screen, our approach is able to offer a high C-D gain, enabling fine motor control. Additionally, screen occlusion can be reduced by moving interaction off of the display and into unused space around the device."

MagiTact (Ketabdar et al., 2010b,a) used the built-in magnetometer on an iPhone in combination with a ring-mounted magnet to enable around-device gesture recognition as well as simple stroke recognition using a stub-shaped magnet held between thumb and index finger.

Nenya (Ashbrook et al., 2011) is a wrist-mounted magnetometer coupled with a magnetized, rotatable ring. The magnetometer is able to detect the rotation of the ring's magnetic field as the ring is rotated, and also its field strength. Rotating the ring can thus be used for selection tasks, and varying the distance of the ring to the accelerometer can be mapped to a "click" event. The advantage of Nenya is that it is more subtle than the previous ring-based approaches, as the magnetic input artifact is manipulated locally on the finger, and no further hand movement is required for interaction.

Magnetic tracking for around-device interaction is elegant, because it can use already built-in sensors of mobile devices. Furthermore, magnets are passive and have no power requirements. An obvious disadvantages of magnetic tracking is that the user needs to be instrumented with a magnet (via a ring or a stylus). Furthermore, magnetometers can only track a single magnetic object, so detecting multiple fingers, for instance, is out of the question.

4.2.2 Mapping of Sensor Data to Interface Actions

Useful mappings of the sensor information to interface elements need to be developed to make the best use of the available sensor data. *HoverFlow*, for instance, demonstrates how to use the data from IR distance sensors to recognize simple hand gestures. In that project, the data was input into a gesture recognizer in order to recognize simple hand gestures. However, for other applications alternative mappings

Sensor	Advantages	Disadvantages
Pressure (FSR)	• low cost • adequate precision • low power consumption	• large sensor size • difficult to incorporate mechanically into device • sensing only on surface of device
Magnetometer	• 3D coverage • can cope with obstructing materials such as clothing • adequate precision • already embedded in certain mobile devices	• can only sense location of a single (magnetized) object • susceptible to electromagnetic interference
IR Distance	• numerous options for integration on device • high precision • low-cost (as surface-mount component)	• high power consumption • requires specialized driver chip • senses only within a narrow beam
Depth Camera	• high precision • high detail image of surroundings • enables complex gestures • detailed tracking of device surroundings	• high cost • medium to high power consumption • size • processing depth images requires significant amount of computational resources

Table 4.1: Comparison of several sensor types for Around-Device Interaction.

may be more advantageous. Currently, our system can only effectively discern a single user action from sensor noise. An interesting improvement of HoverFlow would be the capability to identify a sequence of gestures performed in a single user action (for instance the gestures "rotate-right" followed by "rotate-left" and "sweep left" performed in close succession, appearing as a single gestural phrase to the user (Buxton, 1986)).

A further example of a useful sensor mapping is the one used by (Butler et al., 2008b) in *SideSight*. They use the sensor readings of their prototype to enable multi-touch like user inputs on the sides of a mobile device. They map their sensor readings to a one-dimensional bitmap, from which the finger position and estimated distance can be inferred. In *LucidTouch*, the user's fingers are located on the rear of the device (Wigdor et al., 2007b). The camera image of the fingers is mapped to the device's screen in the form of finger shadows.

PalmSpace (Section 4.4) shows a mapping depth image data of the user's palm to 3D object rotation. One of the constraints here was the movement range of the human hand. The range of pronations, supinations and flexions of the human hand is limited (see Section 4.4.5.3), and was taken into consideration when developing the mapping for this interface.

4.2.3 Feedback

Since the direct manipulation metaphor is broken due to the interaction taking place away from the device, feedback plays an important role for the usability of the interface, as it will help users operate ADI interfaces more effectively. In addition to visual feedback mechanisms, it may be feasible to use vibrotactile feedback (i.e. using the mobile device's built-in vibrator motor), if the device is held in one hand while the other hand performs the interaction. In a similar way, auditory feedback could provide feedback on the status of the gesture recognition, i.e. playing back a notification when a gesture has been recognized.

4.2.3.1 Around-Device Output: the Mobile Ambilight

So far, we have mainly focused on input techniques for around-device interaction. It is of course also beneficial to have output techniques for ADI. Apart from, i.e., mobile projection, ambient lighting can be used to output feedback in the near surroundings of the mobile devices. Ambient lighting for flatscreen televisions was invented by Philips Research (Diederiks et al., 2003). Ambient lighting on televisions is used to create a more immersive viewing experience by mimicking the colors of the displayed scene through colored LED strips on the device's bevel.

To explore the concept of ambient lighting on a mobile device, we[1] developed the *Mobile Ambilight* prototype. The Mobile Ambilight is based on a Google Nexus One Mobile phone for which we developed a custom back panel equipped with a PCB containing a microcontroller, 40 RGB LEDs and two IR distance sensors. A Bluetooth module is used for communication with the mobile device. The LEDs can be updated at a rate of 60fps. Figure 4.4 shows an image of the PCB we developed. When used for visual output, the Mobile Ambilight increases the displayable area of the mobile device by 1664%, albeit at a very low resolution due to the use of discrete LEDs.

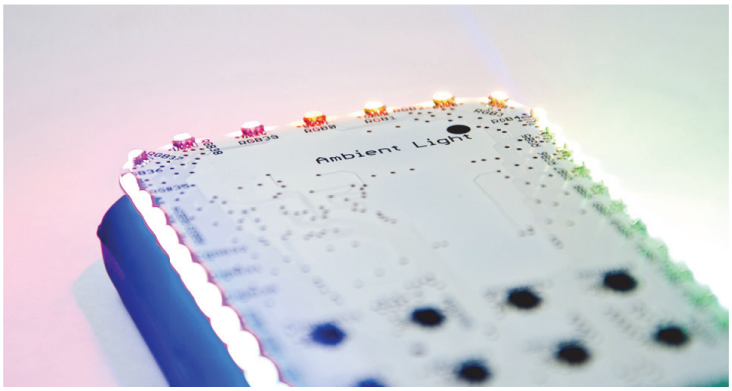
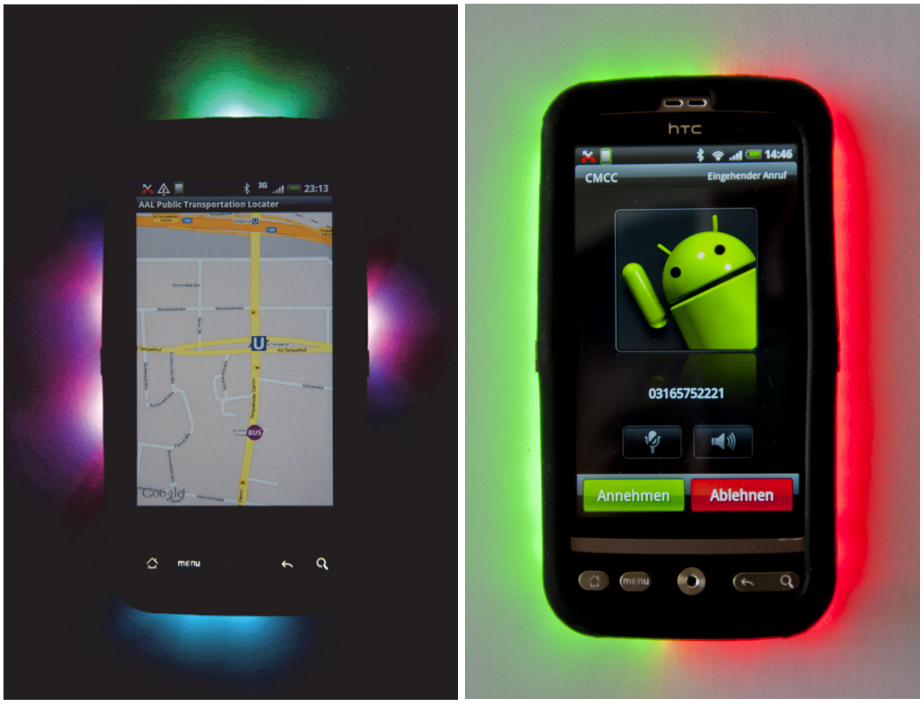


Figure 4.4: The Mobile Ambilight PCB mounted on the rear of a mobile phone, showing part of the 40 RGB LEDs. (Qin et al., 2011).



(a) Public Transportation Locator **(b)** Call Detector

Figure 4.5: The demonstration applications developed for the mobile ambilight. *(a)* Public Transportation Locator. *(b)* Call Detector. (Qin et al., 2011).

Public Transport Locator allows off-screen visualization of transport stops

Two demonstrator applications were developed for the Mobile Ambilight: *Public Transport Locator* and *Call Detector*. Public Transport Locator (Figure 4.5 (a)) helps the user find nearby public transport stops. The application uses the Mobile Ambilight to indicate off-map stops by indicating their presence using the Ambilight's LEDs. The position of the

[1]The hardware and software for the prototype was developed by Qian Qin in his Diploma Thesis. We presented the prototype at UIST 2011 (Qin et al., 2011).

illuminated LEDs indicates the direction of the stop, their direction the type of stop (bus, subway or suburban train) and the intensity of the LEDs indicates the distance of the stops from the map center.

Call Detector is an ambient user interface for accepting or rejecting phone calls

Call Detector (Figure 4.5 (b)) serves as an ambient user interface for accepting or rejecting calls. When a call is received the left side of the device is illuminated in green, the right side in red. The IR distance sensors incorporated in the PCB allow sensing the presence of the user's hands on either side of the device. When she places her hand on the green side, the call is accepted. The call is rejected when the user places her hand on the red side of the device.

The Mobile Ambilight shows the utility of having ambient display capabilities on mobile phones, and how this is especially useful when combined with around-device interaction, i.e., detection of the hand placement in the vicinity of the device in conjunction with Call Detector application. A limitation of the Mobile Ambilight is that it works best when placed on a flat surface. When held in the hand, a large amount of the LED illumination is not visible as there is no surface to reflect the light emitted by the LEDs. Illuminating the device's bevel explicitly or detecting the presence of the hand and mapping the output to the illuminable areas may be possible solutions to this problem.

4.2.4 Framework Support

support for ADI needs to be made available to developers through software frameworks

The majority of the interface frameworks of existing mobile devices do not yet support ADI. However, the palette of supported sensors in the large mobile platforms such as Android or Iphone OS is being continuously expanded and may well support sensors allowing ADI in the near future.

The general advantage of integrating sensor support into mobile interface frameworks is that this allows the actual processing of the sensor data to be abstracted away, allowing developers to focus on the core benefits provided by the additional sensing. An example of such an abstraction can be found in the gesture recognition frameworks provided by Android since version 2.x. The Android gesture recognition frameworks provide UI elements that receive events when touch screen gestures are recognized, as well as utilities to create and manage gesture libraries. The actual gesture recognition algorithm is not visible and of no concern to Android application developers, which makes it easy to develop gestural interfaces on a high level of abstraction.

developing design patterns for sensor-based interaction may help to identify useful abstractions

Thus it is clear that (beneficial) abstractions need to be included into existing mobile UI frameworks to leverage the numerous capabilities offered by ADI. Developing such abstractions will require careful identification and evaluation of the most useful functionalities that are provided by ADI. This could be achievable in future work by developing design patterns for sensor-based interaction that comprise common problems, solutions and exmaples of the solutions.

4.3 HoverFlow

HoverFlow is a demonstrator for above-device hand gesture recognition

HoverFlow is an example application for the Apple iPhone that demonstrates the use of a sensor-based interface for detecting coarse hand gestures above small mobile devices. The implementation of our application is partially based upon the CoverFlow example by (Sadun, 2008). HoverFlow allows the user to select colors from a color palette through hand gestures above the device's touch screen. Possible gestures are moving the hand across the device, presenting a number of hand postures, or by moving a hand rapidly towards or away from the device. Figure 3 shows an overview of all gestures currently implemented in our system.

4.3.1 Supported Gestures

The CoverFlow view provided by the iPhone's iPod application inspired the visual layout of HoverFlow. Thus, we decided to map the user's movements in the following way: if her hand moves across the device from left to right, Figure 4.2 (A), the color palette scrolls from left to right, and vice-versa (B). A hand-edge movement from left to right (C) makes the color palette scroll 5 colors to the right and vice-versa (D). A color is selected when the user moves her hand swiftly towards the device (G). A color is deselected when the user moves her hand rapidly away from the device. Rotating the hand towards the left (E) or right (F) permits the user to scroll directly to the beginning or end of the palette, respectively.

4.3.2 Interface Implementation

In the following, we will discuss how the presence of the user's hand is sensed and how the obtained data can be used to recognize hand movement gestures.

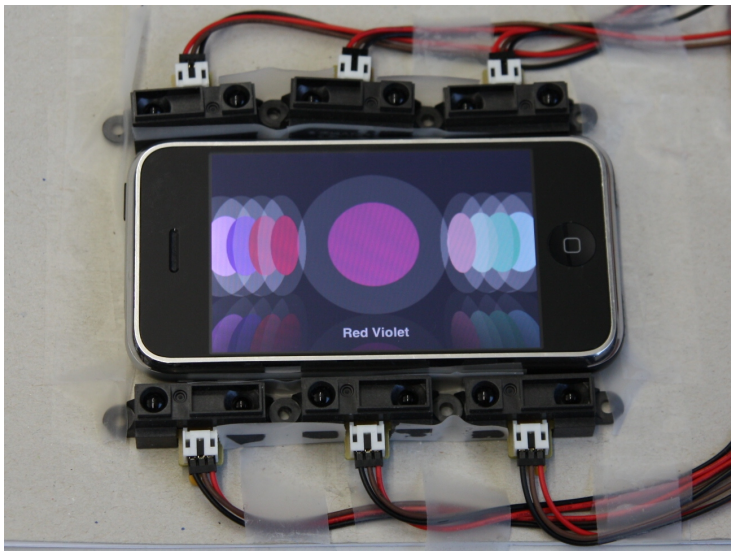


Figure 4.6: The sensor set-up of the HoverFlow prototype. Six Sharp GP2D120X IR distance sensors are placed along the long edges of an iPhone mobile device running the HoverFlow application. Using simple hand gestures, the user can scroll and select colors in the color palette. (**?**).

4.3.2.1 Sensing

To capture simple hand movements and gestures, HoverFlow uses six Sharp GP2D120X IR (Sharp, 2008) distance sensors, placed around the device's edges and facing vertically away from the device. Figure 4.6 shows the current sensor configuration of our prototype.

An Arduino BT Microcontroller board (Arduino, 2012)[W] captures the distance readings provided by the sensors. The sensors supply 256 discrete range readings allowing them to detect objects at a distance of 4 to 30 cm. The sensor update rate is 25 Hz. A PC processes the sensor data, and handles the gesture recognition. In future versions of HoverFlow, we aim to conduct all processing on the mobile device, by establishing a direct link between the Arduino board and the mobile device via RS-232 or Bluetooth.

4.3.2.2 Gesture Detection and Recognition

HoverFlow uses Dynamic Time Warping on windowed sensor data for gesture recognition

To smooth the raw sensor data, it is passed through a Savitzky-Golay filter (Savitzky and Golay, 1964b) in an initial processing step. The filtered data is then added to a queue containing the differences of the last 16 sensor readings. We use the difference values instead of the absolute values in order to make gesture recognition independent of

the distance between the user's hands and the device. The queue is updated every time the Arduino provides a new sensor reading. The window length of 16 was chosen because the sampling rate of the distance sensors is 25 Hz, which means that the system constantly keeps a history of the last 640 ms of interaction. This window length provides us with enough samples do discern user gestures in a meaningful way while at the same time assuring a response time from the system within an acceptable time interval ($<$1000 ms) for a demonstrator system.

An advantage of the method we implemented is that it does not require any clutching mechanism to detect the start and end of a gesture, which is, for example, required for accelerometer-based gesture recognition. When no IR-reflective object is present in the range of the distance sensors, they will provide a noise floor of values close to zero. Gestures can be distinguished from operations on the touch screens by checking whether the screen was touched after the distance sensors detect an object in range. If a screen touch event occurs then this activity is interpreted as touch input and the gesture (if detected) is discarded. Otherwise the activity is treated as a gesture. Gesture segmentation is more difficult using accelerometers. This type of sensor constantly provide sensor data as the user moves. It is therefore much harder, in comparison to using IR distance sensors, to distinguish between moves that are part of a gesture and those that are not.

the amplitude of the sensor readings is monitored to detect gesture input

To determine if a significant user movement has been detected, the Euclidean norm of the oldest element of the readings queue is constantly calculated. If this norm surpasses a predefined threshold, the remaining 15 sensor readings are analyzed to determine the end of the sequence representing user input. Interaction with HoverFlow is designed to take place within a certain distance range around the device, so this threshold is set to the value the sensor array provides when a large object is held in front of them at a distance of about 5-7 cm away from it.

Figure 4.7 shows image map representations of several gestures supported by our system. In each graph, time progresses from top to bottom. The numbers on the y-axis show the sample index. 30 samples are shown, which corresponds to a time span of 1200 ms. The x-axis shows one column of data for each of the six sensors. As can be seen from this visualization, states of inactivity (low-amplitude noise) can easily be distinguished from a gesture entry by looking for a point in time from which on the amplitude of the signal rises significantly.

4.3.2.3 Gesture Classification

Once the bounds of the sequence containing user activity have been detected, a best-matching gesture template from a set of prerecorded

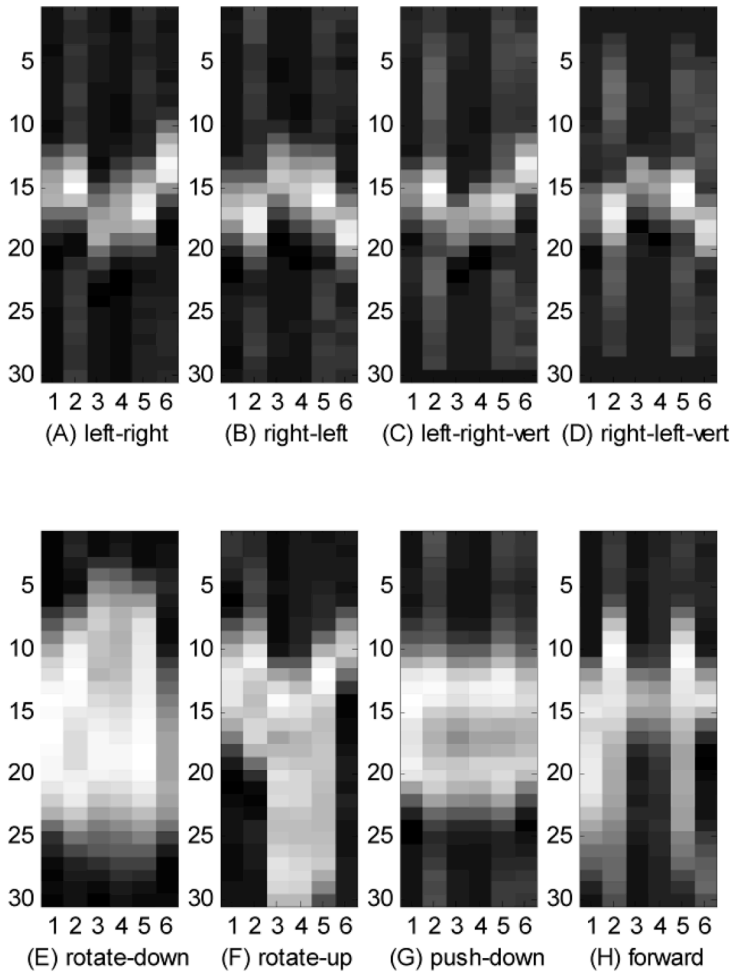


Figure 4.7: Image maps of the six IR distance sensor readings against time. The bright areas signify the proximity of an object. Notice the staggering of the peaks in the sample data, which is one of the distinctive features by which the gestures can be identified. (**?**).

user inputs is estimated using Dynamic Time Warping (DTW). Section 5.1.3.2 provides a detailed description of DTW. Gestures and templates are represented as 16-by-6 matrices of sensor values.

DTW performs well in cases where the captured sample and the matching template are distorted in time, but have similar values. In our case, using DTW allows the recognition of gestures that are similar in movement to but are performed at different speeds than the pre-recorded templates. In our prototype, we achieved acceptable recognition rates using only 2 to 3 training samples per gesture, with a gesture vocabulary of up to 9 gestures.

Because they are template-based, DTW-based approaches generally need less training samples than other methods, such as Hidden Markov Models (Liu et al., 2009b). Thus we do not require an extensive corpus

of gestures to be available in order for our prototype to function correctly. A possible drawback of the DTW algorithm, its time and space complexity of $O(n^2)$, is not an issue due to the small size of the sampling window, which results in a maximum size of the distortion matrix of 256 elements[2]. Because of the small size of the distortion matrix, CPU and memory requirements should not present a constraint for our algorithm if it is run on modern mobile devices. If, however, sensors with a higher sampling rate were to be employed, which would result in larger data sets to be processed at a time, it may be likely that further optimizations of the DTW algorithm, such as FastDTW (Salvador and Chan, 2004), will be required.

4.3.2.4 Update of Mobile User Interface

Once a gesture has been detected, the user interface of the mobile device running the HoverFlow application needs to be updated. In our prototype, the PC sends XML Remote Procedure Calls (XML-RPC) to the mobile device to signal interface updates when new gestures have been detected.

4.3.3 Evaluation of Gesture Recognition

we wanted to find the recognition behavior of several gesture types

As the initial gesture recognition performance of HoverFlow was acceptable, we decided to perform a small user study to further evaluate our interface. Firstly, we wanted to gain insight into the impact that our choice of gesture vocabulary makes on gesture recognition. We were especially interested in identifying gestures possessing similar features with respect to the gesture recognizer, i.e. leading to false positive recognitions. Secondly, we wanted to get a basic overview of our gesture recognizer's recognition rate.

4.3.3.1 Experimental Design

a training set of 3 gestures and a test set of 10 gestures was recorded for each gesture type

We conducted an evaluation with four users experienced in mobile device usage. Each user was given a brief description of our system and of the gestures it can recognize. In an initial training phase the users were asked to train the system with three samples of all the gestures shown in Figure 3, except the "sweep forward" gesture, which is a total

[2]Entry (i,j) of the distortion matrix contains the DTW-distance between samples 1 to i of the gesture and samples 1 to j of the template. Entry (16,16) thus contains our measure of similarity between gesture and template. The distortion matrix is built up from entry (1,1) using a dynamic programming approach.

of 7 gestures (A-G in Figure 4.2). After the training phase, the users were asked to enter each gesture ten times as a test set. For each gesture entry by the test participants, we recorded which gesture the interface recognized.

4.3.3.2 Experiment Results

Table 4.2: This confusion matrix shows the actual gesture entries (row) and the predictions (columns), as the number of the predictions for each gesture class divided by the total number of entered gestures for that class. The average (correct) gesture recognition rate was 88.6%. (The indices A-G correspond to the indices in Figure 4.2.)

		Predictions						
	Gesture	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Actual Input	**A**	0.775	0	0.225	0	0	0	0
	B	0.025	0.925	0.025	0.025	0	0	0
	C	0.1	0	0.9	0	0	0	0
	D	0	0.175	0.025	0.825	0	0	0
	E	0	0	0	0	0.875	0	0.125
	F	0	0	0	0.025	0.025	0.95	0
	G	0	0	0.025	0	0.025	0	0.95

In order to determine which gestures are prone to false recognition, we determined a confusion matrix using the data obtained from our study. The confusion matrix, as shown in Table 4.2, reveals that gestures A, C ("sweep right flat hand", "sweep right hand edge") and B, D ("sweep left flat hand", "sweep left hand edge") are prone to be confused by the recognizer. The similarity of these gestures can be observed in the example plots in Figure 4.7 A-D. Gestures E and F ("rotate left / right"), however, are recognized by the system in a much more stable way.

the similarity of A,C and B,D may be the cause of the high false positive rate

A possible explanation for this behavior is the relative similarity of gestures A,C and B,D. When gestures A or B are quickly executed, they may appear similar to gestures C and D to the system. This can be explained by considering the search strategy of the DTW algorithm, which aims to compensate feature differences in the time domain. In general, though, the average gesture recognition rate of 88.6% was fairly good considering only 3 samples were recorded to train the system for each user and gesture. More importantly, the low number of sensors used by our prototype. We can assume that if more densely

spaced sensors were used, then gestures A and B would cover more sensors at a time than gestures C and D, which should be easily distinguishable by the recognizer.

The study presented here is not representative of the general user population due to the limited number of participants and their high relative level of expertise. In spite of this, we gain some indication of the prototype's performance under controlled conditions. More significantly, the confusion matrix allows us to identify those gestures which are likely to be falsely recognized by the system, which is useful for the design of the gesture vocabularies of future systems employing a similar sensor configuration and using DTW for gesture recognition.

4.3.4 Discussion

HoverFlow demonstrates conceptually how mobile interfaces can expand beyond the physical interaction area of mobile devices, and can thus increase the expressivity and the physical interaction space of mobile input.

HoverFlow contributes a sensing solution as well as a set of gestures that can be used for coarse front-of-device interaction, which may be useful when holding a mobile device or manipulating its touch screen is impractical or even dangerous. Such a scenario may arise when the mobile device is used as a navigation device while driving, for example.

4.4 PalmSpace: Continuous Around-Device Gestures for 3D Object Rotation

making use of the higher detail of depth images, PalmSpace builds upon the ideas developed in HoverFlow

PalmSpace is a further project exploring Around-Device Interaction. The sensor used here is a time-of-flight depth camera. The advantage over the sensor configuration used in HoverFlow (Section 4.3) is that the depth image provided by the depth camera provides a more fine-grained representation of the user's hand. This can be used to recognize more complex gestures than in HoverFlow and also allows us to extract more complex geometrical properties about the hand, such as the angles of wrist flexion, pronation and supination (Grandjean, 1989; NASA, 1995).

We[3] implemented PalmSpace to study ADI for 3D rotation tasks. Rotation was determined by tracking the position of the user's palm using a

[3]Collaborators were: Dennis Guse, Michael Rohs, Jörg Müller, Gilles Bailly and Michael Nischt (Kratz et al., 2012a).

depth camera mounted on a mobile device. Although the depth camera that we used is very bulky and was operated at the limits of its near-field range, we obtained some very promising results. Task completion times were significantly lower than for a traditional touch-screen-based approach. Furthermore, ratings of PalmSpace were comparable to the touch-based interface our test subjects were familiar with.

In addition, we gained some insight into the preferred hand pose for ADI interfaces for direct 3D virtual object manipulation, which will be useful for further work. In the following we describe the PalmSpace project and present the results we obtained.

4.4.1 Motivation for PalmSpace

Current graphics hardware for mobile devices now allows for rendering sophisticated 3D scenes on mobile devices. This capability is useful for gaming, but also allows implementing CAD tools and other scenarios, such placement of virtual furniture in a mobile AR environment or browsing an online shopping catalogue in 3D. While it is now possible to deliver useful 3D content on mobile devices, interacting with this type of content is still challenging. Users not only face the limitations of direct finger interaction on touch screens caused by the fat finger problem, i.e. occlusion and accuracy (Siek et al., 2005), but 3D content itself requires more than the two degrees of freedom provided by touch screens.

Rotation and translation are two fundamental 3D operations. For PalmSpace, we focus on rotation within 3D scenes or the rotation of 3D objects. It is difficult to implement usable interfaces for 3D rotation and translation tasks on current mobile devices equipped with touch-screens. Current solutions for use on (multi-)touch screens, such as the virtual trackball metaphor (Henriksen et al., 2004), only allow for an indirect mapping of 2D input gestures to 3D object rotation. With the virtual trackball metaphor, an invisible sphere is overlaid on top of the object. Using this approach the 2D movement resulting from dragging the finger on the touchscreen is mapped to rotation of the 3D object. However, this technique has the drawback that the movement is mapped indirectly to rotation.

around-device interaction is a way of overcoming the limitations of touch screens

To overcome the limitations of touch screens, we propose to interact around the device for manipulating 3D content. Around-device interaction (Section 4.1) allows separating the input and the output of the mobile device by performing gestures in *proximity* of the device. We extend this concept by enabling the space around the device for gestural interaction: The gesture space is now delineated by the reach of

the user's arm. We denote this reachable space as *PalmSpace*. This is well suited for 3D interaction, because (1) it increases the degrees of freedom for input, (2) it provides a finer control–display gain ratio as a larger interaction volume with longer distances compared to a touch screen is available and (3) it allows more natural interaction as 3D operations can be directly mapped to 3D gestures.

Using the space behind and beside the device for gestural interaction has the advantage of being occlusion-free, as the hand does not cover the device's display, and provide a large input space. We refer to the spaces behind and next to the device as *BackSpace* and *SideSpace*, respectively.

We also propose a novel gesture for performing 3D rotations called *Palm Gesture*. A user holds the device in one hand and uses the non-holding hand to perform the gesture in the PalmSpace. The user orients the palm of the hand, which defines a plane, to manipulate the orientation of the 3D object/scene on the screen. This has the advantage to introduce a direct mapping between the gesture and the 3D operations, which is easy to understand and easy to learn for novice users and efficient for expert users.

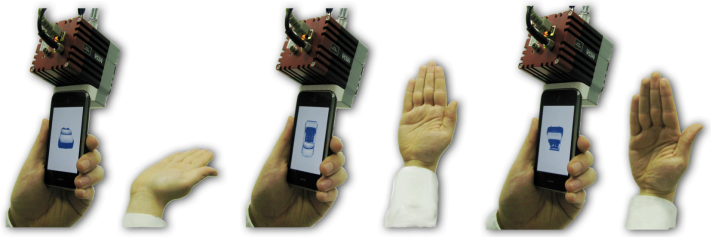


Figure 4.8: Using the pose of the flat hand behind the device to freely rotate a 3D object. A depth camera is used to determine the hand posture. (Kratz et al., 2012a).

We propose that such interfaces can be facilitated with depth cameras, which provide depth information for each pixel. We present a proof-of-concept based on a depth camera attached to an iPhone to capture the palm posture (Figure 4.8). We argue that it is reasonable to assume that manufacturers will be able to equip mobile devices with depth-sensing cameras in the future. In fact, some manufacturers already equip mobile phone with stereo RGB cameras[4].

We conducted a user study to determine the preferred hand poses for around-device gestural interaction. Based on these results, we report the findings of a user study comparing 3D rotation using SideSpace and BackSpace with the virtual trackball (Henriksen et al., 2004) as

[4] e.g. the LG Optimus 3D (LG Corp., 2012)[W].

a baseline. The results show that both SideSpace and BackSpace are faster and obtained ISO9241-9 ratings that are similar to the virtual trackball.

4.4.2 Related Work

PalmSpace draws from a relatively large body of related work. Previous work in the domain of Around-Device interaction is summarized in Section 2.2. In the following, we will elaborate on two specific domains of related work that are relevant to PalmSpace: mobile interaction techniques for interacting with 3D content and also existing work on interaction with depth sensors.

Interaction with 3D Content Mobile interaction with 3D content is often implemented using the device's accelerometer and magnetometer (compass). These sensors provide a reference orientation for the 3D objects, i.e. for augmented reality browser applications such as (Layar, 2012)[W]. The advantage of using this sensor combination is that the 3D scene can be viewed and controlled by holding the device with a single hand. Nevertheless, a problem when interacting using this technique is that the orientation of the display may not always be optimal for viewing, due to the tilt required to control the application. In Section 4.6, we demonstrate that using a virtual trackball on a touchscreen for 3D rotation tasks on a mobile device outperforms tilt-based control approaches. Hachet et al. (Hachet et al., 2005a) use a regular camera to detect color codes on a piece of cardboard for controlling a 3D object. This approach requires markers in the camera view and does not allow for rapid change between different gestures.

Interaction with depth sensors PalmSpace is also related to interaction with depth cameras and hand posture recognition. For instance, (Kollorz et al., 2008) proposed an algorithm for static hand posture recognition using a depth camera. These works mostly focus on *symbolic* recognition of certain hand postures or gestures, whereas this work emphasizes the use of the hand as a natural and *continuous* control mechanism for 3D content in mobile applications.

PalmSpace interaction is based on a continuous control paradigm

Expanding upon the concepts just described, PalmSpace uses the 3D space around the device for hand and finger gestures without the need for additional tokens. To understand the ergonomic limitations of gestural interaction in this space, we implemented a 3D viewer application. It is based on a "palm plane" metaphor to control 3D object rotation (Figure 4.8).

4.4.3 PalmSpace Interaction

In the following, we detail specific issues that we noticed when implementing PalmSpace interaction, that need to be taken into consideration when implementing PalmSpace-type user interfaces. First, we discuss the interaction volume and what limitations can arise from its properties. Second, we discuss the types of gestures that can be recognized with our current and also future systems, as well as the implementation limitations inherent in our approach. Last, we discuss the advantages and disadvantages of several strategies for mapping parameters obtained from gesture input to virtual object control.

4.4.3.1 Interaction Volume

We refer to the 3D space around the device that allows manipulating 3D virtual objects via hand gestures as the *PalmSpace*. One of the advantages of this approach is that the 3D space behind and to the side of the device avoids display occlusion and also achieves a close spatial correspondence between virtual 3D objects and movements in the physical space around the screen. However, we also found a number of critical aspects when designing gestures for this space by analyzing the ergonomic properties of the interaction volume, the types of gestures that can be detected robustly using a depth camera, as well as possible ways of mapping the data obtained from the depth camera to the virtual object control.

The interaction volume is bound by the pyramid-shaped camera's viewing frustum, i.e. its angle of view and depth range. The corresponding interaction volume is further restricted by the arm's reach of the user. Furthermore, entering and exiting the volume can be used as an implicit clutching or "action selection" mechanism[5].

4.4.3.2 Gesture Types

PalmSpace allows for the recognition of a wide variety of gesture types. For the work presented in this section, we chose to implement rotation gestures as a proof of concept that demonstrates PalmSpace interaction. More complex gestures and input techniques, such as pointing or palm-bending can, for instance, be implemented using machine learning techniques such as skeletal matching (Pavlovic et al., 1997).

[5]This has already been implemented for RGB cameras and is available as a commercial library (eyeSight Tech, 2012)[W].

Nevertheless, there are two types of limitations which need to be taken into account for PalmSpace interaction. The first is that the depth-map image provided by the camera scans the 3D volume from one perspective only and hence occlusions can occur. This becomes an issue if multiple fingers have to be tracked and one finger is hiding another while the hand is rotating. Therefore, we focus on palm gestures in which this kind of occlusion cannot occur. Secondly, there are ergonomic constraints. In particular, rotations around the wrist are only possible within certain limits and may be uncomfortable and precision becomes an issue at the limits of the wrist's rotation range (Rahman et al., 2009).

With the above in mind, we propose the *flat hand* as a useful continuous gesture, which we implemented for our prototype and evaluated in the user study. More precisely, the palm approximates a plane, which yields two parameters: *direction* orthogonal to the plane (surface normal) and *distance* to the camera center (origin of the coordinate system).

high-dexterity gestures can be explored in the future with improved hardware

Other types of hand gestures are to be explored in the future – potentially with improved hardware – include finger position and direction, palm bending, or general finger gestures. This opens a large space of potential high-dexterity gestures, however limitations of the depth camera have to be considered in the design of suitable gesture sets.

4.4.3.3 Gesture Parameter Mapping

The performed gesture needs to be mapped to the orientation (i.e., Euler Angles or a rotation matrix) of the manipulated virtual object. There are several options to accomplish this. The first option is to link the hand orientation directly to the virtual object, so users can change the orientation of the object by rotating their hand. This corresponds to *absolute control*, in which the hand orientation defines the orientation of the controlled object. This mapping is very intuitive but limited, due to the rotational ergonomic constraints of the hand and thus does not allow complete, 360°, rotations.

A variant of this mapping is *scaled absolute control*, which extends absolute control by a rotational scaling factor. This enables full rotation of the object with a smaller rotation of the hand, but it is likely that scaling will degrade precision. It is also unclear which scaling function provides the optimal usability in this case.

A further alternative is *rate control*, which maps the angle between the initial and the current hand pose to the rotational speed of the virtual object. This allows to rotate the object completely, but it needs to be done over time rather than defining the orientation directly and is prone

to overshoot. In fact, (Oakley and O'Modhrain, 2005) found that rate-control is significantly slower than absolute control.

extending absolute control using clutching appears to be the best control method

Alternatively , the concept of absolute control can be extended using clutching. Here, the hand orientation is not used to define the orientation of the virtual object absolutely, but instead relative to the orientation of the object and the hand at the time the clutching was engaged. This is called *relative control*. This allows performing complete rotation with the trade-off that using multiple gestures degrades performance. Clutching, however, preserves the intuitive mapping of the virtual object following the rotation of the hand once the clutch has been engaged. Engaging and disengaging the clutch can be accomplished by moving the hand in and out of the depth camera's viewing frustrum.

Figure 4.9: Hardware setup: The depth camera is mounted on top of a mobile phone in a slightly downward angle. The prototype is attached to a lanyard, such that it can be handled comfortably. (Kratz et al., 2012a).

We built a prototype system to evaluate PalmSpace interaction. The prototype consists of a depth camera mounted on a iPhone (Figure 4.9), a mobile application and a PC that performs computer vision to extract control inputs from the depth images for use by the mobile application.

4.4.4 Hardware Prototype for *BackSpace* and *SideSpace*

the hardware setup attempts to simulate the properties of a depth camera built into a mobile device

For depth imaging, we use a Mesa Imaging Swiss Ranger 4000 (Mesa Imaging AG, 2012b)[W], which uses modulated IR light to determine the pixel depth values. Under optimal conditions the camera achieves an absolute accuracy (deviation of the mean measured distance from actual distance) less than ± 10 mm and a repeatability (standard deviation of measured distances) of 4-7 mm. The camera has a resolution of 176×144 pixels and a horizontal and vertical field of view (FOV) of 44° and 35°, respectively. For close and/or highly reflective objects, the image can become partially saturated and provide no reliable depth information. The camera is optimized for distances from 60 cm to 4.40 m, which is too far for the intended PalmSpace setup. Using a low integration time of 1.8 ms and disabling some IR emitters using adhesive tape, we decreased the minimal distance from hand to camera down to 15 cm at the cost of accuracy. At the minimum distance the interactive area has a width of 12 cm and a height of 9.6 cm.

The complete setup aims at mediating the currently bulky hardware and simulating a system that is completely integrated into the mobile phone. For the BackSpace prototype the camera is attached above an iPhone 4 (upside down, portrait orientation) via an iPhone dock in a 38° downward looking angle (Figure 4.9). The SideSpace prototype consists of the iPhone flexibly suspended from the top in landscape orientation, for easier handling, and the depth camera attached to the backside using adhesive tape. We had to use different device orientations for BackSpace and SideSpace due to mechanical reasons. For SideSpace we adjusted the order of the rotation axes, so that the mapping from hand pose to object rotation remained identical to BackSpace. In order to relieve the user from the need to carry the additional weight of the camera, 510 g, the prototype is attached using a lanyard to a Manfrotto 420b lighting tripod. In this way, the prototype hangs freely in front of the users and they are not required to hold the relatively heavy setup.

The described setup allows the iPhone to be viewed and handled comfortably, while the non-holding hand is free for interaction behind or beside the prototype, respectively, in the FOV of the camera.

4.4.4.1 Gesture Recognition Application

For the vision processing, we use a PC with a Dual-Xeon quad core 2.66 GHz CPU running Linux. The prototypical gesture recognition application is written in C++ using the SwissRanger Linux Driver (Mesa Imaging AG, 2012a)[W] and the Point Cloud Library (PCL) (Rusu and

Cousins, 2011)[W]. The RANSAC (Fischler and Bolles, 1981) algorithm is used to estimate the rotational parameters of the plane described by the user's flat hand on a down sampled image, which is segmented using a depth threshold. The application estimates only the rotation around the x- and y-axes as the z-axis could not be reliably estimated using due to the limited field of vision of the camera setup. The estimated Euler angles are transferred via UDP multicast to the iPhone using a wireless router. On the previously described hardware, the application runs at 15 Hz.

4.4.4.2 Mobile Application

We developed an iPhone 4 application as a demonstrator for the interaction concept and as a test application for the user study. The application allows rotating the Stanford Bunny (Turk and Levoy, 1994) in 3D using either a conventional touchscreen approach or PalmSpace gestures. Rotation via the touchscreen is realized by a virtual trackball (Henriksen et al., 2004); please also refer to Section 4.6 for a description of the particular virtual trackball implementation we used.

The trackball allows rotation on all three axes by mapping the rotation input to a 3D Gaussian bell curve and converting the 3D translation on the curve to a rotation relative to the Gaussian's center as an angle/axis pair (Kratz and Rohs, 2010a). For PalmSpace the mobile application uses the Euler angles produced by the gesture recognition application (see previous paragraph) to calculate a model view rotation matrix for the Stanford Bunny.

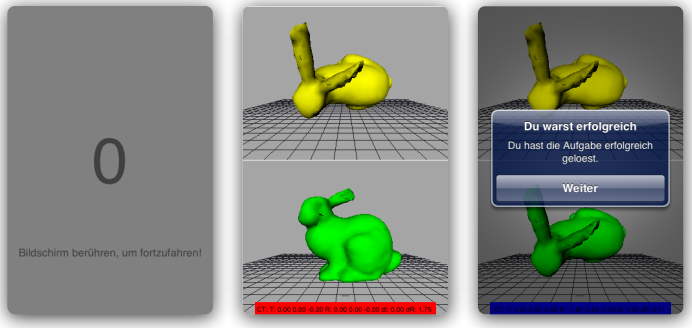


Figure 4.10: The sequence of screens shown for each trial in the user study. *(a)* Countdown screen that can be dismissed with a tap. *(b)* Trial screen that shows the target object on top and the controllable object at the bottom. *(c)* Feedback screen that indicates success or failure of a trial. (Kratz et al., 2012a).

(a) (b) (c)

Figure 4.11: The hand poses we considered in the pre-study: (a) in front of the device (P_{front}), (b) beside the device (*SideSpace*, P_{side}) and (c) behind the device (*BackSpace*, P_{back}). (Kratz et al., 2012a).

Figure 4.10 shows the screens of the application: countdown (a), trial (b) and feedback (c). The countdown screen can be dismissed by tapping the screen. The top/right half of the trial screen shows the target orientation of the object. The bottom/left half of the screen shows the user-controllable object.

During the study trials are served to the mobile device via HTTP. Each trial is separated by a 5 second countdown screen (Figure 4.10 (a)). When the countdown has reached zero, the countdown screen can be dismissed by tapping on the device's screen, and the new trial begins. Once a trial has completed, the device reports either success or failure of the task (Figure 4.10 (c)) and the task duration to the server.

4.4.5 User Study

In our user study, we compared BackSpace, SideSpace and a touch-based virtual trackball for 3D rotation tasks.The goal of the study was to show that BackSpace and SideSpace outperform the virtual trackball and are also rated higher, mainly because of the direct correspondence between palm rotation in 3D space on virtual object rotation.

4.4.5.1 Pilot Study

the optimal hand pose was unclear initially

Initially, the optimal placement of the hand relative to the device was unclear to us. Within the design space we are using, we identified three useful hand poses: hand in front of the device (P_{front}), hand to the side of the device (P_{side}) and the hand behind the device (P_{back}). Figure 4.11 illustrates the hand poses we propose.

the pilot study results indicate that P_{side} is the preferred pose

To determine the preferred and most comfortable pose, we conducted a pilot study with five participants. We asked the participants to simulate interaction in positions P_{front}, P_{side}, P_{back} as well as with the virtual trackball, $P_{trackball}$, as a baseline technique. After the participants made clear that they understood the interaction metaphor and the task to be accomplished, we asked them to rate their comfort levels following ISO 9241-9, and to provide a ranking of the techniques P_{front}, P_{side}, P_{back}, and $P_{trackball}$. The results of the pilot study indicate that P_{side} is the preferred gesture-based technique. It was rated better than P_{front} and P_{back} for comfort and was unanimously ranked higher than the competing gesture-based techniques.

P_{front} was dropped from the main study due to bad ratings

Probably owing to familiarity of touchscreen-based interaction, $P_{trackball}$ was rated best for comfort and was placed first in the ranking. The results of the main study, however, demonstrate the advantages of the gesture-based technique. Because hand pose P_{front} was rated lowest amongst the gesture-based techniques and because it leads to occlusion of the screen contents, we decided to drop this technique in the main experiment.

4.4.5.2 Hypotheses

Having obtained insights into the preferred pose of the hand relative to the device, we formulated the following hypotheses for our experiment:

- **H1** *BackSpace (P_{back}) and SideSpace (P_{side}) have lower task completion times than the virtual trackball (P_{front}).*
  We presume that BackSpace and SideSpace achieve a lower overall task completion time as those techniques provide a direct mapping.

- **H2** *SideSpace (P_{side}) has a lower overall task completion time than BackSpace.*
  We presume that SideSpace achieves a lower overall task completion time than BackSpace as the results of the pilot study revealed the hand pose of SideSpace is perceived as more satisfying.

- **H3** *BackSpace (P_{back}) and SideSpace (P_{side}) are rated worse with regard to required force than the virtual trackball ($P_{trackball}$).*
  We presume that the PalmSpace techniques are rated worse with regard to the required force as the virtual trackball only requires small finger movements whereas the PalmSpace techniques require to hold and move the hand.

4.4.5.3 Experimental Design

In our main experiment, we compared touch ($P_{\text{trackball}}$) with our gestural rotation techniques using poses P_{side} and P_{back}. The experiment used a counterbalanced within-participants design with interaction technique as the single factor. The levels of this factor are SideSpace (P_{side}), BackSpace (P_{back}), and virtual trackball ($P_{\text{trackball}}$).

Participants and Apparatus We invited 5 male and 5 female participants (mean age 27, age range 22-47), all of them were right-handed, and were compensated with a voucher for their participation. None of them participated in the pilot study. All participants were experienced in touchscreen-based interaction with mobile devices. For the BackSpace and SideSpace gestures the participants used the mobile application and setups described in Section 4.4.4. For the virtual trackball interaction they used the same application, but the unmodified iPhone 4 only (not suspended to the ceiling).

Procedure and Tasks The user's task was to rotationally align a 3D-object with a presented target as fast as possible. The control object and the target object were both Stanford Bunnies. The control and the target object differed in color and screen location, with the target object being located at the top of the mobile device's screen and the control object at the bottom. The user interface for the rotation task is shown in Figure 4.10 (b). For each technique the user had to perform 24 trials. The trials consisted of single and combined rotations around the x-axis and y-axis with the Euler angles α and β. Before conducting the trials with each interaction technique the participants had approximately 5 minutes time to try out and explore the technique.

the experiment task was to rotationally align a virtual object with a target object along two rotation axes

In terms of human physiology (Grandjean, 1989; NASA, 1995), the Euler angle α corresponds to the flexion rotation of the human wrist, and β corresponds to both pronation and supination. The hand movements required to rotate around α and β are further depicted in Figure 4.12. Keeping within the detection capabilities of our gesture-based system and the ergonomic constraints of wrist rotation, we selected the following values in degrees for α and β for use in our trials:

$$\alpha \in \{-50, -40, -35, -30, -25, -15, 0, 15, 20, 30, 50\}$$
$$\beta \in \{-70, -50, -40, -25, -20, -15, 0, 10\}$$

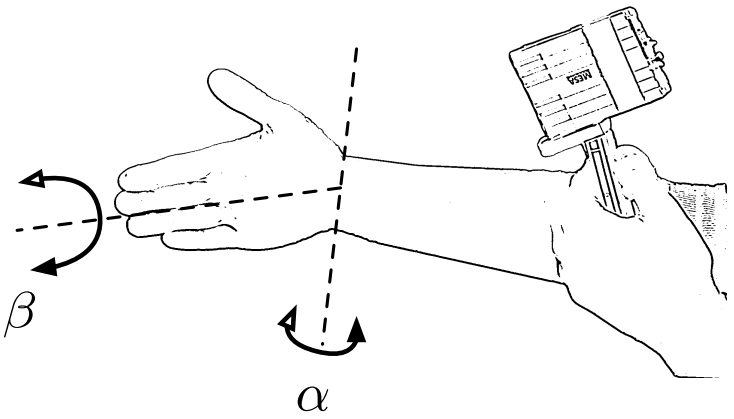


Figure 4.12: The Euler angle α is controlled by wrist flexion, whereas β is controlled using pronation and supination. (Kratz et al., 2012a).

successful trial completion depended on dwelling on the target at a maximum angular distance $\Phi_\epsilon = 9.9°$ for 300ms

For a trial to be completed successfully, the user had to dwell for 300ms within a rotation distance (Huynh, 2009) of $\Phi_\epsilon = 9.9°$, which corresponds to a maximal offset of ± 7 degrees for α and β, respectively. The dwell time is included in the completion times reported in the results section. If, after 30s the user had failed to achieve a rotation satisfying these criteria, the trial was aborted and marked as unsuccessful.

4.4.5.4 Measured Variables

the main measurements were the task completion time and an ISO9241-9 questionnaire

To evaluate the performance of BackSpace, SideSpace, and virtual trackball for the rotational task, we measured the time to completion for each trial and the number of trials that could not be completed. In addition, the users had to fill out a questionnaire after completing all trials for a given technique. The questionnaire is based on ISO9241-9 (ISO/IEC, 2000a). At the end of the experiment the user was asked to fill out an additional questionnaire, which asked to order the interaction techniques according to their personal preference, to rate the intuitiveness and to denote the most comfortable posture. Finally, the users had to indicate which input technique they would be comfortable with using in public places.

4.4.6 Study Results

In the following we present our observation on the interaction postures used by the test subjects during the experiment, as well as the results for task completion times and the user questionnaires.

BackSpace (P_{back}) and SideSpace (P_{side}) require two handed interaction as one hand needs to hold the mobile device and the non-holding hand is used for interaction. This posture was explained and shown to the participants.

Interaction with the virtual trackball ($P_{\text{trackball}}$) is possible one-handedly using the thumb on the touchscreen as well as using two hands, i.e. using one hand to hold the device and one finger of the non-holding hand for interaction. For this interaction technique no specific instructions where given whether the technique should be used in a one- or two-handed manner.

participants eventually all transitioned to two-handed interaction for the trackball technique

In the exploration phase before the test trials, two participants first explored one-handed interaction, but switched to two handed interaction as they found the thumb to be too imprecise as well as too big with regard to occlusion. All other participants directly used two-handed interaction. All participants solved the virtual trackball trials using two hands.

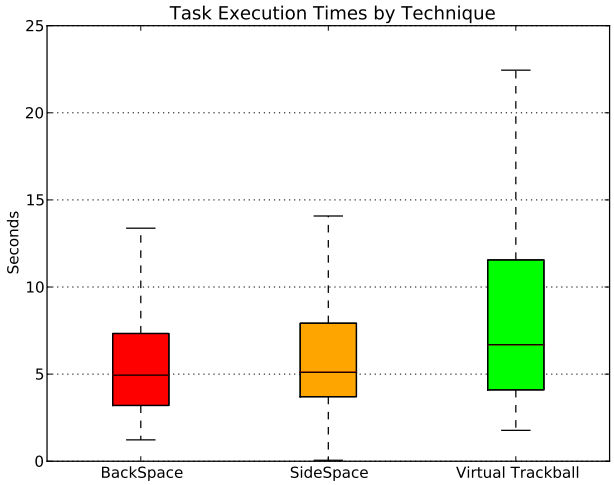


Figure 4.13: Box plots of the task completion times by input technique. (Kratz et al., 2012a).

4.4.6.1 Task Completion Times

We recorded a total of 240 trials for each technique. Not all trials were completed successfully. For BackSpace (P_{back}) 20 trials (8.3%), for SideSpace (P_{side}) 21 (8.8%) and for the virtual trackball ($P_{\text{trackball}}$) 22 (9.2%) trials were marked as being unsuccessful.

Of the successful trials, we obtained the following average task completion times for BackSpace (P_{back}) 6.09s, $\sigma = 4.36$, for SideSpace (P_{side})

7.19s, $\sigma = 5.6$ and for the virtual trackball ($P_{\text{trackball}}$) 8.45s, $\sigma = 5.82$. A box plot of the results for task completion time is shown in Figure 4.13.

A histogram analysis of the task completion data showed a strong left skew of the task completion times. We log-transformed the data to obtain an unskewed Gaussian distribution of the data.

we observed a significant effect for input technique on the task completion time

We conducted an ANOVA on the log-transformed task completion data and found a significant effect for input technique: F(2, 16) = 10.42; $p < 0.001$. A Sidak post-hoc analysis indicates that the differences between each of the techniques are significant:

- BackSpace (P_{back}) vs. SideSpace (P_{side}): $p = 0.04$
- BackSpace (P_{back}) vs. virtual trackball ($P_{\text{trackball}}$): $p < 0.001$
- SideSpace (P_{side}) vs. virtual trackball ($P_{\text{trackball}}$): $p = 0.037$

BackSpace and SideSpace have significantly lower task completion times than the virtual trackball

Thus, **H1** is validated as BackSpace and SideSpace show lower average task completion times than the virtual trackball. However **H2** could not be confirmed as BackSpace shows significantly lower average task completion times than SideSpace, even though SideSpace is the preferred input technique of the pilot study.

4.4.6.2 User Evaluation

we obtained no statistically significant results for the rankings

In the final questionnaire the participants were asked to order the interaction techniques according to their personal preference and their perceived intuitiveness. Figure 4.14 shows the results. Employing the Kruskal-Wallis h-test, we could neither find a statistically significant mean difference for personal preference ($\chi^2 = .817; p = 0.665$), nor for intuitiveness ($\chi^2 = 2,030; p = 0.362$).

we obtained significant results for overall effort and required force

After completing all trials for each interaction technique the participants were asked to complete the ISO9241-9 questionnaire. Figure 4.15 shows the results. A Kruskal-Wallis h-test shows a significant result for the overall effort ($\chi^2 = 10.225$; $p = 0.006$) as well as the required force ($\chi^2 = 10.205$; $p = 0.006$). Thus, **H3** is validated as the user perception of required force and overall effort for SideSpace and BackSpace is significantly higher than for the Virtual Trackball.

virtual trackball is preferred technique for public use

In public situations, 6 of the 10 participants would use BackSpace and their remainder would use SideSpace. The virtual trackball outperforms both PalmSpace variants as all participants would use the virtual trackball in public places. However, it must to be kept in mind that the participants encountered the PalmSpace interface the first time whereas touchscreen-based interaction is widely known and adopted.

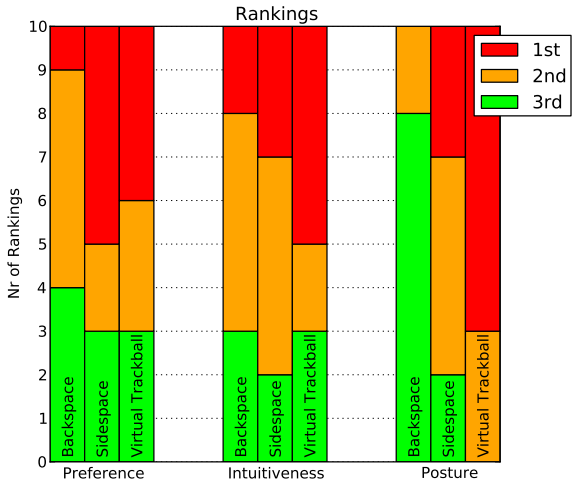


Figure 4.14: This Figure shows the distribution of rankings given by input technique for *preference*, *intuitiveness* and *posture*. (Kratz et al., 2012a).

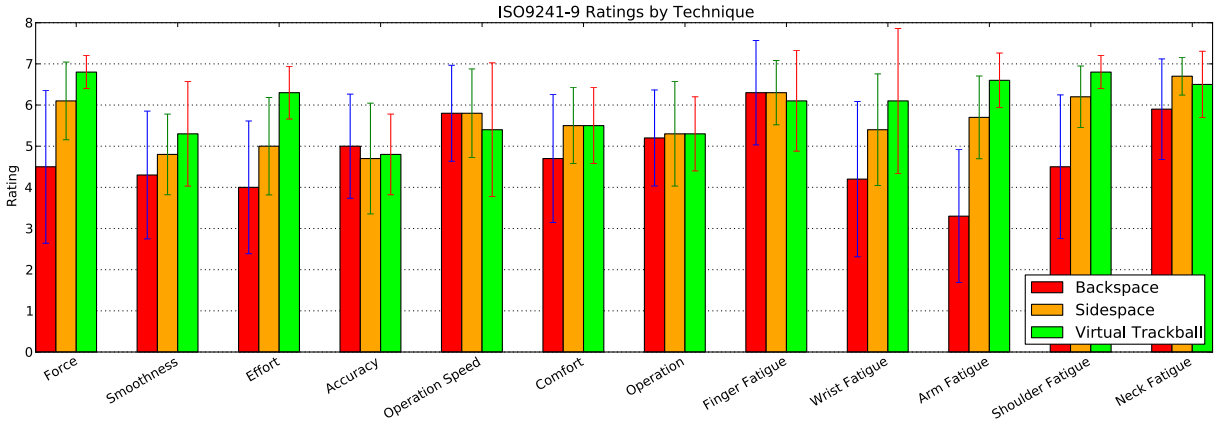


Figure 4.15: The average ISO90241-9 ratings given by technique on a seven point Likert scale. The error bars indicate the standard deviation. (Kratz et al., 2012a).

4.4.7 Conclusions and Future Work

PalmSpace presents two new interaction techniques for orientation manipulation of virtual objects for mobile devices. We introduced the "palm metaphor", which makes manipulation intuitive and fast as the manipulated object always adopts the same orientation as the palm. The results of the presented user study show that our prototypical implementation performs very well with regard to performance as well as user perception and intuitiveness.

"palm metaphor" with two new interaction techniques was introduced

Most notably, the two presented variants of PalmSpace perform better than the virtual trackball, which is the current state-of-the-art solution

for rotational tasks on mobile devices using a touchscreen. Moreover, the PalmSpace might be extendable to translation tasks as well. In this way it is reasonable to assume that 6DOF are achievable using the palm metaphor, which would make it fully suited for viewing and manipulation of 3D objects as well as navigation in 3D environments like in navigational tasks or gaming.

The results we have obtained in the PalmSpace study imply that around-device interfaces for mobile devices are likely to improve the usability for tasks requiring a high number of simultaneous degrees of freedom as well as high precision. We believe that the direct manipulation metaphor afforded by SideSpace allowed the users to rotate the 3D object more intuitively than the virtual trackball because the users can sense the current hand posture very well through proprioception. Thus, we argue that the control of user interface artifacts through direct hand gestures will play an important role in future mobile user interfaces.

future work will explore even finer gestures for around-device interaction

In the future we plan to explore even finer gestures, i.e. on an individual finger level, in around-device space. The will enable us to build even more expressive interfaces, which can be used for very precise manipulation or selection tasks in 3D and very fine-grained gestures. Such an interface has a vast potential to increase the expressiveness of input to mobile devices. To implement these proposed interfaces, we can no longer rely on a depth camera. Instead we plan to use a 3D tracking system, such as the Naturalpoint OptiTrack (NaturalPoint Inc., 2012)[W], which allows tracking of 6DOF bodies in 3D space with a sub-millimeter precision.

4.5 The iPhone Sandwich: Pressure-Based Dual-Sided Multi-Touch Interaction

To study a further aspect of around-device interaction, pressure-based dual-sided back-of-device interaction we[6] developed the iPhone Sandwich prototype (Essl et al., 2009).

4.5.1 Hardware Setup

force-sensing resistors were placed between two iPhones to allow pressure sensing and dual-sided multi-touch

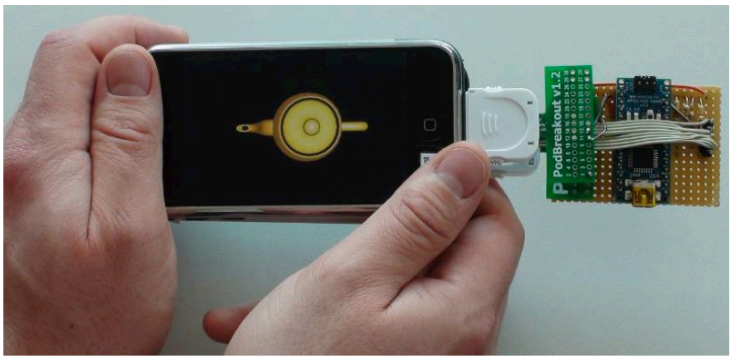
The iPhone Sandwich consists of two first-generation iPhones attached back-to-back via a 3 mm acrylic plate. We instrumented the acrylic plate with four force-sensing resistors (FSR), the placement of which

[6]The prototype was developed in collaboration with Michael Rohs and Georg Essl.

is depicted in Figure 4.16 (b). The 3 mm standoff provided by the plate allowed us to cut grooves into the plate to route the sensor cables.

We used an Arduino microcontroller to perform analog-to-digital conversion and transmit the pressure sensor values to the front iPhone via an serial connection. The touch events from the rear device were transmitted to the front iPhone via wireless LAN using UDP datagrams.

All applications for the iPhone Sandwich were run on the front device for the sake of simplicity. This setup is, however, by no means limited to single-sided display only. Two display sides could be used to extend an application's usable display size. The display currently being looked at by the user could by inferred by measuring the direction of gravity using one of the iPhones' acceleration sensors.



(a) iPhone Sandwich Prototype

(b) Placement of Pressure Sensors

Figure 4.16: *(a)* Shows the assembled iPhone Sandwich prototype with the four pressure sensors attached to the microcontroller board. (Stewart et al., 2010).
(b) shows the placement of of the pressure sensors in the space between the two devices that comprise the iPhone Sandwich.

4.5.2 Affordances for Interaction

Besides allowing local input with a high local expressivity, the iPhone Sandwich affords interactions that are impossible with traditional one-sided touch input.The degrees of freedom at the touch points of the

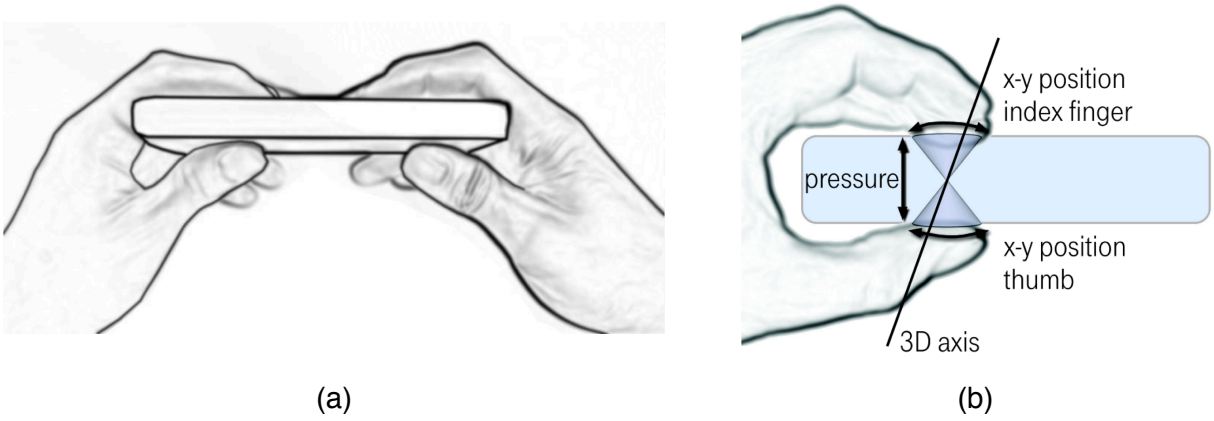


Figure 4.17: When used in its recommended posture *(a)*, the iPhone Sandwich affords a high number of local degrees of freedom *(b)*.

user (when grabbing the device using forefinger and thumb) are: x, y position for the index finger, x, y, position for the thumb and the current pressure value. These degrees of freedom can be used to manipulate a virtual object that is located conceptually in the area between the two screens (Figure 4.17).

Therefore, everyday actions such as grabbing, sliding, twisting and turning are readily supported by the iPhone Sandwich. For instance, it is possible to model grabbing a deformable object with one hand and allow the user to twist or bend it with the other hand.

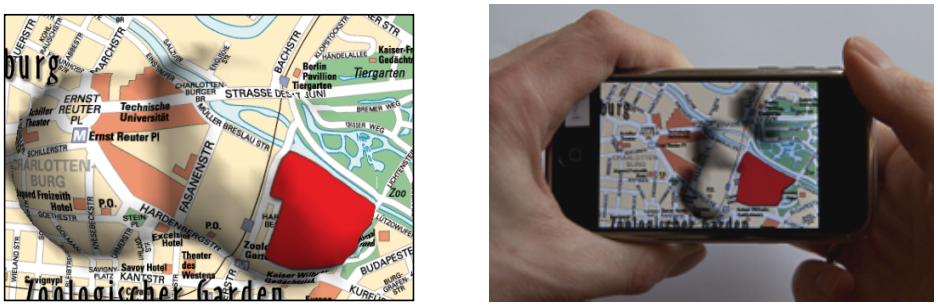
4.5.3 Application Scenarios

object deformation through pressure

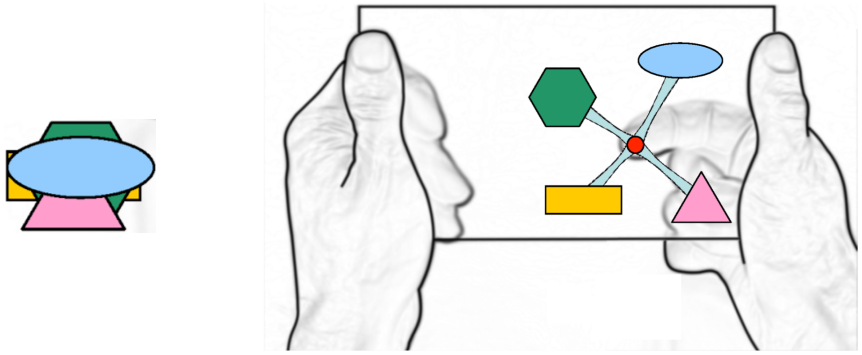
We developed several application scenarios, examples of which are shown in Figure 4.18. Pressure is a natural metaphor for the manipulation of deformable objects. Figure 4.18 (a) shows an example map browser. The map canvas is deformable by applying pressure from the rear. This allows the user to zoom in on regions of interest (ROI), by pushing in the map at a certain touch location, which leads to a fisheye lens effect around the selected ROI. Because the iPhone Sandwich is a multi-touch device, multiple fisheye lenses can be applied to the map canvas by the user.

uncovering of occluded objects

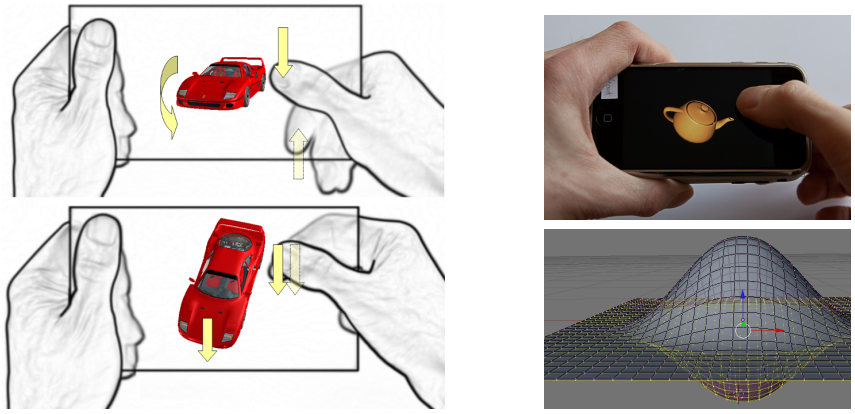
Another map-related technique is uncovering occluded objects on a map (Figure 4.18 (b), left). Rather than applying a fisheye lens, the objects can be moved apart when pressure is applied at the location of the object "stack" (Figure 4.18 (b), right). The user can directly control the divergence of the objects by varying the amount of pressure applied.



(a)



(b)



(c)

Figure 4.18: Sample scenarios for interaction using the iPhone Sandwich. *(a)* Pressure-based map navigation. *(b)* Revelealing occluded objects *(left)* by applying rear-side pressure *(right)*. *(c)* 3D object manipulation using physical analogies *(left)* and rotation using a virtual trackball *(right)*.

dual-sided manipulation of virtual objects

Lastly, dual-sided interaction with pressure input makes it easy to implement user interfaces that allow the manipulation of virtual objects. Figure 4.18 (c), left, shows a manipulation method that derives from the affordances depicted in Figure 4.17 (b). This interface is intended to model manipulating physical objects using the hand. Rotation occurs when the fingers of the right hand are moved in divergent paths, thus rotating the object, as would be the case when manipulating a physical object (Figure 4.18 (c), upper left). By contrast, moving the front in rear fingers along parallel paths results in a translation, as the simulated forces on the object in this case act in equal directions ((Figure 4.18 (c), lower left).

rear-of-device virtual trackball

A more traditional metaphor for object rotation is the virtual trackball, Figure 4.18 (c), right, which performs very well when it is implemented on the rear of the device. We performed a detailed study of rotation tasks comparing a front and a rear virtual trackball as well as a tilt-based technique, the results of which are discussed in detail in Section 4.6.

4.5.4 Pressure Sensor Characteristics and Mapping Functions

To explore the users' ability to input pressure using the iPhone Sandwich we[7] conducted a user study on pressure input controllability and different grip conditions and sensor mappings (Stewart et al., 2010).

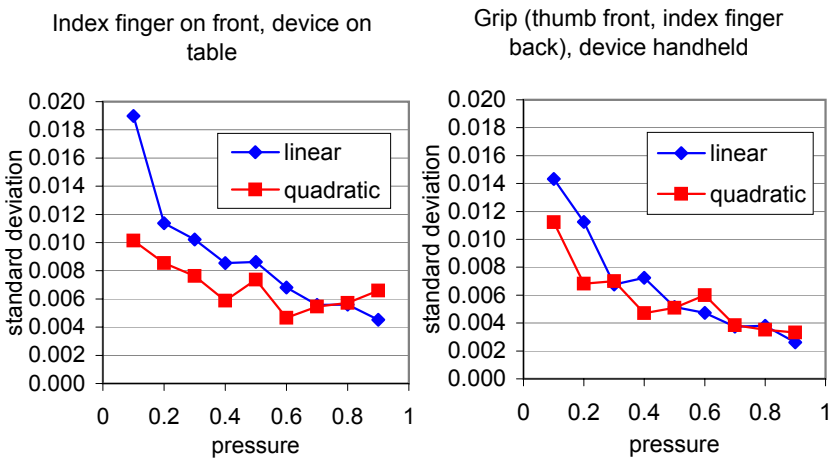


Figure 4.19: Standard deviation for different pressure levels during 3 s intervals. Variability is high for low pressure levels. (Stewart et al., 2010).

The task was structured such that the user had to keep pressure at a certain level for five seconds. Users had to move to the target pressure

[7]Work conducted in collaboration with Craig Stewart, Michael Rohs and Georg Essl.

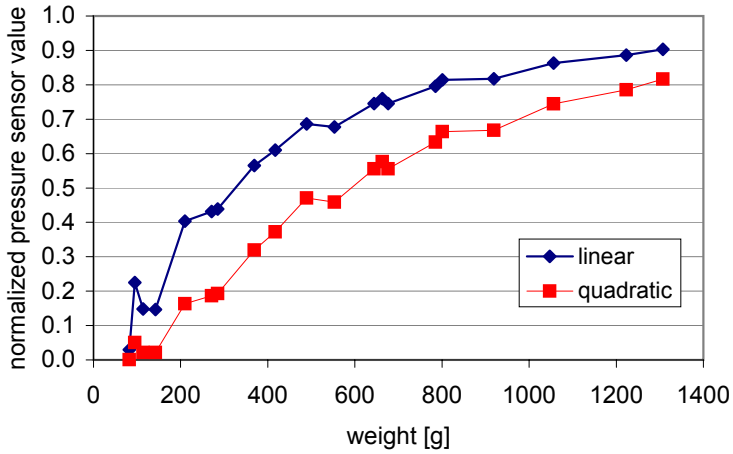


Figure 4.20: Combined characteristic of FSR response curve and voltage divider. (Stewart et al., 2010).

level and then had to keep pressure at that level as precisely as possible. The goal was to estimate the variability of pressure input for each of the levels.

Pressure input was measured for two poses, one with the device lying flat on a table with pressure applied on the front of the device (*front-on-table*) the second while the device was gripped in the user's hands (*grip*). Additional conditions were three different sensor output mappings: *linear*, *quadratic* and *logarithmic*. We measured the pressure stability by computing the standard deviation of the pressure values measured during the last three seconds of the five-second hold-down time at a given pressure level.

Figure 4.19 shows the results of the study for the front-on-table and grip techniques using the linear and quadratic mappings. The logarithmic mapping falls between linear and quadratic and is left out for clarity. The graphs show the median variability over the last three seconds of each five-second step. We computed this as the median of the standard deviations for each condition. Variability decreases noticeably with increasing pressure levels.

Reasoning that the decreased variability may be due to non-linear output introduced by our sensor setup, we conducted a weight-to-sensor value measurements. We put a range of weight on top of a single FSR and sampled the sensor output over two seconds. Figure 4.20 shows the resulting curves.

The blue curve shows the pressure range linearly rescaled from 0 to 1 (linear mapping in the experiment), the red curve shows the result of the quadratic mapping. It can clearly be seen that the blue curve is

not linear but steeper for lower pressure values, i.e., the sensor reading is not a linear function of the pressure value. The data has a good fit to a logarithmic function ($p = 0.3144\ln(x) - 1.3116, R^2 = 0.98$). The quadratic mapping shows a flatter slope for low pressure values. For the linear mapping this means that user input variability at a low pressure value will be translated into a larger variability in the sensor output than the same input variability at a higher pressure level. In order to compensate for this characteristic of the FSR and voltage-divider circuit one would have to use a mapping that is the inverse function of the resulting logarithmic characteristic, which would be an exponential function, in this case:

$$q = \exp((p + 1.3116)/0.3144) \tag{4.1}$$

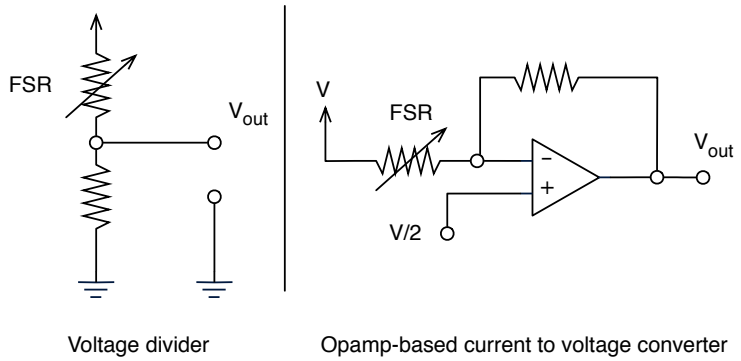


Figure 4.21: Voltage divider *(left)* and opamp-based circuit *(right)*. The latter provides linearized sensor input.

4.5.4.1 Linearizing the Sensor Output

As we wanted to use the resolution of the sensor to its full capacity we decided to build a new hardware setup in which the hardware already provides linear sensor input. We used an opamp-based current to voltage converter (Figure 4.21). The transfer function of the voltage divider is:

$$V_{out} = \frac{R}{R + R_{FSR}} \cdot V_{in} \tag{4.2}$$

Hence, a voltage divider does not simply create a linear relationship between the resistance of R_{FSR} and the output voltage (Horowitz and Hill, 1989; Putnam and Knapp, 1996)[W]. This relationship is arrived by simple use of Ohm's law. This is also noted in documentation of the force sensing resistor (Interlink Electronics), who propose a current-to-voltage circuit to achieve a linear relation itself.

The operational amplifier has two defining characteristics. One is that the impedance between the two inputs is very high and theoretically

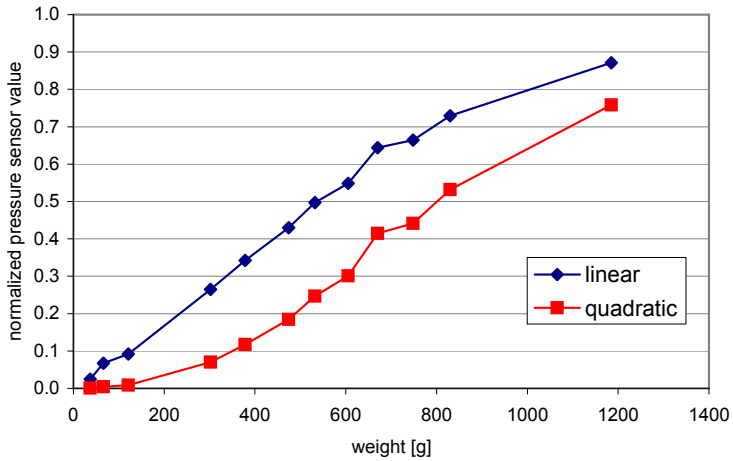


Figure 4.22: Combined characteristic of FSR response curve and opamp- based circuit. (Stewart et al., 2010).

often treated as infinite. This has the effect that there is minimal load on any circuit placed left of the opamp. The second characteristic is that the output impedance is very low which makes the circuit insensitive to load or energy demand at the output. The operational amplifier will amplify the output as needed to achieve these properties. This makes this element very versatile for building a range of analogue circuits (Horowitz and Hill, 1989). To understand the current-to-voltage converter of Figure 4 note that the input impedance is such that practically no current will flow into the inputs. Hence all current at the negative input will instead flow across the resistor connecting to the output. Hence the output voltage simply obeys Ohm's law taking the negative polarity of the input into account. Thus we arrive at a relationship of

$$V_{out} = \frac{V}{2}(1 - \frac{R}{R_{FSR}}) \tag{4.3}$$

the output will be linear with respect to R_{FSR} for a large range of output loads since the resistance of the FSR is inverse proportional to the applied pressure force (Interlink Electronics) and (Putnam and Knapp, 1996)[W]:

$$F_{FSR} \sim \frac{1}{R_{FSR}} \Rightarrow V_{out} \sim -F_{FSR} \tag{4.4}$$

The characteristic of the opamp circuit was measured in the same way as described above. Figure 4.22 shows the normalized linear mapping as well as the quadratic mapping. The linear mapping now has a good fit to a linear function ($p = 0.0008x + 0.0339, R^2 = 0.97$).

4.5.4.2 Pressure Controllability with Linearized Sensor Input

We repeated the study of Section 4.5.4, using only the grip pose and comparing both hardware setups, voltage divider and op-amp. For the voltage divider we used the exponential function from Equation 4.1 to linearize the input data. The experimental factors thus were hardware (voltage divider and opamp-based) and transfer function (linear and quadratic). Otherwise the experimental task was identical to the one previously described.

The results of the input variability are shown in Figure 4.23. For both the old hardware (voltage divider and exponential correction function) and the new hardware (opamp-based) the linear transfer function works better than the quadratic function. The reason is probably that the quadratic function over compensates the already linearized sensor input. Moreover, comparing the linear mappings for the old and new hardware, one can observe that there is an advantage of the new hardware. This is probably due to the better use of the dynamic range and resolution of the setup.

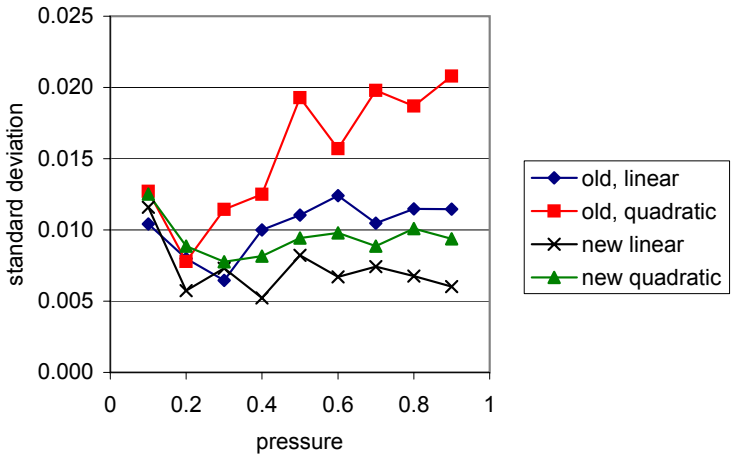


Figure 4.23: Standard deviation for different pressure levels during 3s intervals. Overall variability is better for the new hardware and the linear mapping. (Stewart et al., 2010).

a linear transfer function pressure input is superior to a quadratic mapping

Overall these results are in line with the findings of (Srinivasan and Chen, 1993) and show that human ability to control pressure input is not per se worse at lower pressure levels. A linear mapping works better than a quadratic mapping if the sensor data is a linear mapping of the input force. The results seem to suggest that controllability of force is uniform for a wide range of pressure levels.

4.5.5 Poses for Pressure Input

Pressure-based input usually assumes that the device to which pressure is applied is resting on a stable surface. Examples are pressure-sensitive pen input to tablet PCs or pen tablets. (Ramos et al., 2004) explored pressure-based input for stationary devices. Pressure application to mobile devices when handheld has not been extensively researched. In addition, many previous pressure-based interfaces assumed a pen or stylus to apply pressure (e.g. (Ramos et al., 2004; Mizobuchi et al., 2005)).

4.5.5.1 Investigating Pressure Input on Handheld Device Poses

We conducted a study to investigate direct finger-based pressure input for handheld devices. Our goal was to find out which device poses are most suitable for handheld pressure input. We compared a device resting on a table with a handheld device as a baseline. Our interest lay in how quickly and accurately users can control pressure they exert with one or more fingers on the device and what pressure range is useful for interaction. In particular, we investigated user performance of pressure-based input under the following poses (4.24):

the goal of our study was to determine the most suitable pose for handheld pressure input

- **Front-on-table:** index finger on front of device, device resting on table.
- **Front:** thumb on front of device, device handheld.
- **Back:** index finger on back of device, device handheld.
- **Grip:** thumb on front and index finger on back of device, device handheld.

4.5.5.2 Apparatus

The target pressure was represented as a value ranging from 0.1 to 0.9 where 0 represents zero pressure and 1 corresponds to the maximum pressure. We presented target widths on this scale of 0.02 and 0.04. The pressure input was linearized using the hardware described in Section 4.5.4.1.

We measured the pressure using two FSRs, one attached to the front and one to the back display of the iPhone Sandwich. The device was always held in landscape orientation. The FSRs were vertically centered and attached about 3 cm from the right edge of the device to be easily reachable with both the thumb and the index finger (4.24).

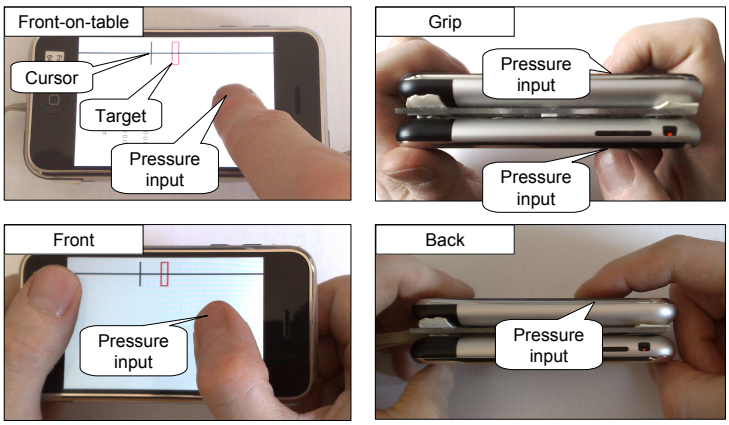


Figure 4.24: Device poses tested for handheld pressure input: index finger on front, device resting on table (baseline); grip with thumb and index finger; thumb on front; index finger on back. Changing pressure moves the cursor on the horizontal line. The red rectangle indicates the target pressure and target width. (Stewart et al., 2010).

4.5.5.3 Participants and Experimental Design

We recruited twelve participants (8 male, 4 female) ranging in age from 15 to 36 years ($\mu = 26.6$, $\sigma = 6.7$) for the experiment. All participants were right-handed.

The experiment used a $4 \times 9 \times 2$ within-subject factorial design. The factors and levels were *pose* (front-on-table, front, back, grip), *target pressure* (9 levels from 0.1 to 0.9), *target width* (narrow (0.02) and wide (0.04)).

To solve the task, the users had to sequentially select targets that appeared at random positions on a horizontal bar (compare Figure 4.24, top left). The left end of the bar corresponded to zero pressure, the right end to maximum pressure. The device's display showed continuous visual feedback. As the user increased pressure a vertical line cursor moved along the bar. The target was shown as a red rectangle on the bar. The target was selected by keeping the cursor within the target rectangle for the dwell time of 1 s. Selecting the target ended the trial and the target moved to the left end of the bar (zero pressure). The user had to release pressure and wait for one second after which the next trial would be started at the new target position.

The order of presentation of the four device poses was counterbalanced using a latin square design. The order of target widths was counterbalanced within the poses. The distances were presented in three blocks. Within each block the ten distances (0.1 to 0.9) presented in random order. This amounts to 4 poses $\times$ 2 widths $\times$ 3 distance blocks $\times$ 10 distances per block = 240 trials per user.

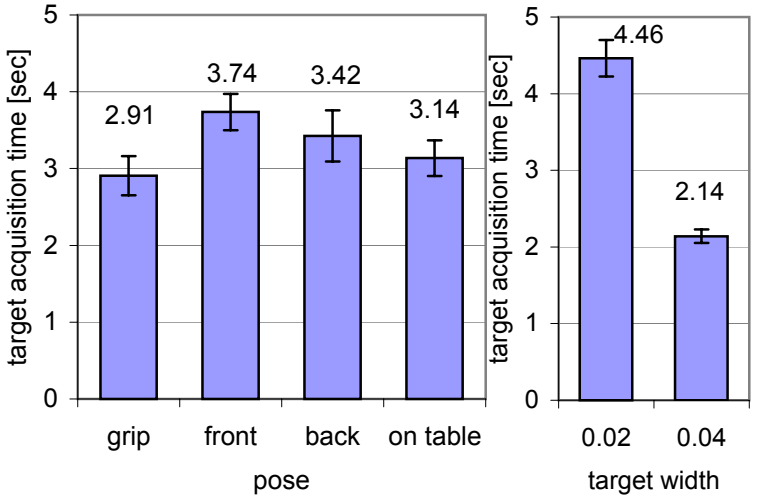


Figure 4.25: Target acquisition times for the four device poses. (Stewart et al., 2010).

4.5.5.4 Results

grip posture performs comparably to pressure application on a solid surface

We measured the time required for selecting a target and logged the pressure sensor values over time. The mean selection times (1s dwell time subtracted from these values) are: 3.14 s for front-on-table, 2.91 s for grip, 3.74 s for front, and 3.42 s for back (Figure 4.25). A repeated-measures ANOVA on the log-transformed data shows that these differences are statistically significant ($F_{3,33} = 9.11$, $p < 0.001$). Bonferroni corrected post-hoc comparisons show a difference between front and other poses, but not among the other poses. The mean values suggest that handheld pressure application, specifically for grip posture is not disadvantaged compared to pressure application to a solid surface.

target pressure significantly raises acquisition time

The average selection time was 4.46s for the narrow target and 2.14 s for the wide target ($F_{1,11} = 152.08$, $p < 0.001$). Wide target selection differentiates the results more strongly. Front hand-held shows up to a factor of 3 degradation in time-to-target compared to grip for low pressure values (Figure 4.26). The median selection time showed a linear relationship with pressure, ranging from 1.5 s at 0.1 to 3.4 s at 0.9 ($t = 2.277p + 1.261$, $R^2 = 0.95$). The results show a significant effect of target pressure on acquisition time ($F_{8,88} = 50.88$, $p < 0.001$). We also asked users which of the poses they preferred. Six of twelve users preferred grip, 3 index finger front-on-table, 2 index finger on back, and 1 user preferred thumb on front.

The results show that pressure-based selection is possible in reasonable selection times, even with 9 targets equally distributed on the pressure range and with fairly narrow target widths. All handheld poses, except

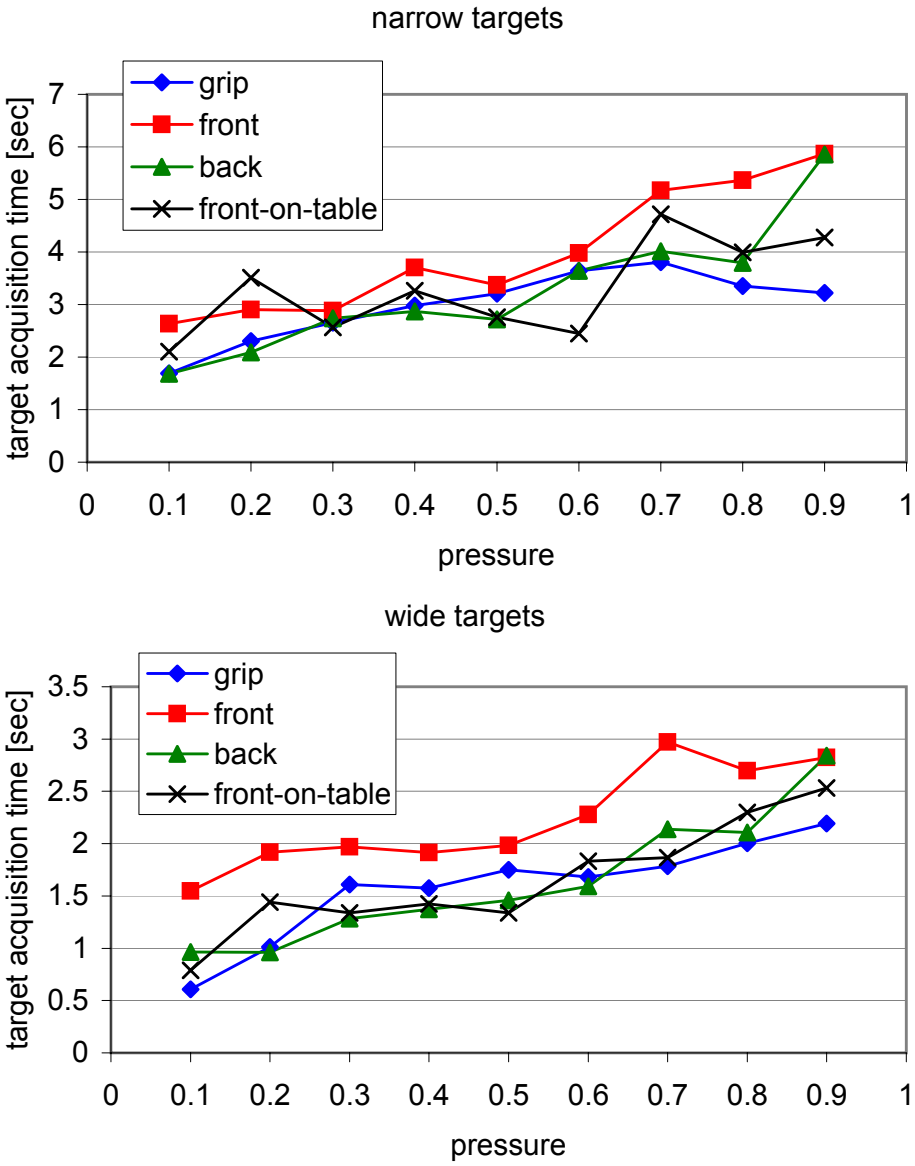


Figure 4.26: Target acquisition times by target pressure for the four device poses. The top graph shows the results for narrow the bottom one for wide targets. (Stewart et al., 2010).

front, are on par with the front-on-table pose, in which the device is supported by a table surface.

4.6 Rear-of-device interaction for Rotation Tasks

Having developed the iPhone Sandwich prototype, which is described in Section 4.5, we conducted a user study about using the iPhone Sandwich for 3D rotation tasks (Kratz and Rohs, 2010a), by implementing a

rear-of-device virtual trackball. Our motivation for placing the rotation control widget on the rear of device is that the occlusion of the display by the user's hands and fingers is no longer be an issue. Moreover, the front side of the display can be used for secondary tasks, such as object selection, while the user can continue with the primary task of 3D viewing using the device's rear.

4.6.1 Background

Interaction with 3D objects and scenes has many applications on mobile devices, including games and 3D model inspection. The hardware capabilities of mobile devices for rendering 3D content are increasing rapidly. Direct touch input on mobile devices potentially allows for more direct interaction with 3D objects than is possible on desktop PCs. However, there are unique challenges of devices with small displays, such as the relatively large amount of occlusion that occurs with direct finger-based touch interaction. This problem has been named the "fat finger problem" (Baudisch and Chu, 2009). Several solutions have been proposed for overcoming or alleviating this issue, including back-of-device interaction (Baudisch and Chu, 2009; Shen et al., 2009b; Sugimoto and Hiroki, 2006; Wigdor et al., 2007b), tilting-based interfaces (Rekimoto, 1996b), and temporal separation of input event and interface action (Decle and Hachet, 2009).

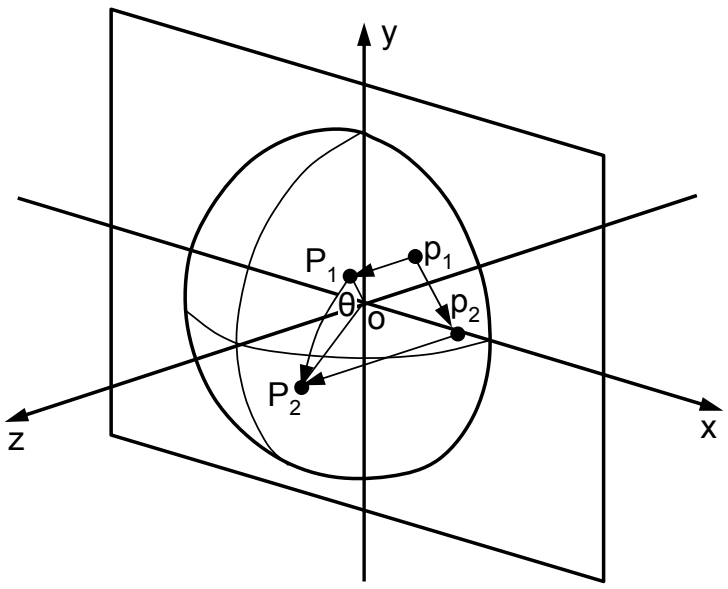


Figure 4.27: Touch points (p_1, p_2) inside virtual hemisphere: rotation defined by great circular arc through projected touch points (P_1, P_2). (Kratz and Rohs, 2010a).

Relatively little research has focused on mobile interaction with 3D content (an example is (Decle and Hachet, 2009)). The study presented

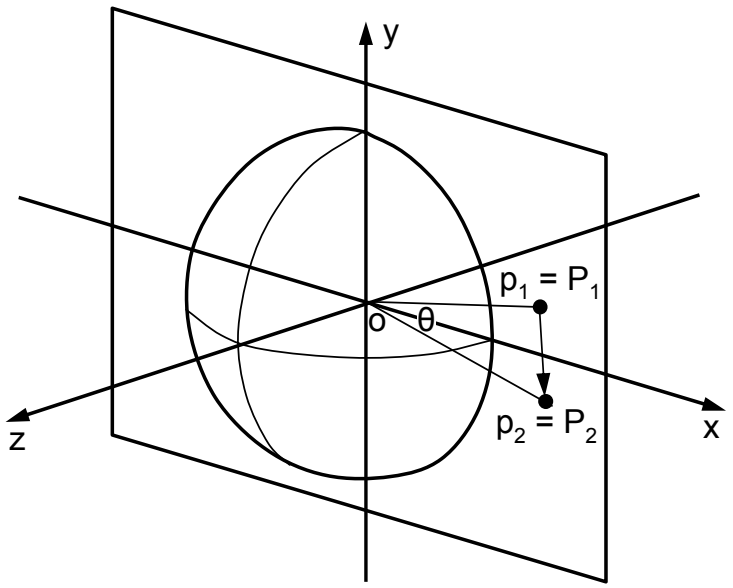


Figure 4.28: Touch points (p_1, p_2) outside virtual hemisphere: rotation around z-axis. (Kratz and Rohs, 2010a).

in this section focuses on 3D rotation, which is a fundamental task allowing users to inspect and understand 3D objects. A widely used interaction technique for rotating 3D objects around arbitrary axes is the virtual sphere (Chen et al., 1988), also called the virtual trackball (Henriksen et al., 2004). A further related technique to be mentioned is the Arcball (Shoemake, 1992). In this technique, rotation is controlled by projecting cursor points from the display surface onto a virtual half sphere that is centered on the 3D object (Figure 4.27). Rotation around the object's center is then computed using the projected points. The virtual trackball is typically invisible and does not have to be known to the user in order to be applied effectively. Virtual trackballs allow for rotation around arbitrary axes in a natural manner. They integrate controller and controlled object and can thus be categorized as a direct manipulation technique (Hutchins et al., 1985). In user tests virtual trackballs have been shown to be effective for precise 3D rotation tasks (Chen et al., 1988).

The axis and angle of rotation are determined as follows. Assume that p_2 is the current touch point and p_1 is the previous touch point in a sequence of touch points produced during a stroke. The axis of rotation is then the cross product of the projected points: $a = m(p_1) \times m(p_2) = P_1 \times P_2$ and the angle of rotation is $\theta = \arccos(\frac{P_1 \, P_2}{\|P_1\| \|P_2\|})$. This means that the rotation will be around the z-axis if p_1 and p_2 are both outside the circle (Figure 4.28). If both projected points are located on the hemisphere, then the rotation axis is given by the great circle passing through P_1 and P_2 (Figure 4.27).

4.6.2 Two-Sided Trackball and Gaussian Mapping

As noted in (Henriksen et al., 2004) there is a discontinuity between positions inside and outside the hemisphere. In our pilot tests we found that the change in rotation behavior on entering or leaving the hemisphere during a drag movement can indeed be irritating to users. To avoid this discontinuity and to have a smoother transition from z-axis rotation to x- or y-axis rotation, we changed the mapping from a hemisphere to a Gaussian function (Figure 4.29, upper part), similar to the combined spherical and hyperbolic mapping function of Bell (Henriksen et al., 2004). The specific mapping function we use is:

$$z = g(x,y) = r_1 \exp\left(\frac{-((x-o_x)^2 + (y-o_y)^2)}{2r_2^2}\right) \tag{4.5}$$

where $o = (o_x, o_y)$ is the center of the screen. Note that we do not need to distinguish between the cases of the point being located inside the hemisphere or outside of it. Exclusive rotation in the z axis thus results if input is performed at a sufficient distance from the center of the virtual trackball. The parameter r_1 corresponds to the height of the Gaussian curve and r_2 to its width. For the 480×320 pixel screen of our target device we set these parameters to $r_1 = 120$ and $r_2 = 80$, respectively. A precise correspondence to the size of the object was not found to be necessary.

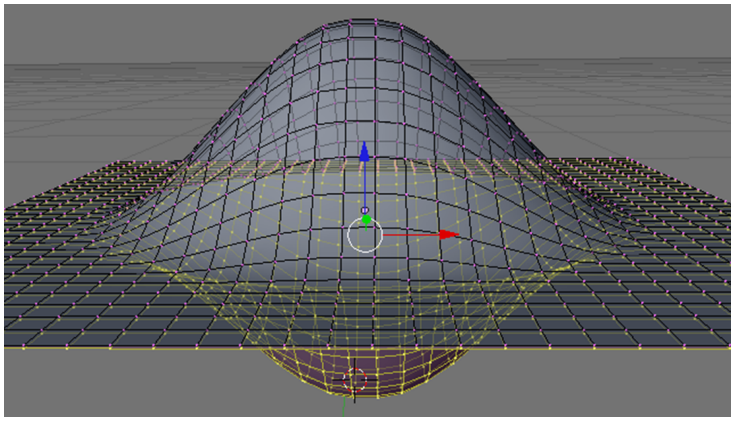


Figure 4.29: To avoid the discontinuity at the half sphere boundary, we use a Gaussian function instead. (Kratz and Rohs, 2010a).

for back-of-device rotation control, the half sphere is extended to the rear of the device

To allow for back-of-device interaction we conceptually extended the half sphere to the backside. For the back side mapping function we inverted the above Gaussian function to extend its range into negative z values as shown in Figure 4.29 (lower part). Depending on whether the touch input event originates from the front side or the back side, the corresponding mapping function is used. For front-side touch events

the mapping function is $z = g(x,y)$, for back-side touch events it is $z = -g(x,y)$.

In addition to the single sided touch input described above and used in the study, we also implemented a simultaneous front- and back-touch input. In this mode the thumb provides the front touch point and the index finger provides the rear touch point. The center of the virtual sphere is then relocated to be in the center between thumb and index finger. This relocation of the virtual sphere allows for 3D rotation in the case of larger displays in which the center of the object might not be reachable with simultaneous front- and back-input.

4.6.3 Tilt-Based Rotation as a Comparison Technique

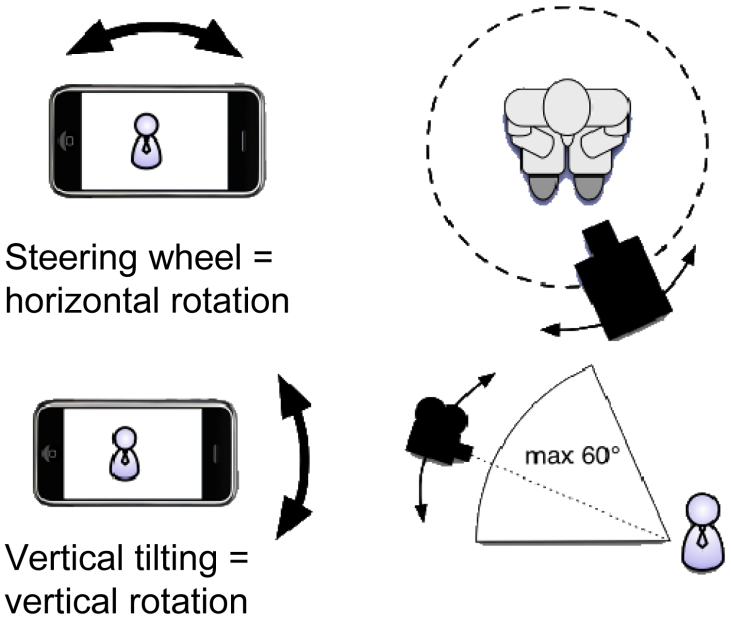


Figure 4.30: Tilt-based navigation using tilt around the z-axis (horizontal camera rotation) and x-axis (vertical camera rotation). (Kratz and Rohs, 2010a).

we chose tilt as a comparison technique due to its popularity

In addition to touch screen input we implemented a tilt-based scheme that uses an accelerometer to sense device orientation (Figure 4.30). The motivation for choosing tilt-input is the widespread use of tilt input for gaming applications on several current smart-phones.

The navigation is based on a steering-wheel metaphor, in which the object rotates horizontally as the user tilts the device around the z-axis. Horizontal rotation of the object is initiated if the device tilt exceeds $\pm 25^\circ$ around the z-axis. The rotation speed was set to be proportional to the tilt angle of the z-axis, with a minimum rotation speed of

$100°/s$ at 25° tilt and a maximum rotation speed of $350°/s$ at 90° tilt. Pre-tests showed that users could comfortably operate the tilt interface with these settings. Moreover, the amount of vertical camera rotation is controlled by tilting around the x-axis. This determines the angle from which the object is observed. The virtual object is always oriented with the right side up, i.e., the y-axis is always oriented opposite to the direction of gravity.

4.6.4 Implementation

The front and rear touch interface for our study was implemented on an iPhone "Sandwich" (see Section 4.5 for more details). The software consists of a dedicated *rear* application that runs that sends rear touch events to the front device, which runs the *front* application used in the user study. The data is transferred via UDP over a dedicated WiFi network. There was no perceivable delay. The front and rear virtual trackballs were implemented using the touch events provided by the front and rear iPhone, respectively.

We obtained Device tilt readings for the x and z axes, using data from the front iPhone's built-in 3axis acceleration sensor, with the following equations to calculate tilt_x and tilt_z:

$$\begin{aligned} \text{tilt}_x &= \arctan\left(\frac{z}{x}\right) \\ \text{tilt}_z &= \arctan\left(\frac{y}{x}\right) \end{aligned} \tag{4.6}$$

The intuition behind Equation 4.6 is that the fractions z/x and y/x represent the influence of Earth's gravity on the acceleration vector of the respective axis. z represents the acceleration measured perpendicular to the device's screen, x the acceleration in direction of the long side of the device and y the acceleration in direction of the short side of the device.

The graphics for the *front* application were implemented using OpenGL ES (Khronos Group, 2012)[W]. We used Blender (Blender Foundation, 2012)[W] to model and export the 3D objects with corresponding normal vectors and texture mapping coordinates. Via a UDP connection, the experimenter can change the number of objects displayed as well as the number of textured faces shown on the objects in the scene. Through this connection, the experimenter can also start and stop trials remotely.

4.6.5 User Study

We conducted a user study in order to measure the effectiveness of 3D object rotation using a front and rear touch virtual trackball as well as tilt. We extended the experimental setup of (Decle and Hachet, 2009), who compared direct and planned 3D object inspection on a touch display.

4.6.5.1 Task

the subjects were tasked with counting the number of textured faces in a 3D scene

Following the approach proposed by (Decle and Hachet, 2009), the test subjects were presented with a freely rotatable 3D scene comprising a regular grid of tetrahedrons. The subjects had to count the number of object faces textured with a logo. Each face of the tetrahedrons was colored in a distinct color, which allowed the test subjects to remember the sides of the objects in a given scene which had already been observed.

Initially, a blank screen was presented to the test subjects. The experimenter could remotely change the settings for grid size and number of textured faces remotely as well as initiate a trial. Once the trial had been initiated, the corresponding scene was presented to the test subject and a timer started. When the test subjects were satisfied that they had found all the textured faces, they reported the number of textured faces found to the experimenter. The task completion time and number of found textured faces found was recorded by the experimenter after each trial.

This type of task is well suited to evaluate 3D rotation input techniques, because it requires the test subjects to look at all the faces of the objects, thus requiring a substantial amount of rotation input from the subjects. The effectiveness of the rotation technique can be deduced from the trial completion times (input speed) and also the error rates (an indicator of mental load).

4.6.5.2 Improvement over Existing Methodology

our methodology allows parametric definition of the task difficulty

In contrast to the previous work, which used a randomly chosen object in their study, we used a regular grid of tetrahedrons (Figure 4.31) as object set. The number of objects and the amount of textured faces that must be found by the test subjects can be programmatically defined. This not only allows us to precisely parametrize the characteristics of the experiment, it also provides a more controllable and comparable scenario for comparison of rotation techniques in future studies.

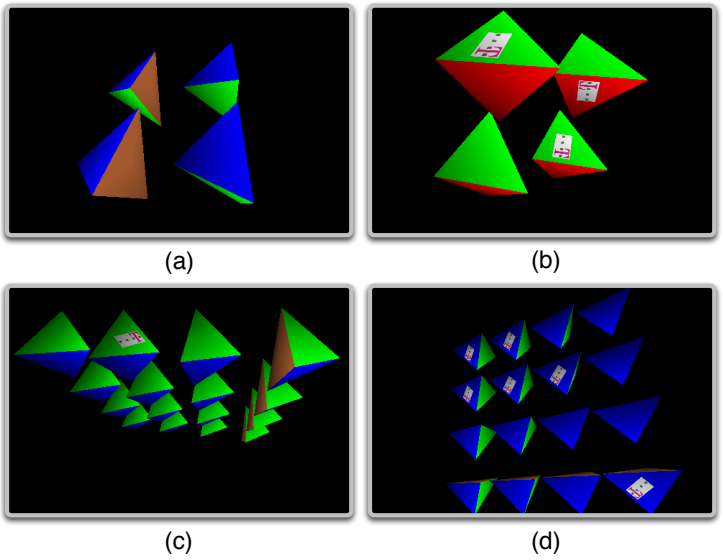


Figure 4.31: Several variants of the tetrahedron grid presented to the user in the study are shown. (a) is 2×2 grid with no textures. (b) is 2×2 grid with 3 of 5 textures visible to the user. (c) shows a 4×4 grid with a single visible texture. (d) shows a 4×4 grid with 6 of 10 textures visible. (Kratz and Rohs, 2010a).

4.6.5.3 Participants, Apparatus, and Design

We recruited 12 test subjects from a university environment. All participants were between 20 and 25 years of age. They all owned a mobile phone, but only one test subject reported to own a touch-enabled smart phone. Only two of the subjects had prior experience with mobile 3D applications. All participants had experience in the use of computers, with 42 percent of them stating advanced or expert skills. The participants received a monetary compensation after the experiment.

We conducted the experiment using the iPhone Sandwich discussed in Section 4.5 for the trials involving the front and rear trackball and an unmodified iPhone 3G for the trials involving the tilt input.

The experiment had a $3\times2\times2$ within groups factorial design. Factors were *input method* for rotation control (front trackball, rear trackball, and tilt), *grid size* (2×2 and 4×4) as well as textured *face count* (3 ± 1 and 9 ± 1). The textured face count was randomly chosen in a ±1 range around 3 and 9 in order to prevent the test subjects from inferring the correct number of textured faces and forcing them to really count all textured faces in the scene presented to them. The order of input techniques was counterbalanced according to a Latin Square design. The trials for each input technique were conducted in sequence, with the

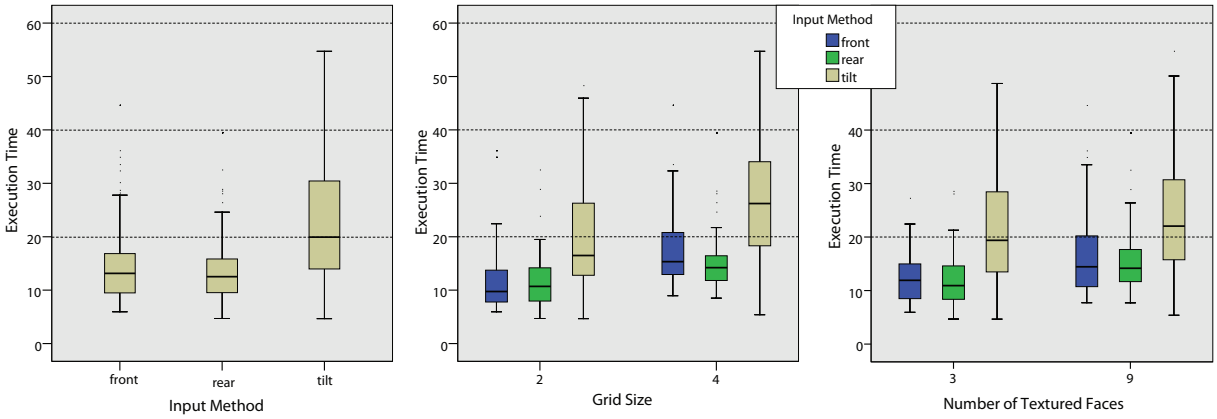


Figure 4.32: Box plots of the task completion times in *s*, grouped by input method (*left*), grid size (*center*) and number of textured faces (*right*). (Kratz and Rohs, 2010a).

order of the grid size and textured face count settings also counterbalanced according to a Latin Square. Each setting was repeated two times resulting in a total of $12 \times 3 \times 2 \times 2 \times 2 = 288$ trials conducted.

measured variables were task completion time, error rate and workload

We measured the task completion time as well as the error rate of the reported textured face counts. Additionally, after each series of trials for a given input technique, the participants were asked to subjectively rate the workload of the input method using the NASA TLX (Hart and Staveland, 1988a) rating scale.

4.6.5.4 Results

Figure 4.32 shows box plots of the task completion times grouped by input method, grid size and number of textured faces.The mean execution time for *front* was 14.72*s*, SD = 7.51*s*, for *rear* 13.89*s*, SD = 6.56 and for *tilt* 23.73*s*, SD = 12.60*s*.

Task Execution Time A univariate ANOVA shows significant effect in the task execution time for input method ($F_{2,287} = 38.73, p < 0.001$), grid size ($F_{1,287} = 34.722, p < 0.001$) and number of textured faces ($F_{1,287} = 16.820, p < 0.001$). A Bonferroni pairwise comparison of input method shows a significant difference for *front* vs. *tilt* and *rear* vs. *tilt* (both $p < 0.001$), but no significant difference between *front* and *rear* ($p = 1.0$).

The error rate, i.e. the number of incorrect responses to the total number of responses, was 60.4% for *front*, 69.8% for *rear*, and 68.6% for *tilt*. Participants were not provided with feedback whether they counted the right number of textured surfaces, because we wanted to measure neutral error performance. The error rates appear quite high, however, the

responses were very close to the actual numbers. The mean square error, i.e., the deviation of the reported count from the actual count, was 0.97 for *front*, 0.70 for *rear*, and 0.94 for *tilt*.

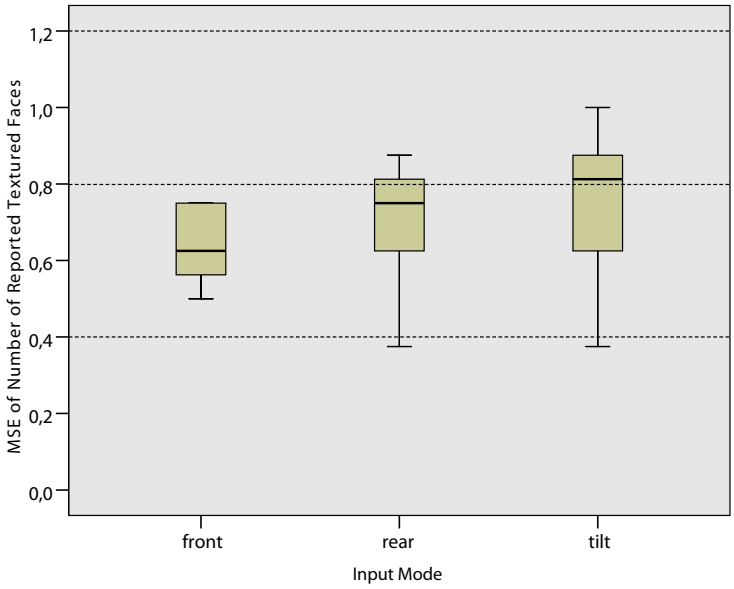


Figure 4.33: The mean square error rates for the faces counting ordered by input technique.

Error Rate of Face Count We computed the Mean Square Error (MSE) for all trials per user for each of the input methods from the difference between the textured faces counted by the users and the actual number of textured faces shown for a particular trial. The results for MSE are shown in Table 4.3.

Table 4.3: Mean Square Error of face counting task, by input technique. (Kratz and Rohs, 2010a).

Input Method	Average	Median	Std. Dev
front	0.97	0.63	1.33
rear	0.70	0.75	0.16
tilt	0.93	0.81	0.89

differences in face counting error are not significant

Notwithstanding, these differences are not significant ($p = 0.737$). Grouping the data by grid size or textured face count shows a slight increase in average MSE for increasing grid size or textured face count, but the differences weren't significant also in those cases.

Our results for error rate suggest that the input methods we tested do not influence the precision of counting the textured faces.

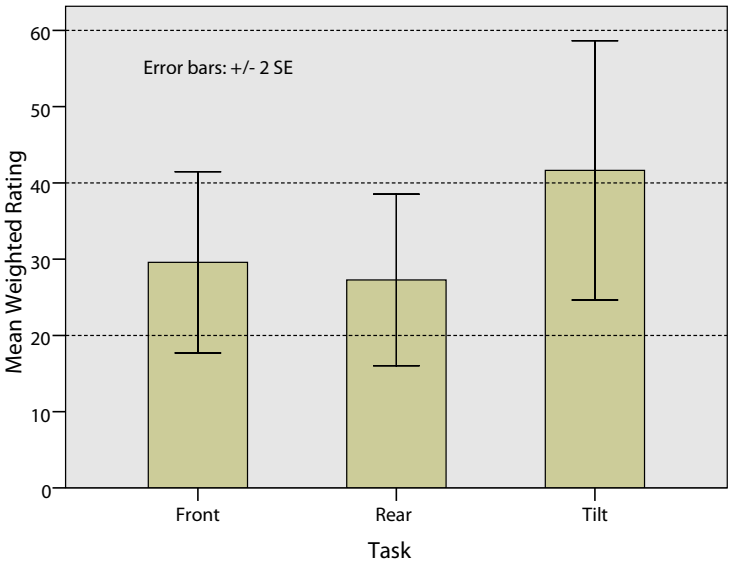


Figure 4.34: The average adjusted workload of the TLX rating scale. (Kratz and Rohs, 2010a).

NASA TLX The average adjusted NASA TLX workload ratings are shown in Figure 4.34. *Front* and *Rear* received very similar workload ratings (*front*: $\mu = 29.6$, $\sigma = 20.6$; *rear*: $\mu = 27.3$, $\sigma = 19.5$). *Tilt* was rated worse ($\mu = 41.6$, $\sigma = 29.4$). These differences are however only indicative and not statistically significant ($F_{2,35} = 1.28$, $p = 0.29$).

4.6.6 Results

In this section, we proposed the extension of the virtual trackball metaphor from single-sided input on a half-sphere to double-sided input on a full sphere .

We described an alternative Gaussian mapping function that avoids the discontinuity of a spherical trackball, similar to Bell's (Hart and Staveland, 1988a) spherical and hyperbolic mapping function (Section 4.6.2).

In a user study (see Section 4.6.5), we found that the virtual trackball with this mapping function is effective for front as well as back touch input and that both outperform a tilt-controlled navigation scheme. Even though the tilt condition avoids occlusion, it is rate controlled and has a more indirect mapping compared to the virtual trackball. NASA TLX showed a slightly higher workload for tilt, however, without being statistically significant. Overall, it is noteworthy that the subjective workload rating was quite low (the scale ranges from 0 to 100, the maximum rating was 41.6 for tilt) yet the error rate was relatively high. This

means that the participants had difficulty in judging their true performance and the result might be an indication that 3D object inspection is inherently difficult.

The results also indicate that occlusion, or "fat finger problem" did not have a significant performance effect on the tasks we measured. This may have been due to the relative rotation of the 3D objects realized with the virtual trackball, which allowed the users fingers to operate at an offset with respect to the rotated objects.

In the future we plan to extend this work by adding precision alignment studies. We also want to investigate the use of multiple fingers on one side of the device as well as the use of fingers on both sides of the trackball. Furthermore we aim to integrate the behind-the-device trackball paradigm in mobile applications scenarios.

4.7 Conclusion

In this Chapter, we presented the concept of Around-Device Interaction (ADI). In Section 4.2 we outlined the design space afforded by ADI and detailed sensor technologies, feedback mechanisms and framework support in this space.

The rest of the chapter presented mobile user interface prototypes developed to explore the ADI concept, *HoverFlow*, *PalmSpace* and the *iPhone Sandwich*. For the most part, these prototypes rely on augmenting the mobile device with one or more additional sensor types.

Our results for HoverFlow show that around-device gestures can be recognized with a very simple set of IR distance sensors. HoverFlow-type user interfaces provide input capabilities to mobile devices in situations where it is impractical or not advisable to touch the mobile device, such as when cooking or driving a car.

Our work on PalmSpace shows that if mobile devices were equipped with depth cameras, they could detect much more complex gestures in comparison to HoverFlow. Our user study results show that complex around-device gestures can, for certain task types, lead to performance benefits over the standard interface mechanisms (e.g. touch screen input) in mobile devices.

The iPhone Sandwich compares pressure input with dual-sided multitouch capabilities. We contributed insights into the correct sensor mappings for touch-based pressure input. Whereas previous pressure-based work mostly relied on pressure input using a stylus with the device placed on a flat surface, we analyzed the properties of touch-based

pressure input with the device in hand-held poses. Our results indicate that touch-based pressure input on mobile devices is performs as well as when the device is placed on a flat surface, except in the case that pressure is applied to the front of the device while the device is held.

Finally, we used dual-sided interaction to extend a well-known 3D rotation interface, the Virtual Trackball. While moving the trackball to the rear of the device does not provide significant gains in performance, it does reduce the occlusion due by the user's hand and fingers, allowing an easier view of the object being manipulated. We also demonstrated that virtual trackballs are superior to tilt-based techniques for 3D object rotation on mobile devices.

Chapter 5

Motion Gestures

"Why do people always gesture with their hands when they talk on the phone?"

—Jonathan Caroll

Modern smart phones are equipped with sophisticated micro-electro-mechanical (MEMS) acceleration and rotation sensors. These sensors are so precise that they can be used for dead-reckoning for mobile navigation. Most mobile user interfaces, however, only make relatively primitive use of the rich data provided by the onboard accelerometers and gyroscopes. Obvious uses such as rotating the user interface to match the device's pose have been implemented. Tilt-labyrinths have been implemented as one of the first applications for the iPhone (Illusion Labs, 2010)[W]. Primitive gesture recognition, such as detecting device shakes, has also been implemented, for example in the iOS 5 SDK (Apple Computer Inc., 2012a)[W].

In this section we present techniques that aim to broaden the scope of uses—motion gestures in particular—for mobile accelerometer and gyroscope data. We begin the chapter with a brief overview of machine learning techniques and algorithms relevant to the sections that follow (Section 5.1). We then present two motion gesture recognition algorithms, the *$3 Gesture Recognizer* (Section 5.2) and *Protractor3D* (Section 5.3). The aim of these algorithms is to provide easy-to-implement gesture recognizers for motion gestures that support researchers and practitioners in developing mobile user interfaces using motion gestures. These algorithms do not rely on any special toolkits or optimizers and can be deployed early on in the development cycle.

To show a "serious" application of accelerometer and gyroscope data on mobile devices, we analyze the feasibility, usability and security of motion gesture-based authentication in Section 5.4.

It has only recently become commonplace to equip smart phones with a gyroscope/accelerometer pair. In Section 5.5 we analyze the potential gains in motion gesture recognition accuracy provided by fusing the two data types—acceleration and rotation—provided by these sensors.

5.1 Machine Learning Foundations

In this section we will briefly describe machine learning algorithms and techniques which are relevant to the following sections of this chapter. The aim is to help readers better contextualize the contributions we made. Furthermore, we wish to describe in additional detail several concepts which are used throughout this chapter and are relevant to gaining a better understanding of gesture recognition in general.

In the following, we will describe how to select features for gesture recognition and normalize them for use in machine learning algorithms. We will then describe the following machine learning algorithms which are used in the later sections of this chapter: k-Nearest-Neighbors, Dynamic Time Warping, Regularized Logistic Regression and Hidden Markov Models.

5.1.1 Features for Gesture Recognition

In his seminal work on stroke recognition, Rubine proposed a set of 13 features (Rubine, 1991) for classification of strokes. The original work was extended by (Long Jr. et al., 2000) with an additional 9 features. In machine learning, a "curse of dimensionality" (Marsland, 2009) exists. One of its implications is that if the data to classify has a high amount of features, a large number of samples must be provided for acceptable training of the classifier. Thus, a classifier using all of Rubine's features will need a large number of training data to function properly. Therefore, recent works in stroke recognition (Wobbrock and Wilson, 2007; Paulson et al., 2008; Li, 2010b) have focused on using the original geometric data for classification, to great success.

Apart from using the raw sensor data from accelerometers and gyroscopes, it is posible to normalize, subsample and transform the data such that the individual datapoints are of equal Euclidean distance from each other. This kind of pre-processing is performed by the *$3 Gesture Recognizer* and *Protractor 3D* (Sections 5.2 and 5.3).

More complex features, partially based on a discrete Fourier-transform of windows of gesture data, can also be calculated. Wheres we have not

made use of these features in projects featured in this dissertation, we will mention these more advanced types of features, suggested by (Ravi et al., 2005) in the context of accelerometer-based activity recognition, for the sake of completeness. Let x_i be a data point of dimensionality k:

- **Average:**

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.1}$$

- **Standard deviation:** The standard deviation equal to the square root of the variance.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{5.2}$$

- **Energy:** The energy is the sum of squared Fourier-components $x(t) = (x_1, x_2, \ldots, x_N)$ of a discrete time signal , divided by its length N:

$$\text{energy} = \frac{\sum_{i=1}^{N} |x_i|^2}{N} \tag{5.3}$$

- **Correlation:** For every possible coordinate axis pair (x, y) we calculate the covariance and the standard deviation and then calculate the ratio. The correlation is a measure of statistical dependence between the data for each of the coordinate axes:

$$\text{corr}(x, y) = \frac{\text{cov}(x, y)}{\sqrt{\text{var}(x)\text{var}(y)}} \tag{5.4}$$

with

$$\text{cov}(x, y) = \sum_{x=1}^{N}(x_i - \mu)(y_i - \mu) \tag{5.5}$$

$$\text{var}(x) = \sum_{i=1}^{N}(x_i - \mu)^2 \tag{5.6}$$

using a large feature set is only useful if a large amount of input samples per gesture is available

As already mentioned earlier in this section, using a large number of features requires a large number of training samples for machine learning techniques to be effective. For gesture-based user interfaces, training samples are usually obtained from users during user studies. Due to temporal constraints, only a limited amount of input gestures can be obtained per gesture class. For projects in this dissertation the average number of recorded entries per gesture class is about 20. With this relatively low amount of training samples, using the raw sample data obtained

from the sensors is usually sufficient for classification. For future longitudinal or massively deployed studies (i.e., through a mobile application store) where the amount of obtained gesture entries is would be much higher, using a larger amount of features may significantly increase the gesture recognition accuracy.

5.1.2 Feature Normalization

feature normalization is required to compensate for differences in magnitude and means between feature types

Features obtained, i.e., from readings of different sensor types can differ significantly in value magnitudes. If left uncorrected, the larger-magnitude sensor readings will "dominate" the readings with a lower magnitude. One possible strategy to counter this effect is to apply feature scaling and mean normalization.The features are normalized such that their values lie within $[-1, 1]$ and the mean is re-centered to 0. The components j of a feature vector $x^{(i)}$ are thus scaled and normalized as follows:

$$\dot{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j} \tag{5.7}$$

where s_j and μ_j are the value range ($max - min$) and the means for component j of $x^{(i)}$.

5.1.3 Machine Learning Algorithms

In the following, we detail several machine learning classification algorithms that are used by the contributions in this chapter. The description of the algorithms is ordered by implementation complexity.

5.1.3.1 k-Nearest Neighbors

One of the simplest machine learning recognizers is k-Nearest Neighbors (kNN). This algorithms is purely data-driven and classifying is done by comparing a sample point s to the k closest labeled points in a training set, and determining the class of s by the majority of the labels of the k nearest neighbors. The usual metric for determining the proximity of s to other data points is the Euclidean distance (L_2 norm):

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{5.8}$$

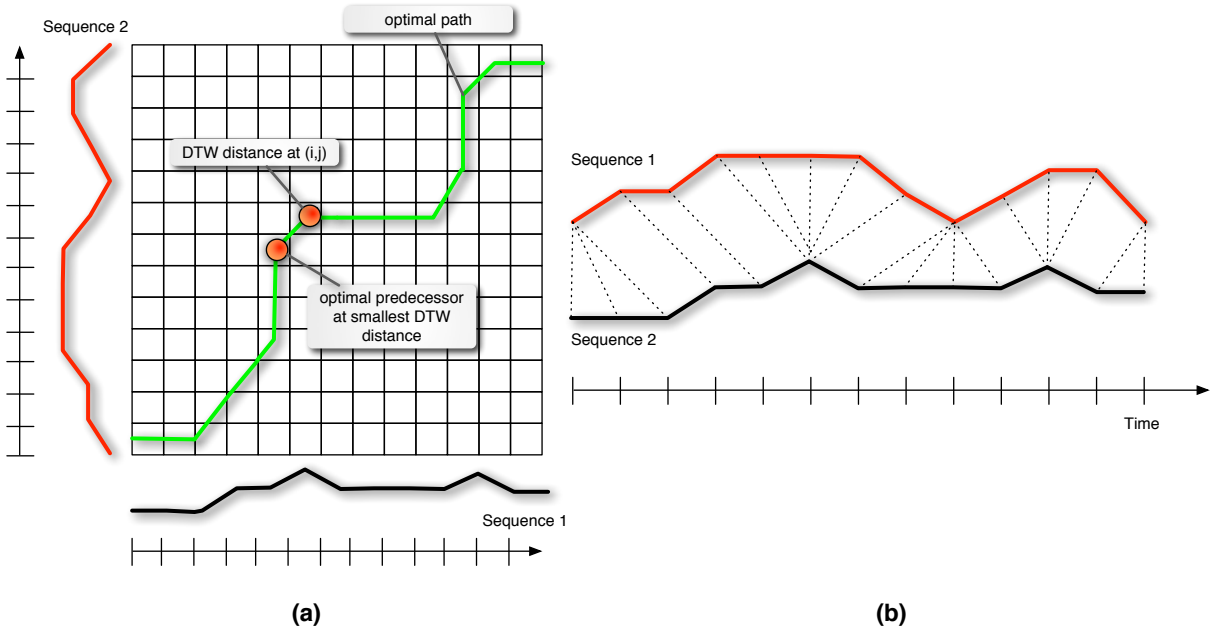


Figure 5.1: (a) Dynamic Time Warping calculates the optimal alignment between two temporally skewed sequence. The DTW update rule ensures that a point (i, j) on the optimal DTW path (green) is always at the minimum distance from its predecessor. **(b)** The result of DTW is the cost-optimal alignment between two time sequences.

Although it is simple to implement, kNN has a number of disadvantages. Since for classification, a new sample has to be compared with all other points in the data set, naïve kNN implementations do not scale well in terms of computational performance for large data sets. In the case of large data sets, k-d trees can be used to reduce the nearest neighbor search time to $O(log\ n)$ (Marsland, 2009), where n is the number of data points in the training set. A further weakness of kNN is that it is not very robust towards noise in the training data. Due to the simple majority-based classification scheme even relatively few spurious data points will seriously degrade the classifier's accuracy.

Both the *$3 Gesture Recognizer* (Section 5.2.2.7) and *Protractor3D* (Section 5.3)use a kNN strategy for classification.

5.1.3.2 Dynamic Time Warping

When dealing with temporal data, such as gestures, it is important to use a technique that is robust towards temporal variations between input samples and training data. Dynamic Time Warping (DTW) is a technique that is used to calculate the minimal-cost alignment between two time sequences (Figure 5.1).

DTW works by finding the optimal path through a $N \times M$ cost matrix D, which is initialized to infinity. The entries $D_{i,j}$ of the cost matrix are generated using the following DTW update function:

$$D_{i,j} = \begin{cases} 0 & i = j = 0 \\ \min(D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1}) + d_{i,j} & i > 0, j > 0 \\ \infty & \text{otherwise} \end{cases} \tag{5.9}$$

Usually, the Euclidean distance (Equation 5.8) is chosen to calculate the distance $d_{i,j}$. Following the path starting at $D_{M,N}$ of minimal neighboring entries through the matrix will yield the optimal alignment path through the matrix. The DTW cost is the matrix entry $D_{N,M}$.

DTW has been used widely in previous work, both in gesture recognition for HCI and also in speech recognition (Sakoe and Chiba, 1978), the field from which the algorithm originated.

Because the number of entries in the DTW matrix rises quadratically in relation to the sequence lengths, DTW does not scale well to long sequences. There have been a number of approaches to compensate for this problem. One possible solution is to window (or "envelope") the search space. There are two very common envelopes called the *Sakoe-Chiba Band*, and *Itakura Parallelogram* (Sakoe and Chiba, 1978; Itakura, 1975). Another approach, called *FastDTW* (Salvador and Chan, 2004), is to approximate the full DTW calculations by subsampling the original DTW matrix.

Our work on gesture-based authentication in Section 5.4 uses DTW as one of the main machine learning algorithms. In Section 5.5 the performance of DTW is evaluated with combined accelerometer and gyroscope data.

5.1.3.3 Logistic Regression

Logistic regression is a simple but powerful supervised learning classifier that is very popular in the machine learning community. Logistic Regression, in essence, represents the operation of a single neuron of an Artificial Neural Network.

Hypothesis The hypothesis of Logistic Regression is given by

$$h_\theta(x) = g(\theta^T x) \tag{5.10}$$

where x is a feature vector and the model, θ, is a parameter vector the length of which corresponds to the number of features. The outputs of $h_\theta(x)$ are constrained to $0 \leq h_\theta(x) \leq 1$, in order to classify two distinct classes, 0 and 1. This behavior is obtained by choosing the Sigmoid (also known as Logistic) function for g:

$$g(z) = \frac{1}{1+e^{-z}} \tag{5.11}$$

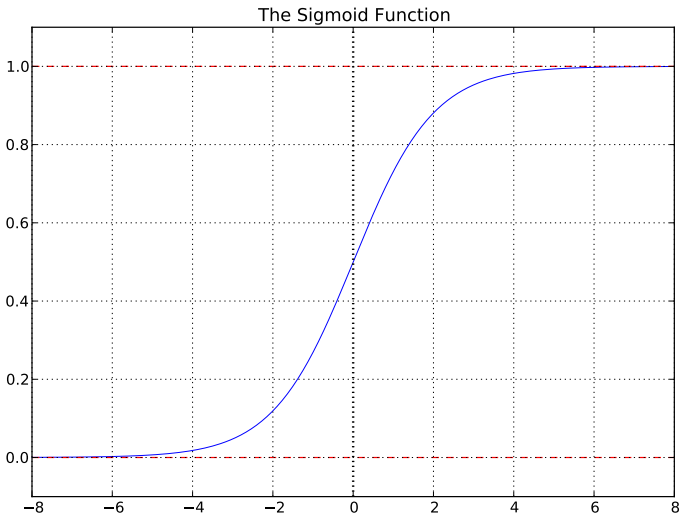


Figure 5.2: Plot of the Sigmoid function. Values for $x > 0$ asymptote at 1, values for $x < 0$ asymptote at 0.

Figure 5.2 shows a plot of the Sigmoid function. The intuition behind using the Sigmoid function is that we would like the classifier to output the probability of class 1 given an input x and the parameter vector θ, i.e. $h_\theta(x) = P(y = 1|x;\theta)$. Thus, we can predict 1 if $h_\theta(x) \geq 0.5$ and 0 if $h_\theta(x) < 0.5$, because $h_\theta(x) \geq 0.5$ whenever $\theta^T x \geq 0$ and $h_\theta(x) < 0.5$ whenever $\theta^T x < 0$.

Training the Classifier In order to train the model parameters θ using input data vectors X and labels $y \in \{0,1\}$ we need to minimize the following cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \tag{5.12}$$

where

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \tag{5.13}$$

Using the logarithm guarantees that $J(\theta)$ remains convex, i.e. the global maximum is guaranteed be found using gradient descent. Also, for $y = 1$, $-\log(h_\theta(x))$ captures the intuition that there should be a near infinite penalty when $h_\theta(x) \approx 0$ and a near zero penalty when $h_\theta(x) \approx 1$. The reverse is the case with $-\log(1 - h_\theta(x))$, for $y = 0$. The cost function can be further simplified as follows:

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] \tag{5.14}$$

The minimum of the cost function can be found using gradient descentwith the gradient of $J(\theta)$:

$$\frac{\delta}{\delta_{\theta_j}} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)} \tag{5.15}$$

The gradient descent update function can then be applied repeatedly up to a given convergence criterium to optimize $min_\theta J(\theta)$ for all components j of θ:

$$\theta_j = \theta_j - \alpha \frac{\delta}{\delta_{\theta_j}} \tag{5.16}$$

α is a parameter that controls the speed of gradient descent. Since α can be difficult to choose correctly, more advanced gradient descent algorithms, such as the Broyden-Fletcher-Goldfarb-Shannon (BFGS) algorithm[1](Fletcher, 1987) perform calculations on the input data in order to determine useful values for α.

[1]For the application of regularized Logistic Regression to motion gesture recognition featured in Section 5.5, we used the BFGS (Fletcher, 1987) implementation from the *Scipy.optimize* Python library to minimize the cost function. The specific function call we used was `scipy.optimize.fmin_bfgs(...)`.

Predictions To make a prediction given a new input vector x, we simply calculate

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{5.17}$$

to obtain $p(y = 1|x;\theta)$.

Multi-Class Classification LR classifiers can be used for multi-class classification. Multi-class classification can be achieved using a one-vs-all approach. For this, we train one logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$. In order to make a prediction for a new input x, we pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

Decision Boundary and Regularization The version of Logistic Regression used in this section only determines a linear decision boundary. To obtain more complex decision boundaries, additional, higher-order features can be generated. For instance, if we have the features x_1 and x_2 a more complex boundary can be calculated by generating higher-order features such as $x_1x_2^2, x_1^2x_2, x_1^2, x_2^2$. This will also result in a higher dimensionality for θ.

In order to avoid overfitting due to too many features, *regularization* can be applied to the cost function in order to "weaken" the effects of θ. This will lead to a more "general" hypothesis and reduce the problem of over fitting. The regularized cost function for linear regression is thus defined as:

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{5.18}$$

The gradient for regularized Logistic Regression is defined as[2]:

$$\frac{\delta}{\delta_{\theta_j}} J(\theta) = \begin{cases} \frac{1}{m}\sum_{i=1}^{m} h_\theta(x^{(i)} - y^{(i)})x_j^{(i)} & \text{if } j = 0 \\ \frac{1}{m}\sum_{i=1}^{m} h_\theta(x^{(i)} - y^{(i)})x_j^{(i)} - \frac{\lambda}{m}\theta_j & \text{if } j > 0 \end{cases} \tag{5.19}$$

[2]The distinction for $j = 0$ is made because, by convention, every input vector x is padded such that $x^{(0)} = 1$.

5.1.3.4 Hidden Markov Models

Hidden Markov Models (HMMs) are a statistical machine learning technique designed for the classification of time-series data. Developed originally for speech recognition applications (Rabiner, 1990), HMMs have become one of the most used machine learning techniques in research.

The basic premise of HMMs is that by looking at a sequence of inputs, or *observations*, we can calculate the probability of the model being in a certain state, e.g, "gesture recognized" or "no gesture recognized". Thus, observations are not uniquely tied to a specific state (Marsland, 2009). This makes HMMs very robust towards variations in the time-series data. More formally, HMMs are composed of:

- an observation alphabet V.
- an underlying (hidden) transition system with a set of states $S = \{q^{(0)}, \ldots, q^{(N)}\}$.
- a probability distribution matrix A, where the entries $a_{i,j}$ describe the probability of a transition from state q_i to state q_j given the current observation:

$$p(q_n^{(j)}|q_{n-1}^{(i)}) \equiv a_{i,j} \tag{5.20}$$

- an emission probability distribution $B = b_i(v)$ which tells us the probability of observing a symbol $v \in V$ at state $q^{(i)}$:

$$p(v|q^{(i)}) \equiv b_i(v) \tag{5.21}$$

- an initial state distribution $\pi = \{\pi^{(i)}\}$ with $p(q_0^{(i)}) \equiv \pi^{(i)}$.

The compact notation for an HMM is therefore:

$$\lambda = (A, B, \pi) \tag{5.22}$$

There are three fundamental problems for HMMs. The first problem is called the *Evaluation Problem*. The question is how to efficiently calculate the probability of an observation sequence $O = o_1, o_2, \ldots, o_N$ given the model λ. An efficient solution to this problem is given by the Forward Algorithm (Marsland, 2009).

The second problem of interest is the *Decoding Problem*. Given an HMM λ , what is the optimal state sequence $Q = q_1, q_2, \ldots, q_N$ given an observation sequence $O = o_1, o_2, \ldots, o_N$? In other words, we want to find

out what state sequence best explains the given observation sequence. The Viterbi Algorithm solves this second problem (Viterbi, 1967).

To train an HMM, we need to adjust the model parameters of $\lambda = (A, B, \pi)$ such that $P(O|\lambda)$ is maximized, e.g., training an HMM-based gesture recognized with labeled sample gestures, in order to increase its gesture recognition accuracy. This is referred to as the *Training Problem*. Through training, we obtain models that yield a high observation probability for sequences similar to those that they were trained with. The training problem can be solved using the Baum-Welch Algorithm (Marsland, 2009).

In Section 5.5 we examine the performance HMMs for gesture-based authentication.

5.2 The $3 Gesture Recognizer: Simple Gesture Recognition for Devices with 3D Accelerometers

The $3 Gesture Recognizer ($3GR) is a simple but robust motion gesture recognition system for input devices featuring 3D acceleration sensors (Kratz and Rohs, 2010b). The algorithm is designed to be implemented quickly in prototyping environments and intended to be device-independent and does not require any special toolkits or frameworks, but relies solely on simple trigonometric and geometric calculations.

5.2.1 Motivation and Related Work

Gesture input for mobile devices can be a way to overcome their restricted input capabilities and small display sizes. One of the reasons for this is that the range of input during gesture entry is not restricted by the size of device.

the $3 Gesture Recognizer extends Wobbrock's $1 Gesture Recognizer

The $3 Gesture Recognizer is based on previous work[3] by Wobbrock et al. (Wobbrock and Wilson, 2007), who developed a simple "$1 Recognizer" using basic geometry and trigonometry. The "$1 Recognizer" is targeted at user interface prototyping for 2D touch-screen-based gesture recognition and therefore focuses on ease of implementation on novel hardware platforms. The authors even provide a pseudocode implementation of the complete recognizer in the paper. Our contribution is in extending and modifying Wobbrock et al.'s algorithm to work with 3D acceleration data. Instead of capturing exact pixel positions on

[3]For more related work on accelerometer-based gesture recognition, consult Section 2.5.3

a touch screen, acceleration data is of much lower quality because it is prone to noise. In addition, drift error accumulates as the path of a gesture entry is integrated. We extend Wobbrock's original algorithm with a scoring heuristic to lower the rate of false positives. Using actual user input, we present an evaluation of the performance of Wobbrock's modified algorithm and show that this method is well suited to implement 3D gesture recognition in rapid prototyping environments.

$3GR can recognize true 3D motion gestures

The major contribution of this work is the creation of a simple gesture recognizer that is designed to recognize "true" 3D Gestures, i.e., gestures which are not limited to shapes that can be drawn in a 2D plane. The advantage of true 3D gesture recognition is that more natural movements, such a tennis serve or boxing punches can be input by the user.

$3GR can be implemented quickly in prototyping scenarios

Like the "$1 Recognizer," our approach is quick and cheap to implement, does not require library support, needs only minimal parameter adjustment and minimal training, and provides a good recognition rate. It is therefore very valuable for user interface prototyping and rapid application development. It can also easily be integrated into mobile interfaces that take advantage of other modalities, like speech, or touch-based interaction with RFID/NFC.

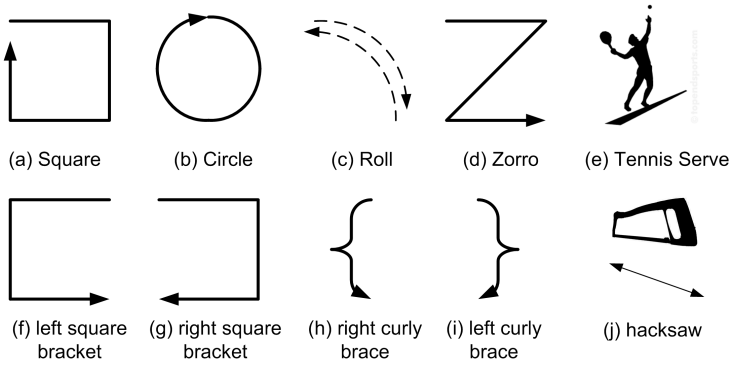


Figure 5.3: The reference gesture vocabulary containing the gesture classes used for the preliminary evaluation. (b) describes a clockwise circular motion, (c) a wrist rolling motion (e) stands for a gesture resembling the serve of a tennis player and (j) represents a repeated rapid forward-backwards motion. (Kratz and Rohs, 2010b).

5.2.2 Implementation

Extending the work by (Wobbrock and Wilson, 2007), we present a gesture recognizer that can recognize gestures from 3D acceleration data as input. To test our algorithm we used acceleration samples obtained from a Nintendo Wii Remote (WiiMote) (Lee, 2008). The WiiMote features an ADXL 330 Accelerometer (Analog Devices Inc., 2012)[W] and

the acceleration data can be sent, as in our case, via a Bluetooth connection to a PC. Our algorithm is by no means limited to the WiiMote. It can be used in any acceleration-enabled device, for instance modern smart-phones such as the Nokia N95 or Apple's iPhone.

5.2.2.1 Gesture Trace

the gesture trace contains the differences in acceleration vectors over time

In contrast to (Schlömer et al., 2008), we do not modify or pre-process the raw acceleration data in any way (filtering, smoothing, etc.). To determine the current change in acceleration, we subtract the current acceleration value reported by the WiiMote from the previous one. We thus obtain an *acceleration delta*.

By summation of the acceleration deltas, we obtain a *gesture trace* T (which can be projected and plotted into a 2D plane to obtain a graphical representation of the gesture (Kallio et al., 2006)).

Figure 5.4: UML object diagram showing the relationship between the gesture class library, gesture classes and gesture traces.

5.2.2.2 Gesture Class Library

the gesture class library contains training gestures for every gesture class

The *gesture class library* L contains a predefined number of gesture traces for each *gesture class* G. We also refer to these traces as *training gestures*. Figure 5.4 depicts a graphical representation of these relationships.

5.2.2.3 Gesture Recognition Problem

The basic task of our algorithm is to find the best matching gesture class G from a *gesture class library* L, for a given input gesture I. An example set of gesture classes is given in Figure 5.3.

To find a matching gesture class, we compare the trace t_i of I to the traces of all training gestures $t_{G_k} \in L$ and generate a score table that lists the comparison score of t_i and each t_{G_k}. A heuristic then is applied to the score table to determine if a gesture has been recognized.

The following sections describe the steps of the $3 Gesture Recognizer in detail.

Algorithm 1 ***Gesture Trace Resampling :***

Input: gesture trace t, desired number of sample points in normalized trace length N

Output: length-normalized gesture trace t_N

Calculate the total length L of the gesture trace, then calculate the increment length $I = L/(N-1)$
n = 0
while $n < N$ **do**
  start a new segment s_n
  while the current segment's length $l_{s_n} <$ I **do**
    – add points of the original trace to a segment s_n
    – backtrack along the excessive distance $l_{s_n} - I$ along the original trace using the unit vector obtained from the difference of the last two points added to s_n
    – append s_n to t_N
    – $n = n + 1$
  end while
end while

Our resampling algorithm uses an approach slightly different to Wobbrock's version. In this way all points from the original gesture trace t are consumed, and we obtain a resampled trace t_N, which consists of N equidistant points.

5.2.2.4 Resampling

for classification, the number of points in an input gesture must be resampled to length N

In order for our trace to be classifiable by the gesture recognition algorithm, it needs to be resampled so that the gesture trace has a fixed number N of equidistant sample points. This is because the gesture input duration and movement speeds can vary between users, even for the same intended gesture. In our case $N = 150$, which is slightly above the average amount of acceleration deltas received while users enter a gesture with the WiiMote. Setting N to a lower value decreases the gesture recognition precision, while choosing a higher N just increases the computation time for gesture recognition, without a significant gain in accuracy.

The Algorithm 1 box shows a pseudocode representation of the resampling algorithm we developed.

5.2.2.5 Rotation to "Indicative Angle" and Rescaling

rotating to indicative angle aims to correct for rotational differences between gestures

To correct for rotational errors during gesture entry, the resampled gesture trace T_N is rotated once along the gesture's *indicative angle*. Like Wobbrock, we define the indicative angle as the angle between the gesture's first point p_0 and its centroid $c = (\bar{x}, \bar{y}, \bar{z})$. The angle is determined by taking the arccosine of the normalized scalar product of p_0 and c:

$$\theta = acos(\frac{p_0 \bullet c}{\|p_0\| \, \|c\|}) \tag{5.23}$$

The rotation along the indicative angle is then performed using the unit vector of the vector orthogonal to p_0 and c. The orthogonal vector is obtained using the cross product of P_0 and c:

$$v_{axis} = \frac{p_0 \times c}{\|p_0 \times c\|} \tag{5.24}$$

Using v_{axis} as axis and θ as angle, we generate a rotation matrix for rotation around an arbitrary axis to rotate all points of T_N to obtain T_{N_θ}.

scaling needs to be performed to correct for size differences

After rotation, T_{N_θ} is scaled to fit in a normalized cube of 100^3 units, to compensate for scaling differences between gestures. The algorithm has now finished pre-processing our the original user input and has obtained a gesture T_M, which is ready for matching with candidate gestures from the gesture class library.

5.2.2.6 Golden Section Search for Minimum Distance at Best Angle

Like Wobbrock, we use the average MSE (Mean Square Error) to calculate the path distance d between T_M and candidate gesture from the gesture class library. We convert the path distance to a $[0, 1]$ scale using a version of Wobbrock's scoring equation adapted to three dimensions, where d signifies the path distance and l the side length of the cube that T_M was scaled to in the rescaling step.

$$Score = 1 - \frac{d}{0.5\sqrt{3l^2}} \tag{5.25}$$

Golden Section Search is used to approximate the optimal rotation angles

Following Wobbrock's discussion of rotation invariance of path distances, we have adapted a Golden Section Search (GSS) using the Golden Ratio $\varphi = 0.5(-1 + \sqrt{5})$ to approximate the local minimum path distance within an angular range of $[-180° \ldots 180°]$, for rotation around the three axis of the coordinate system, signified by the angles α, β and γ. We define a minimum cutoff angle for GSS of $2°$, in order

to guarantee that the approximate minimum is found after exactly 11 iterations of GSS. We chose a rather large angular search range as we could not confirm that Wobbrock's observation that the minimum distance consistently occurs within $\pm 45°$ from the indicative angle applies for gestures in 3D space.

It seems very likely that this does not hold true for gestures input in 3D[4]. Figure 5.5 shows the presence of a very distinct local minimum in the vicinity of the indicative angle, notice the steep drop-off in distance in the vicinity $\alpha = \beta = 180°$ which represents the center of the search-space.

a score table is built from the pairwise distances of the gesture entry from all template

The GSS-Based minimum distance approximation is repeated for each trace for every gesture class in the gesture class library. We thus obtain a table of scores with classes of likely matches.

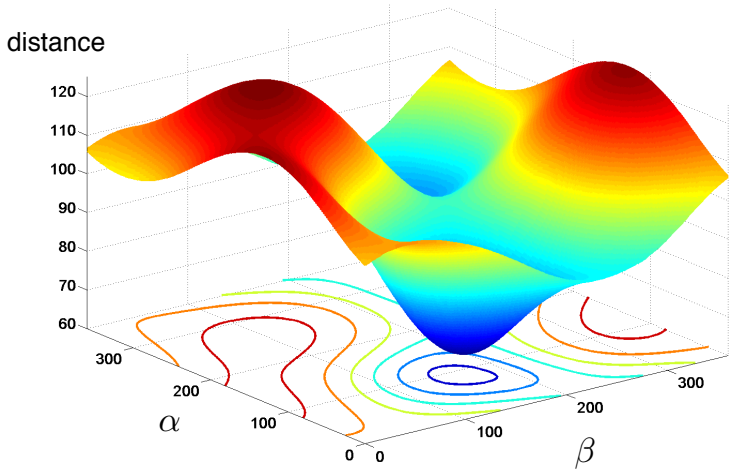


Figure 5.5: A 3D height map of the classification distance depending the rotation of the gesture trace. The vertical axis represents distance, whereas the two horizontal axes represent the rotation angles α and β. In this projection, γ is fixed at $0°$. Section 5.3 contains a closed-form solution to the problem of finding the optimal rotation.

5.2.2.7 Scoring Heuristic

Wobbrock's original algorithm did not feature a heuristic to reduce the occurrence of false positives, which is a common problem for simple gesture recognition algorithms operating on large gesture vocabularies (Wobbrock and Wilson, 2007).

applying our heuristic improves the precision of the gesture recognition

The matches obtained from gestures entered as 3D acceleration data

[4]After the $3GR, we developed Protractor3D which solves the matter of finding the correct rotation for 3D gestures by applying a closed-form solution to the problem, see Section 5.3 for more details.

are not as precise as strokes entered on a touch screen. To compensate for the weaker matches, we have developed our own scoring heuristic, which processes the score table described in the previous section.

Using this heuristic, we achieved a considerable reduction of false positive recognitions compared to Wobbrock's original strategy of selecting the gesture candidate with the highest matching score to determine the recognized gesture. After sorting the score table by maximum score, our heuristic determines the recognized gesture with the following rules:

Algorithm 2 ***Scoring Heuristic :***

input score table S containing all the values of the distance between the input gestures and the templates in L

output the id of the recognized gesture or "gesture not recognized"

- ε is defined as the threshold score.
- **if** the highest-scoring candidate in the score table has a score $> 1.1\varepsilon$, **then** return this candidate's gesture ID.
- **else if**, within the top three candidates in the score table, two candidates exist of the same gesture class and have a score $> 0.95\varepsilon$, respectively, **then** return the gesture ID belonging to these two candidates.
- **else**, return "gesture not recognized".

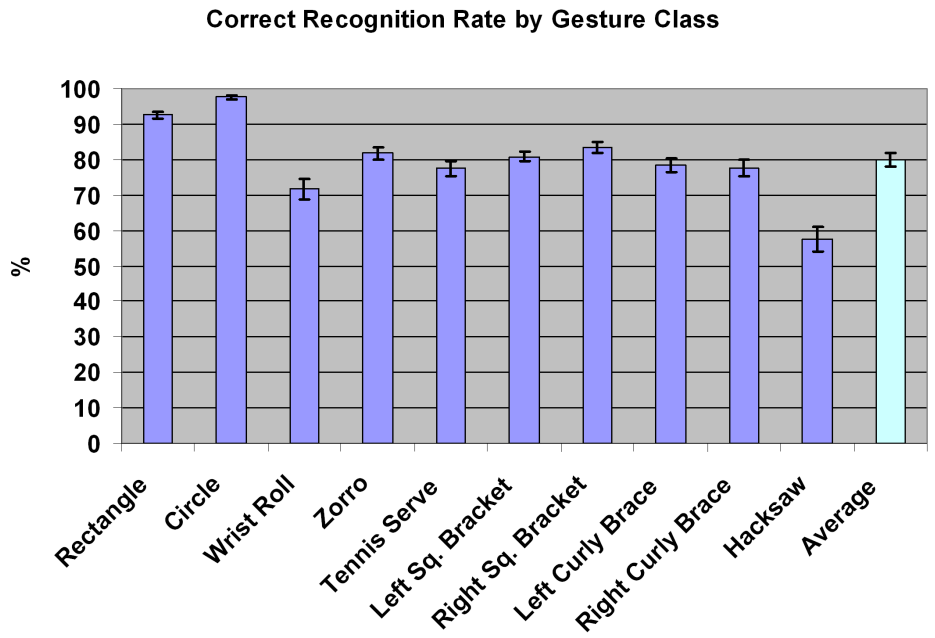
5.2.3 Evaluation

To get an initial estimate of the gesture recognition performance of our method, we evaluated the 3$ gesture recognition algorithm with twelve participants, who were compensated for their effort.

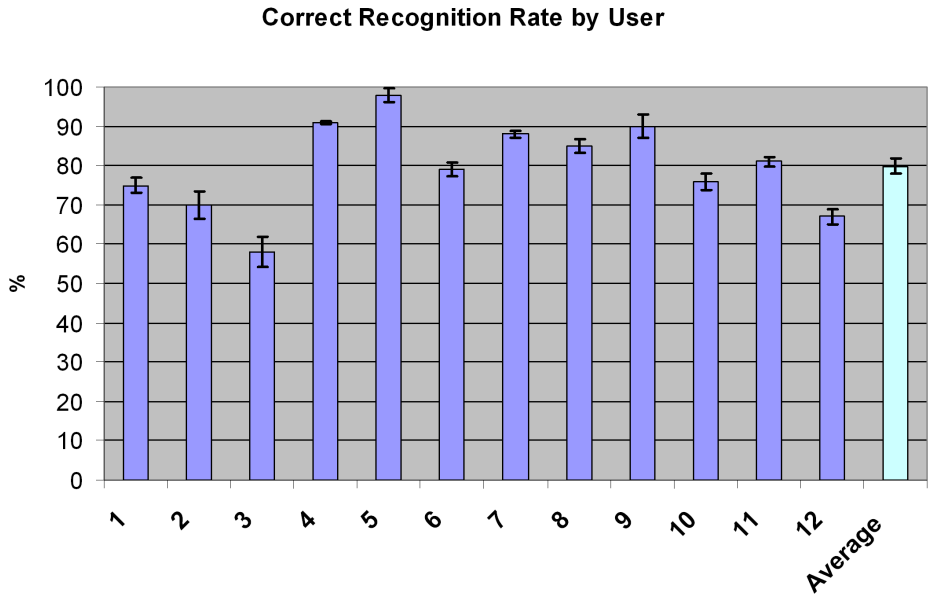
Our reference gesture vocabulary contained all of the gestures used by (Schlömer et al., 2008) as well as a subset of Wobbrock's unistroke gestures (Wobbrock and Wilson, 2007), as displayed in Figure 5.3, totaling 10 unique gesture classes. Each user was asked to enter each of the 10 gestures 15 times using a WiiMote. The gesture data was recorded and stored on a PC.

gesture recognition was performed offline

The actual gesture recognition was performed offline using the stored gestures entered by the users. From each gesture class, the first five entered gestures of a particular gesture class were chosen as the training set for that class. The remaining gestures were input into our gesture



(a) Correct Recognition Rate by Gesture Class



(b) Correct Recognition Rate by User

Figure 5.6: Average correct recognition rates with standard error, sorted by gesture class (top) and by user (bottom). (Kratz and Rohs, 2010b).

recognition algorithm. Knowing the gesture class of the tested gesture beforehand, we recorded the number of times the gestures were correctly recognized, incorrectly recognized or not recognized at all.

Our evaluation resulted in average (correct) recognition rate of 79.83 percent. Between test subjects, the recognition rate varied between 58 and 98 percent, with a standard deviation of 11.4. As can be seen in Figure 5.6, the recognition rate was fairly constant across all users and gestures, with gesture class (b) having the highest average recognition rate and gesture class (j) being the most error-prone gesture. We speculate that the low recognition rate of gesture class (j) is due to the ambiguity of that gesture, as users varied the "sawing motion", which they were expected to perform, considerably. Notably, users commented that gesture classes (h) and (i) were the most uncomfortable gestures to perform. Furthermore, our results indicate that our scoring heuristic functioned acceptably, as only about 8 percent of all detected gestures were false positives.

Our gesture recognition algorithm yielded a lower correct recognition rate than those obtained with the system featured in (Schlömer et al., 2008). In spite of this, we deem our correct recognition rate to be fully acceptable given that we used substantially simpler methods, and, which is more important, twice as many gesture classes with a significantly smaller gesture training set per class to achieve this recognition rate.

It is likely that the nearly 20% lower recognition rate of our method compared to (Wobbrock and Wilson, 2007) is influenced by the following factors. Gestures in 3D space are more difficult for a human to re-produce perfectly than in 2D, even for simple 2D shapes. More important, the equipment which we used to capture the gesture information was far from perfect, and may have contributed to the reduced recognition rate. Instead of capturing exact pixel positions on a touch screen, we had to rely on acceleration data. Data of this kind is of much lower quality as it is prone to noise, and additionally, drift error accumulates as the path of a gesture entry is integrated.

5.2.4 Limitations

$3GR has lower performance than more complex algorithms

As it is a simple algorithm, the $3 Gesture Recognizer has several limitations. For one, in contrast to more refined methods such as those based on HMMs, it cannot be used to detect gestures in a continuous motion stream. Only gestures which are explicitly started and stopped by the user, or have automatically computed start and end points, can be recognized.

precision decreases and computational cost increases with gesture vocabulary size

A further limitation is the size of the gesture vocabulary. Not only does the number of false positive recognition rise together with the size of the gesture vocabulary, but the computational overhead ($O(N \cdot M)$, where N is the amount of motion samples, and M is the number of training gestures in the gesture class library), due to the intensive use of trigonometric functions, increases as well, which limits the maximum practicable size of the gesture vocabulary to about 10-15 gestures for current devices.

These limitations, however do not represent an impediment for the use of our recognizer in its target domain—rapid prototyping of gesture-based interfaces.

5.2.5 Summary and Future Work

In this Section, we presented a simple, easy-to-implement gesture recognizer for input devices equipped with 3D acceleration sensors. The idea behind our gesture recognition algorithm is to provide a quick and cheap way to implement gesture recognition for true 3D gestures (such as the reference gesture in Figure 5.3 (e)), for devices equipped with 3D acceleration sensors. Our method does not require any advanced software frameworks or toolkits. An example application area for our gesture recognizer is user interface prototyping.

acceptable recognition rates and low implementation time

In an initial evaluation of our algorithm, we obtained gesture recognition rates which are comparable to those of more advanced approaches. The advantage of our system is that it is specifically targeted for use in prototype environments, in which gesture-based interfaces or multimodal interfaces using gestures as one of many input components, are needed that provide quick results with less coding and minimal training data.

5.3 Protractor 3D

In Section 5.2 we discussed the $3 Gesture Recognizer ($3GR), a lightweight gesture recognizer of 3D motion gestures for rapid application development and prototyping purposes. A major drawback of the $3GR was that it could only heuristically approximate a good rotational alignment between template gestures and an input gesture by applying Golden Section Search (Section 5.2.2.6).

a closed-form solution to the rotation problem exists under certain assumptions

However, if we assume that the point sets of a template and an input gesture are very similar (which they should be if the input gesture is similar to the template), a closed-form and optimal solution exists. A new classifier, which we call Protractor3D (Kratz and Rohs, 2011) extends the $3GR with this closed-form solution, and we show that the obtained precision of the algorithm is significantly better than that of the $3GR.

Protractor3D retains most of the desirable properties of $3GR and has a higher precision

Although Protractor3D is slightly more complex to implement than the $3GR and needs the support of a linear algebra library to calculate Eigenvalues and Eigenvectors[5], we feel that inherits most of the desirable properties, such as ease of implementation, easy integration in rapid-prototyping scenarios and simple calibration of the relevant parameters. What is most important, however, is that just like the $3GR, Protractor3D requires only a low amount of training samples to function well.

5.3.1 Motivation and Related Work

Motion gesture recognition can be implemented using advanced machine learning techniques such as Support Vector Machines (SVM) (Marsland, 2009) or Parzen Windows Classifiers (PWC) (Duda et al., 2000). However, as a growing body of recent work shows, ad-hoc or simpler classification techniques that are mostly data-driven are growing in popularity.

On of the reasons for this is that adapting state-of-the art machine learning techniques can be cumbersome. Most of them require special libraries or entail a high implementation effort, due to the high mathematical and algorithmic complexities of the approaches used. For instance, while the theoretical concepts of SVMs should be accessible to most developers, training SVMs requires a Quadratic Programming (QP) solver (Marsland, 2009), which is nontrivial to implement, and is in most cases supplied by specialized third-party toolkits. Optimizing such advanced classifiers so that they achieve their best classification performance is likely to be beyond the grasp of a standard mobile application developer, as this requires detailed knowledge of the workings of a particular classifier and also a great deal of experience in machine learning.

simple motion gesture recognizers usually achieve an average precision of over 90%

Recent publications (Liu et al., 2009b; Schlömer et al., 2008) have focused on "simple" gesture recognition of 3D motion gestures. We

[5](Horn, 1987) describes an explicit solution to finding the Eigenvalues and Eigenvectors, in case a linear algebra library is unavailable.

note that this previous body of classifiers performs relatively well, with average correct recognition rates of 90%.

Protractor3D aims to significantly improve the computational requirements and the precision of data-driven motion gesture recognizers. A major problem of existing recognition techniques for motion gestures is that the gestures cannot be recognized in a rotation-invariant way. For instance, a symbolic gesture, such as drawing a circular shape in the air may be performed either in the vertical or horizontal plane or simply using a non-standard grip on the mobile device. Such rotated input data cannot be matched accurately with training templates that were entered using a different device posture. In the following, we refer to this problem as the *template–gesture rotation problem*. Protractor (Li, 2010b), an extension to Wobbrock's $1 gesture recognizer, addressed this problem in 2D by extending the original algorithm with a closed-form solution to finding the optimal rotation between template and entered gesture.

Protractor3D solves the *template–gesture rotation problem*

Protractor3D is a gesture recognition algorithm based on the closed-form solution to the absolute orientation problem for measurements in two 3D coordinate systems (Horn, 1987). This technique solves the template–gesture rotation problem for motion gesture recognition. An evaluation we conducted shows that, using actual input generated by test subjects, Protractor3D significantly increases gesture recognition accuracy in comparison to an implementation that does not apply rotation correction when matching entered gesture data to gesture templates.

5.3.2 Optimal Solution to the Gesture–Template Rotation Problem

The solution to the absolute orientation problem for points measured in two different Cartesian coordinate systems (Horn, 1987) can be applied directly to solving the gesture–template rotation problem. In the following we will only discuss Horn's technique in sufficient detail to implement a rotation-invariant recognition algorithm for motion gestures. For the relevant mathematical derivations and correctness proof we refer the reader back to Horn's original work.

5.3.2.1 The Template–Gesture Rotation Problem

Let G be a gesture class represented by a set of template gestures. Each template is a sequence of 3D accelerometer values. We consider a gesture $\mathbf{g}$ entered by the user, again a sequence of 3D accelerometer values.

Ideally, this gesture is entered in a way similar to a template gesture $\mathbf{t} \in C$. However, it is very likely that the posture with which the user enters $\mathbf{g}$ is different from the one with which the user entered the template $\mathbf{t}$. This can be due to variations of the user's grip on the mobile device or also differences in the user's own body posture, such as being seated or standing, while entering $\mathbf{g}$. For gestural interfaces, it is in many cases undesirable to constrain the device posture for gesture entry, as for many applications the device's movement ought to determine the type of gesture entered, not the device's posture. To solve the template–gesture rotation problem, we must find a rotation R that minimizes the sum of squares error between $\mathbf{g}$ and $\mathbf{t}$.

$$\sum_{i=1}^{n} \|\mathbf{t}_i - R(\mathbf{g}_i)\|^2 \tag{5.26}$$

between the points of $\mathbf{t}$ and the rotated points of $\mathbf{g}$. Minimizing this sum of squares is equivalent to maximizing the scalar (or dot) product of $\mathbf{t}$ and $R(\mathbf{g})$ (Horn, 1987).

$$\sum_{i=1}^{n} R(\mathbf{g_i})^T \cdot \mathbf{t_i} \tag{5.27}$$

5.3.3 Finding the Optimal Rotation with a Quaternion-Based Solution

Horn uses a technique based on Quaternions to determine the optimal rotation R. Using the compound product $\mathring{q}\mathbf{g}\mathring{q}^*$ with

$$\mathring{q} = w + iq_x + jq_y + kq_z; \;\; \mathring{q}^* = w - iq_x - jq_y - kq_z \tag{5.28}$$

that represents $R(\mathbf{g})$ utilizing Quaternions. The maximization of the above equation can be reformulated as follows:

$$\sum_{i=1}^{n} (\mathring{q}\mathbf{g_i}\mathring{q}^*)^T \cdot \mathbf{t_i} \tag{5.29}$$

We assume here that $\mathbf{g}$ and $\mathbf{t}$ have been centered on the origin of the coordinate system, i.e., $\mathbf{g}$ and $\mathbf{t}$ are differences of the original point sequences and their respective centroids $\overline{\mathbf{g}}$ and $\overline{\mathbf{t}}$. This means that the centroid of both $\mathbf{g}$ and $\mathbf{t}$ is the null vector. Horn shows that maximizing Equation 5.29 is equivalent to finding the unit Quaternion that maximizes

$$\mathring{q}^T N \mathring{q}$$

Here, N (see Equation 5.33is a matrix that is derived from the matrix sum M of the products of $\mathbf{t}$ and $\mathbf{g}$:

$$M = \sum_{i=1}^{n} \mathbf{t_i} \cdot \mathbf{g_i}^T \tag{5.30}$$

M, defined by its elements can be written as

$$M = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix} \tag{5.31}$$

where

$$S_{xx} = \sum_{i=1}^{n} \mathbf{g}_{x,i}\mathbf{t}_{x,i}, S_{xy} = \sum_{i=1}^{n} \mathbf{g}_{x,i}\mathbf{t}_{y,i} \tag{5.32}$$

and so forth.

The matrix N can then be constructed from the elements of M, so that:

$$N = \begin{bmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}-S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}-+S_{yx} & -S_{xx}+S_{yy}-S_{zz} & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & -S_{xx}-S_{yy}+S_{zz} \end{bmatrix} \tag{5.33}$$

Eigenvector Maximizes Matrix Product

$\mathring{q}^T N \mathring{q}$ is maximized by finding the eigenvector $\mathring{e}_m$ corresponding to the largest positive eigenvalue of N. By normalizing $\mathring{e}_m$, we obtain the quaternion $\mathring{q} = w + iq_x + jq_y + kq_z$, which encodes the optimal rotation angle $\theta = 2\cos^{-1}(w)$ and the unit representation of the corresponding rotation axis $(q_x, q_y, q_z)^T$. The optimal rotation matrix R is thus obtained from θ and the imaginary parts of $\mathring{q}$ as follows.

Let $\mathbf{s} = \sin(-\theta), \mathbf{c} = \cos(-\theta),$ then

$$R = \begin{bmatrix} q_x^2(1-q_x^2)\mathbf{c} & q_xq_y(1-\mathbf{c})-q_z\mathbf{s} & q_xq_z(1-\mathbf{c})+q_y\mathbf{s} \\ q_xq_y(1-\mathbf{c})+q_z\mathbf{s} & q_y^2(1-q_y^2)\mathbf{c} & q_yq_z(1-\mathbf{c})-q_x\mathbf{c} \\ q_xq_z(1-\mathbf{c}) & q_yq_z(1-\mathbf{c})+q_x\mathbf{s} & q_z^2+(1-q_z^2)\mathbf{c} \end{bmatrix} \tag{5.34}$$

5.3.4 Protractor3D Gesture Classifier

Using the optimal solution to the gesture – template rotation problem, we can now formulate the Protractor3D gesture classification algorithm.

5.3.4.1 Input and Output

We define

$$L = \bigcup G_i,\ i \in \mathbb{N}$$

as the set of all gesture classes (or gesture library), i.e. the different gesture movements that Protractor3D is trained to detect. Each G_i contains a number of training gestures $t_{G_i,k},\ k \in \mathbb{N}$. For a given input gesture g, Protractor3D will find the template $t_{G_x,y} \in L$ with the lowest Euclidean distance, corrected for rotation, with respect to g.

Protractor3D finds the gesture with the lowest distance to the input gesture g

5.3.4.2 Subsampling, Scaling, Centering

In order for gestures and templates to be comparable, we subsample[6] every input gesture to consist of a fixed number of points. We chose $n = 32$ as the number of subsampled points per gesture, as a higher n did not have a noticeable effect on the gesture recognition rate. We followed a subsampling strategy similar to Wobbrock et al. (?), adapted to work with 3D data (see also Section 5.2.2.4, Algorithm 1).

After subsampling, the input gesture is scaled to fit into a cube of a fixed side length, in our case $s = 100$. Finally, to be able to perform optimal alignment of the gesture data, we calculate the centroid of the input gesture and subtract it from all gesture points, thus centering the gesture. When creating gesture templates for the gesture library, user inputs are subsampled, scaled and centered in the same way before they are added to the gesture library.

[6]Subsampling is not the only strategy which can be applied here. The goal of subsampling is, in essence, only to obtain an equal number of sample points n with that of the template gestures. An alternative approach would be, for instance, to align the input gesture and template gestures with Dynamic Time Warping (DTW) (Sakoe and Chiba, 1978). N-Dimensional variations of DTW could also be used to index the best matching candidate template for a given gesture class, if multiple templates per gesture class are available in the gesture library (Keogh and Ratanamahatana, 2005).

5.3.4.3 Gesture Recognition

In the main part of the algorithm the gesture input, transformed as described in the previous section, is compared to every template in the gesture library, using rotational correction, to minimize the effects of device posture. The algorithm uses the Euclidean distance as the metric to compare the distance between the template and the input gesture. An advantage of Protractor3D is that it provides the angle of absolute rotation θ between an input and the template during comparison steps. Using θ helps Protractor3D decide if the template should be rejected as a possible recognition candidate. This does not need to be applied if complete rotation-invariance is desired.

gestures at extreme rotations are discarded

In our implementation, we have set the cutoff angle θ_{cut} to $\pm 45°$. If all gestures in the gesture library have been rejected due to θ being greater than θ_{cut}, Protractor3D recognizes the input gesture as "unrecognized." Otherwise, Protractor3D reports the gesture class of the template with the lowest Euclidean distance with respect to the input gesture as the recognized gesture class.

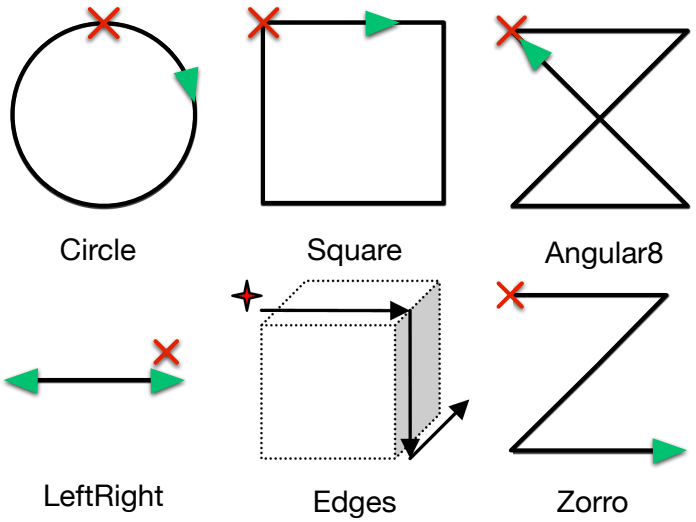


Figure 5.7: This set of iconic gestures is a subset of the gestures we used in the study. (Kratz and Rohs, 2011).

To evaluate the recognition performance of Protractor3D, we recorded gestures from ten paid test subjects. This approximate number of subjects has been also been used in previous studies on motion gestures (Liu et al., 2009b; Schlömer et al., 2008). The gesture set consisted of 11 gestures, six of which consisted of iconic gestures (see Figure 5.7), partially used in previous work by (Schlömer et al., 2008). The other half of the gestures was derived from the requirements of a related study on mobile gestures conducted in our group (Kirschnick

et al., 2010), which includes a user-defined gesture (*own*), everyday movements (*shakehand, shakearm*) and also some usual movements performed with mobile phones ("take out of *pocket*", "take out of *handbag*").

Each test subject recorded at least 40 repetitions of every gesture in the gesture set. A SHAKE SK6 (SAMH Engineering Services, 2009)[W] sensor package was used to capture the data at a rate of 100 Hz. The accuracy and sampling rate of the SHAKE SK6 acceleration sensor is comparable to those integrated into modern smartphones. To delimit individual gesture entries, the users were required to press and hold the navigation button of the SHAKE SK6 during gesture entry, and to release the button upon termination of the gesture entry. To determine the benefit of rotational correction that is delivered by Protractor3D, we measured the *Correct Recognition Rate* (CRR) of Protractor3D with rotational correction, as well as Protractor3D without rotational correction, which in the following we refer to as MSE (mean square error).

5.3.4.4 Training and Validation Set

40:60 split between training and validation sets

In order to evaluate the performance of Protractor3D, we defined *training* and *validation* sets for each of the gesture types on a per-user basis. We chose a 40:60 split between training and validation sets. When constructing the training and validation sets, we took only the first 40 gesture entries per gesture class, and discarded the rest (if present). Thus, for each gesture class the training set consisted of the first 16 gestures, and the remaining 24 were used at the validation set.

Protractor3D was trained on the last five gestures of the training set

As a consequence, we define CRR as the number of correctly recognized gestures divided by the size of the validation set. To "train" Protractor3D, we used the last five gestures of the training set for each gesture, for each user. We chose this low number of training samples to demonstrate the ability of Protractor3D to produce relatively robust gesture recognition results with only a few training samples. This property is advantageous for users seeking to rapidly add new gestures to their mobile device or developers who wish to rapidly prototype new gestures in a mobile user interface they are creating.

5.3.5 Recognition Results

Figure 5.8 shows the recognition results using Protractor3D with a training set size of 5 per gesture class. "Angular 8" was the gesture with the highest correct recognition rate of 98.9%, whereas "shakearm" has the lowest correct recognition rate of 83.3%. The average CRR over all gestures was 91% with a standard deviation of 4.5%. The reason for

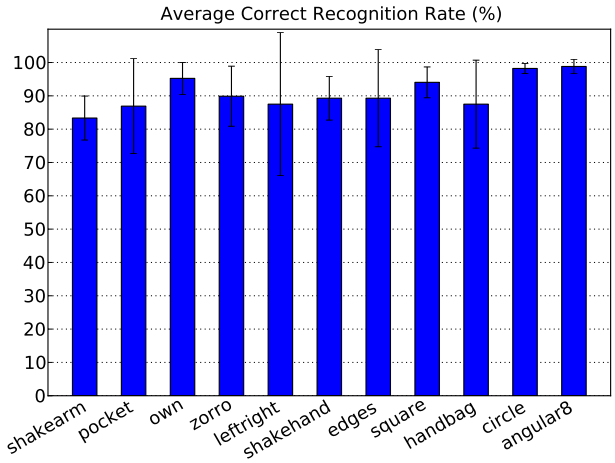


Figure 5.8: The average correct recognition (CRR) rates by gesture for Protractor3D, with a training set size of 5. The error bars show the 95% confidence intervals of the gesture CRR means. The global average CRR was 91%. (Kratz and Rohs, 2011).

these differences in gesture recognition rate is unclear, but it could be that the movements corresponding to the gestures with a lower correct recognition rate may have been more ambiguous to the test subjects than the gestures which achieved a higher gesture recognition rate. This ambiguity is also reflected, for instance, in the relatively large standard error in the average recognition of *leftright*, where we did not specify the exact way this movement was to be performed.

5.3.6 Effect of Rotational Correction

Protractor3D yields the same CRR for each rotation angle

To verify if Protractor3D has a noticeable correction effect on rotated input data, we repeated the evaluation of the algorithm with pre-rotated input gestures and compared the CRR of Protractor3D and (unrotated) mean square error (MSE) matching. We chose $(1,1,1)^T$ as the rotation axis and pre-rotated the input gestures by 0, 15, 30, 45, 90 and 180 Degrees (positive and negative). As can be seen in Figure 5.9, MSE is highly unstable under rotation, yielding very poor recognition results. In contrast, Protractor3D yields the same CRR for each rotation angle, thus showing that Protractor3D is rotation-invariant and truly finds the optimal rotation compensation for any given rotation of the input points.

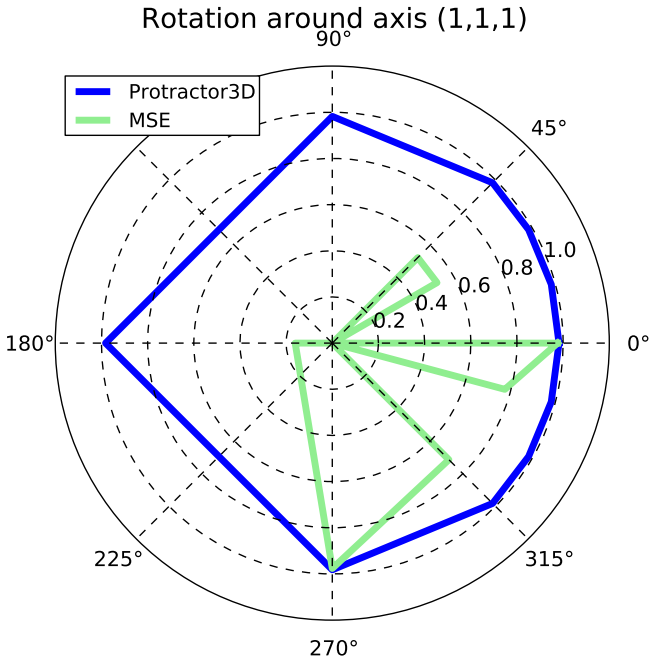


Figure 5.9: Polar plot of the influence of rotation on the correct recognition rate. Protractor3D has a constant CRR under all rotations and thus is rotation-invariant, whereas MSE-only is highly susceptible to changes in the rotation of the input gestures, and in some cases falls back to zero recognized gestures. (Kratz and Rohs, 2011).

5.3.7 Discussion

Protractor3D makes searching for the optimal rotation unnecessary

Protractor3D is a lightweight recognizer for motion-based 3D gestures that solves the problem of finding the optimal rotation between an input gesture and a template gesture by employing a closed-form solution. This solution to the template–gesture rotation problem is a major step over exhaustive search, since the latter requires finding the optimum in a three-dimensional search space. A brute force approach of checking all possible rotations for all templates in the gesture library is thus out of the question, in particular for resource-constrained mobile devices.

Protractor3D requires only a low number of training gestures but still maintains an acceptable recognition rate

Protractor3D applies a closed-form solution to finding the optimal rotation based on a quaternion representation originally proposed by Horn (Horn, 1987). This approach leads to an efficient and accurate gesture recognizer that can be implemented easily. Protractor 3D requires only a low number of training gestures and thus imposes a low overhead in design and prototyping phases of gesture-based interfaces, in which designers (or later also users) wish to come up with their own custom gestures. The design and personalization of gesture-based interfaces thus becomes more approachable by designers as well as users.

5.4 Gesture Based Authentication

At present, mobile devices are used almost ubiquitously for a growing number of tasks and access to online services. An increasing amount of sensitive data is accessed, stored and transferred through mobile devices, exposing mobile device users to potential attacks.

To prevent misuse such as copying or modifying stored data, it is important that only the genuine user is allowed to access security-sensitive features of the device. This protection can be achieved by employing a mechanism for user authentication, in order to verify the identity of the current user.

From the user's perspective, any authentication mechanism is an obstacle, as the goal of security can be described as "making undesirable actions more difficult" (Kainda et al., 2010). Users often regard authentication mechanisms as annoying and avoidable obstacles as authentication does not support them in fulfilling their tasks. To be successful an authentication mechanism needs to be secure as well as usable. As the interaction with mobile devices is usually short and happens very frequently (Falaki et al., 2010), it is necessary that a mechanism must be not annoying and fit in the context of the interaction. The usability of an authentication mechanism can either "make or break system security" (Cranor and Garfinkel, 2005).

In the following we[7] present a behavioral biometric authentication mechanism for handheld mobile devices based on personalized motion gestures. The device measures the user movements using a built-in 3D accelerometer and 3D gyroscope. We performed a user study evaluating several attack scenarios. Our results show that it is technically possible to differentiate between persons using gesture data obtained from accelerometers and gyroscopes. Furthermore, we found several usability advantages in comparison to existing authentication mechanisms.

5.4.1 User Authentication Mechanisms for Mobile Devices

Most authentication mechanisms for mobile devices are based upon shared secret between the device and the genuine user. Common examples are passwords and personal identification numbers (PINs). The concept of a shared secret is easy to implement and well understood by users. However, such mechanisms have certain limitations on mobile

[7]The implementation and user study was conducted by Dennis Guse in is Master's Thesis (Guse, 2011).

devices because of their limited text input capability. Entering a complex password correctly on small physical or even virtual keyboard is demanding task. PIN entry usually demands the user's visual attention, which in many mobile situations might not be possible or impractical. An inherited problem of knowledge-based mechanisms is that the genuine user needs to memorize a complex secret that can be recalled easily but that isn't guessable easily by attackers. The memorization of passwords or PINs is thus not a trivial task.

Biometric authentication mechanisms are a promising alternative as performing authentication this way is usually less demanding mentally, because one or more inherent physiological or behavioral features of the user is used for authentication. Physiological features use static measurable properties of the user's body, such as finger prints. Behavioral features are usually more complicated to verify than physiological ones, as they depend on the actual context of the user, i.e., her inputs during interaction.

Common examples of behavioral features are keystroke dynamics and written signatures. Generally, user authentication using biometric features is more complicated in comparison to knowledge-based mechanisms, because measuring the properties of humans is complicated. The sensors used for measurement are prone to noise and more importantly, biometric features may vary over time. For this reason, biometric mechanisms need to be robust under a certain variance. However, they do remain beneficial in many situations.

5.4.2 Gesture-Based Authentication Mechanism

We measure gesture input data from a 3-axis accelerometer as well as a 3-axis gyroscope. Using a gyroscope adds additional features to the data and can measure rotations that cannot be derived from analysis of accelerometer data, for instance when the accelerometer is oriented in the horizontal and a rotation is performed in the vertical axis—in this case the accelerometer can only measure minimal acceleration, thus no valid rotation data can be obtained.

We chose to use personalized motion gestures for authentication as this does not require the users to learn a gesture from some arbitrary set but rather use their own movement preferences. We also believe that this makes reproducibility easier and also makes it more difficult for gestures to be forged. To delimit the beginning and the end of a gesture during input, we used a *push-to-gesture* button, because this is the simplest approach and we didn't want an unpredictable automatic

segmentation technique to confound our results. Indeed, the interaction with the push-to-gesture button becomes part of the gesture through small accelerations registered while the user is pushing the button, making attacks more difficult.

The authentication mechanism uses the raw sensor readings from both sensors. The mechanism determines a similarity score for sensor readings representing an input gesture in comparison to a set of enrollment samples and then uses a threshold to determine if the input gesture is a valid authentication. An input gesture is accepted as valid if its similarity score lies below the defined threshold.

We evaluated several variants of Dynamic Time Warping (DTW) and Hidden Markov Models (HMM) for calculating the similarity score. These machine learning techniques are well suited to classifying sequential information and are able to cope with variations in timing, which is typical for gesture entry.

We implemented our variants of based on the formulation by (Sakoe and Chiba, 1978), using the weighted Euclidean norm (Redzic et al., 2010), to compensate for differences in scale between accelerometer and gyroscope data. A DTW model for an input gesture is consists of one data sequence. This can either be one of the enrollment (training) samples, usually the one with the smallest distance to the input gesture, or the sequence best modeling the input gesture can be extracted from the entire enrollment set using Crosswords Reference Templates (CWRT), which leads to significantly improves the recognition rate (Abdulla et al., 2003).

% As an alternative to DTW, we studied first-order HMMs with a multivariate Gaussian emission distribution (Rabiner, 1990) using the normalized likelihood as a metric for similarity. The HMM's variation in length was constrained, in order to limit the maximum allowed difference in length of unknown inputs in comparison to the mean length of the enrollment samples.

5.4.3 Attack Types

Authentication mechanisms are only useful if they achieve a certain security performance by being able to resist attacks. An attack is deemed successful if an attacker is able to forge the gesture of a genuine user.

We distinguish between three types of forgeries depending on the knowledge available to an attacker about the genuine gesture:

- **Naïve:** an attacker that has no knowledge about the genuine gesture can only create *naïve forgeries* (Ballard et al., 2007).
- **Semi-Naïve:** if an attacker has knowledge about the general shape if the genuine gesture, he can perform *semi-naïve forgeries*.
- **Visual:** The most sophisticated class of attacks is based upon visual disclosure of the genuine gesture (Liu et al., 2009a). We call these *visual forgeries*.

It is clear that the more knowledge about the genuine gesture is available to an attacker, the more likely it is that he can produce successful forgeries. However the attacker must also be able to physically reproduce the gesture with enough precision. The difficulty for the attacker thus depends on the complexity of the genuine gesture as well as the ability of the genuine user to reproduce the gesture with a variance that will pass the threshold set by our mechanism.

5.4.4 User Study

We conducted two user studies to evaluate three key aspects relevant to gesture-based user authentication on mobile devices: *feasibility*, *usability* and *resistance* against attacks.

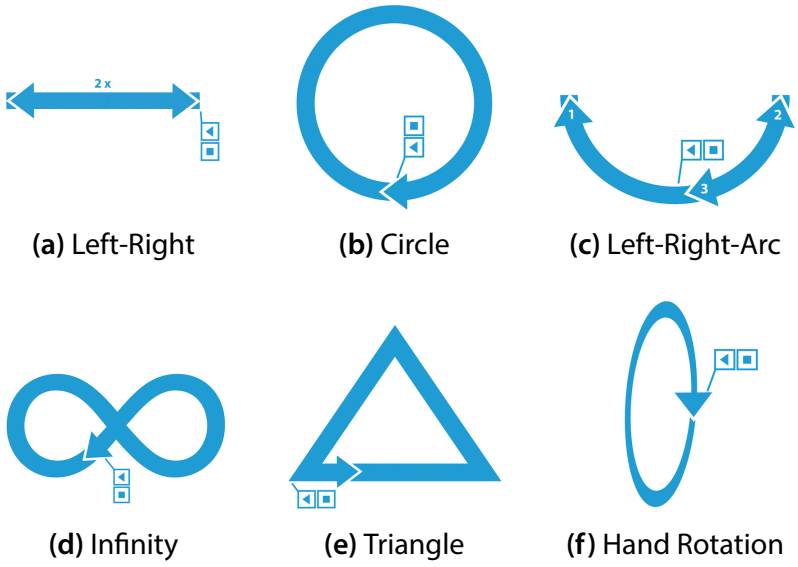


Figure 5.10: Visualization of the gestures we designed for use in the first user study.

Feasibility addresses the question whether gesture-based authentication (GBA) is in general possible using motion sensors embedded in

mobile devices. For the system to be usable, it needs to be perceived by users as a useful alternative or even a replacement for currently available authentication mechanisms.

In the first study 15 participants simulated the perspective of genuine users, so that we could study the feasibility and usability aspects of our system. We predefined six gestures for use with our system (Figure 5.10). This way, we avoided burdening the participants with inventing their own gestures, giving them more time to understand the authentication mechanism itself. Also, this allowed us to evaluate semi-naïve forgeries. The gesture labels helped the test subjects to attach meaning to the gestures, such that they could memorize them more easily.

Each participant provided 5 enrollment and 15 validation samples for each predefined gesture class. The enrollment samples were used to build a model for each gesture class and the validation samples were used to test the model's accuracy. The enrollment samples and 10 validation were recorded while the user was standing. Video recordings of the first user study were used in the second user study to evaluate the risks due to visual disclosing the genuine gesture. We selected two interpretations of each designed gesture and showed the video recordings of the enrollment samples to the so-called forgers. In this study participated 10 persons that not participated in the first user study, who did not know the visualization and description of the gestures.

We recorded the gesture entries during the first user study so that we could use them in the second user study to evaluate the risks due to visual disclosure of the genuine gesture. We selected two interpretations of each designed gesture and showed the video recordings of the enrollment samples to each participant ("forger") of the second study. In total, 10 subjects participated in the second user study. None of those test subjects had participated in the first study, and the visualization and description of the gestures was unknown to them.

We developed two questionnaires to evaluate the usability as well as the social acceptability of GBA. For the user study, we implemented an application for an iPhone 4, which displays the push-to-display button and logs motion sensor data at a frequency of 80 Hz.

5.4.5 Results

The results of the user study are promising. We found that a length constraint of $\pm 23\%$ around the average sequence length of the enrollment gestures performs well. This constraint includes 97.3% of the enrollment and 90.7% of the validation samples. It excludes 58.3% of

the naïve and semi-naïve forgeries and 36.9% of the visual forgeries. For DTW, we obtained the best results with a slope constraint of 1 and a non-diagonal alignment penalty. The integration of multiple samples for matching performed better than using only the enrollment sample with the lowest distance.

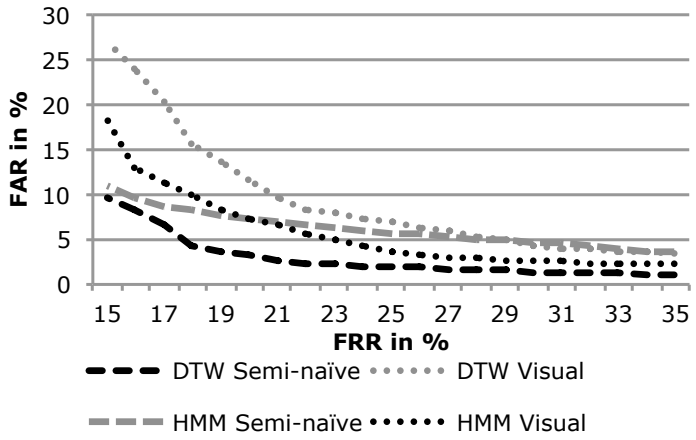


Figure 5.11: ROC for the 12 interpretations attacked in the 2nd user study. The x axis shows the false rejection rate (FRR) and the y axis shows the false acceptance rate (FAR). (Guse, 2011).

As expected, naïve forgeries were rarely accepted as genuine by both algorithms. In Figure 5.11, the Receiver Operating Characteristics (ROC) of HMM and DTW for semi-naïve and visual forgeries are compared with the attacked interpretations. DTW performs as expected, because visual forgeries are more likely to be accepted than semi-naïve ones.

We obtained an unexpected result for HMMs. Above a certain threshold, HMMs accept a larger number of semi-naïve forgeries than visual forgeries. A possible cause for this is the use of likelihood as similarity metric. In general, HMMs perform better for visual forgeries than all variants of DTW we studied. In a detailed evaluation of DTW we found that 5 of the 12 attacked interpretations achieved a False-Rejection-Rate (FRR) of less than 20% without accepting any of the visual forgeries. With one model, we achieved a FRR of 0% and a single model was completely unusable with a FRR of 93%.

According to the evaluation of the questionnaires, none of the participants of the first user study perceived the mechanism as unnatural, annoying or fatiguing. 2/3 of them would use gestures for authentication in public places. Nevertheless, 7 participants believed that a gesture is easily forgeable. The majority of forgers believed that they can create an exact forgery for 9 of the attacked gestures. However, as we demonstrated with the second user study, this is not true.

5.4.6 Discussion

Through our user study, we have shown that GBA is *feasible*—an identification based upon biometric input measurements of a user's movement can be accomplished using the accelerometers and gyroscopes embedded in mobile devices.

Our results indicate that GBA is *usable*. It was not perceived negatively or as annoying or tiring by any participant. The only issue was that gestural authentication may not be suitable in certain social contexts. Possible solutions for this are to retain the existing (and socially non–critical) authentication mechanisms alongside GBA, or to extend the GBA mechanisms with implicit authentication mechanisms. There has already been work in this domain, i.e.(Jakobsson et al., 2009).

We have shown that GBA has the potential to be *secure*. Although, as shown in Figure 5.11, the FRR for a "reasonable " FAR still remains high, with a FRR $> 20\%$ for a FAR $< 5\%$, increasing the number of training templates or applying a more sophisticated learning algorithm than DTW should significantly improve these values. The number of training templates should be trivial to increase, as the user supplies a new template upon every successful authentication. Of course, DTW will only scale performance-wise up to a certain number of templates. Thus, templates with lower similarity will have be discarded over time. This scheme would also help the system adapt to changes in the users' movement characteristics over time. Alternatively, a more sophisticated supervised learning algorithm such as an Artificial Neural Network. Every successful authentication could then be used for an additional optimization step of the model. Even though it remains to be seen if the security level of PIN entry can be achieved with GBA, the gesture-based technique is a promising candidate for authentication tasks that need to be used repeatedly in low- to medium-risk scenarios, such as device unlocking or accessing highly frequented services with a relatively low damage potential, such as chat applications or social networks[8].

Finally, what is interesting to note is that the *perception* of the security of our system was lower than its actual performance, as most of the forgers believed they could successfully forge legitimate entries. A study may need to be made with professional movement practitioners, such as mimes or dancers, to see if the security of the system lies mainly in the ineptitude of the attackers to properly mimic the movements of the legitimate users or if the biometric movement characteristics that

[8]It is likely that individual users will have differing views on what the risk level of a particular task is. This risk-assessment is given solely for exemplification and represents the author's opinion on the stated scenarios.

we measure through acceleration and rotation are truly unique. Future work may also be directed at identifying the exact biometric invariants that are measured through GBA.

5.5 Combining Accelerometer and Gyroscope Data for Motion Gesture Input on Mobile Devices

A growing number of smart phones are being equipped with 3-axis gyroscopes in addition to 3-axis acceleration sensors. Combining the data from these two sensor types provides significantly more motion information compared with using only an accelerometer. This increase in motion information can be used for more precise recognition of motion gestures, allowing more complex gesture types to be used in mobile user interfaces, since the users can also add rotational components to their motion gesture inputs. Motion gestures can be used for a variety of applications on smart phones. These include gaming interfaces, where, for instance, the user interacts via a spell casting metaphor (Ballagas et al., 2007), entering special modes (Ruiz and Li, 2011), navigation tasks (Ruiz et al., 2011) or user authentication (Guse, 2011).

On devices without a gyroscope, rotation can be approximated using accelerometers alone by using the direction of gravity as a reference for tilt. However, this approximation is not reliable as in certain cases, i.e. when the device is rotated in the plane perpendicular to gravity, no tilt and thus rotation information can be obtained.

We cannot assume that, in general, developers of mobile applications have profound knowledge of machine learning algorithms, let alone the skills to implement them efficiently. However, it is desirable to have gesture recognizers for motion gestures on mobile devices that can be used early in development of an application. This requires such algorithms to be easy to implement and to tune. Template-based techniques such as nearest-neighbor search are generally easy to implement. However, naïve template-based techniques will generally not compensate for variations in gesture execution time.

We cannot assume that, in general, developers of mobile applications have profound knowledge of machine learning algorithms, let alone the skills to implement them efficiently. However, it is desirable to have gesture recognizers for motion gestures on mobile devices that can be used early in the development of an application. This requires such algorithms to be easy to implement and to tune. Template-based techniques such as nearest-neighbor search are generally easy to implement. However, naïve template-based techniques will generally not

compensate for variations in gesture execution time. A popular algorithm that does compensate for differences in time series is Dynamic Time Warping (DTW) (Sakoe and Chiba, 1978). Protractor3D (Section 5.3) is a template-based technique developed for 3D acceleration data that compensates for rotational derivations between input sequences and templates by finding the optimal registration between input points and templates, in a way similar to Protractor (Li, 2010b), which only works with 2D data. A problem with Protractor3D is that this algorithm does not work with data of dimensions higher than 3.

In the following, we present a consensus-based approach using two instances of a template-based gesture recognizer (such as Protractor3D or DTW) to perform motion gesture recognition on the 6-D data obtained from an accelerometer-gyroscope pair. By analyzing a large corpus of motion gesture entries by users, we show that the combination of accelerometer and gyroscope data increases the gesture recognition rate by up to 4% on our data set. We show that combining acceleration data with rotation data from a 3-axis gyroscope will improve the gesture recognition rates for mobile motion gestures, and we give recommendations on the choice of a gesture recognizer dependent on the nature of the data set and the available computational resources.

5.5.1 Extending Protractor3D with Gyroscope Data

By design, Protractor3D cannot use data of a higher dimensionality than 3. This is inherent in the mathematics it uses to calculate the optimal registration between input points and templates. A simple extension to add support for gyroscope data in addition to accelerometer data is to run a second instance of Protractor3D in parallel on the gyroscope data. The remaining challenge is then to reconcile the two recognition results in order to determine which gesture has been recognized.

We propose a weighted approach that depends on the order of similarity of comparisons between input gestures and stored templates, i.e. templates that are less similar to the current input have a lower influence on the final recognition result. Our results indicate that the match with the lowest distance (or highest similarity) is most probably the gesture recognized. In other words, comparisons with a lower similarity are less likely to contribute to the decision on the recognition result.

Algorithm 3 shows pseudocode for the data combination algorithm (DCA) we devised. The DCA can be generalized easily to combine the results for more than two sensor data types.

The only parameter of the algorithm that needs to be chosen specifically by the developer is *bias*. The optimal value for *bias* needs to be determined by analyzing existing user inputs. Figure 5.12 shows the influence of *bias* on *Precision* and *Recall*[9] results for our data set. In our case, the optimal value of *bias* lies between 1.4 and 2.4.

bias is the only parameter that needs to be set by the developer

Algorithm 3 ***Data Combination Algorithm (DCA) :***

Inputs	*accResults*	acceleration comparison results (list of gesture IDs sorted in descending order by similarity to the input)
	gyrResults	acceleration comparison results (list of gesture IDs sorted in descending order by similarity to the input)
	N	number of items in each of the comparison lists
	bias	bias value determining influence of weaker comparisons
	nGestureIDs	number of gesture classes (each class has a distinct ID)
Output	*bestGestureID*	the ID of the best matching gesture

```
counter = float[nGestureIDs]
for i=0 to N-1 do
  accRecogID = accRresults[i]
  gyrRecogID = gyrResults[i]
  counter[accRecogID] += 1.0/(bias*i +1.0)
  counter[gyrRecogID] += 1.0/(bias*i +1.0)
end for
bestGestureID = argmax(counter)
return bestGestureID
```

5.5.2 Analysis of Combining Accelerometer and Gyroscope Data

To analyze the effects of combining 3-axis acceleration data with 3-axis rotation data for recognition by motion gesture recognizers, we present gesture recognition results for the following recognition algorithms: *Protractor3D*, *DTW* and *Regularized Logistic Regression (LR)* (Lee et al., 2006). We chose DTW due to its popularity. We chose LR because it is, in contrast to DTW and Protractor3D, a feature-based supervised learning algorithm that performs well with higher-dimensional data and it

[9] See Section 5.5.2.2, Page 165, for a definition of *Precision* and *Recall*.

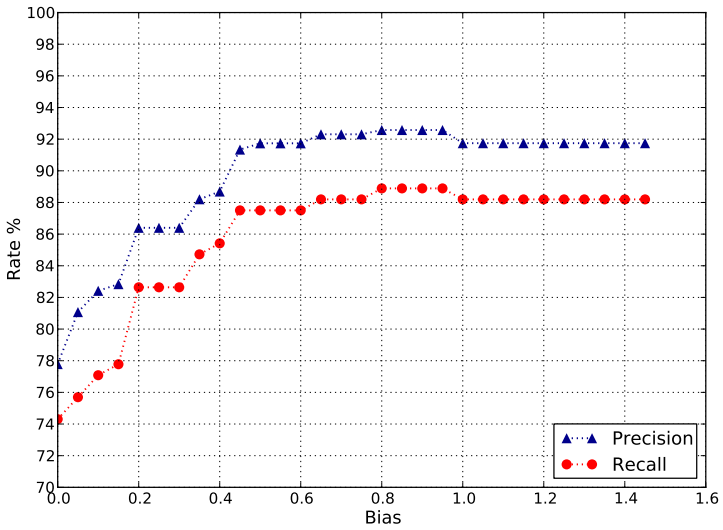


Figure 5.12: The plot shows the influence of the bias variable in the weighted reconciliation algorithm. In this case $N = 5$. Choosing a $bias > 0.8$ does not improve the gesture recognition rate further, while giving each result an equal "vote" ($bias = 0$) generally results in a lower gesture recognition rate.

is still fairly easy to implement. LR is very efficient, since classification basically consists of multiplying a weight vector θ of a fixed length N (where N corresponds to the amount of features) with the features of the gesture input. Thus, compared with Protractor3D and DTW, the execution time of LR is independent of the number of training samples used.

5.5.2.1 Gesture Data Set

The gesture data for our analysis is the same that we used for the user authentication study in Section 5.4. However we used on only the gestures entered by the 15 users of the first part of the study (Section 5.4.4).

For all further analysis, we use the first 5 entries as training templates and the last 10 entries for validation. In total, we recorded $15 \times 15 \times 6 = 1350$ gesture entries. 20 gesture entries had to be discarded, because they had too few samples due to erroneous entry. Our final data set size comprises 1330 gestures. Each gesture entry contains an average of 127 time-based data samples ($\sigma = 38.4$).

The hardware we used for recording was an iPhone 4 running a custom-built application that logged the gesture data and implemented the push-to-gesture button. We recorded the following data from the

iPhone 4's accelerometer[10] and gyroscope[11] at a frequency of 80 Hz: *acceleration*, *rotation rate* and *attitude*[12]. All acceleration values were measured in g, all rotation was measured in radians/second and attitude was given in Euler angles.

5.5.2.2 Recognition Results

We evaluated Protractor3D, DTW and LR for motion gesture recognition on the data set. As input to the algorithms, we used the following data and combinations thereof: *acc* (acceleration), *rot* (rotation), *att* (attitude), *acc-rot* (acceleration + rotation), *acc-att* (acceleration+attitude).

We configured Protractor3D to subsample and normalize the gesture entries to contain 64 samples. In order to keep the length of the feature representation for each gesture constant, we used the subsampled data generated by Protractor3D as input for LR. Each gesture was thus represented as a feature vector with a length of $64 \times 3 = 192$ for *acc*, *rot* and *att* and $64 \times 6 = 384$ for *acc-rot* and *acc-att*. DTW used the unmodified sensor samples as input, as this algorithm is specifically designed to cope with length differences between inputs and templates. Our DTW implementation used the Euclidean (L^2) Norm as the distance function. We did not apply any step constraints for calculating the distance matrix.

Classifier Performance Metrics As quantitative metrics for classifier performance, we use the *Precision* (P), *Recall* (R) and the F_1 Score (F_1). Given the amount of true positives tp, the amount of false positives fp and the amount of false negatives fn.

Precision is defined as:

$$P = tp/(tp + fp) \tag{5.35}$$

i.e., the ratio of correct gesture recognitions out of all generated predictions.

Recall is defined as:

$$R = tp/(tp + fn) \tag{5.36}$$

[10] STMicroelectronics STM331DLH 3-axis MEMS accelerometer

[11] STMicroelectronics L3G4200D (equiv.) 3-axis MEMS gyroscope

[12] *attitude* represents the absolute change of rotation with respect to an initial reference frame, i.e. the device attitude at the beginning of a gesture recording.

i.e., the ratio of correct gesture recognition out of all gestures in the data set.

F_1 is defined as:

$$F_1 = 2\frac{PR}{P+R} \tag{5.37}$$

which represents a combined metric for the accuracy of a classifier and is based on the harmonic mean. The F_1 Score ranges from 0 to 1, where 1 is the best score obtainable.

Figure 5.13 shows the results for Precision (P), Recall (R) and F1-Score ($F1$) for the classifiers Protractor3D, DTW, and LR for each of the combinations of data types *acc* (acceleration), *rot* (rotation), *att* (attitude), *acc-rot* (acceleration + rotation), *acc-att* (acceleration+attitude). The combined data was evaluated using the DCA. In addition, we evaluated DTW and LR without the DCA by using 6D feature vectors obtained by merging each 3D acceleration sample with its rotation counterpart. DTW with *acc-rot* and the DCA achieved the best result with an F_1 Score of 0.95, followed by Protractor3D using the same settings with an F_1 Score of 0.92. The best F_1 Score for LR was 0.86 using combined accelerometer and gyroscope data without the DCA. The lower score for LR suggests that a larger amount of training samples may be needed in order to obtain better results for this algorithm.

Algorithm Execution Times For motion gesture recognizers, gesture recognition performance is not the only important criterion. The execution time is, arguably, equally important, especially on mobile platforms. The runtime of template-based approaches Protractor3D and DTW directly depends on the number of training templates, whereas this does not affect LR.

To exemplify this for our data set, Protractor3D and DTW have to compare each of the 450 template gestures with 900 gesture entries, resulting in a total of 405,000 necessary comparisons for a single pass over the data set. For each comparison, DTW needs to construct a distance matrix with a size of $n \times m$, where n is the number of elements in the template and m of the sample, including evaluation of the distance function. Protractor3D does not have as much overhead as DTW, but still needs to complete the same number of comparison operations.

By contrast, LR requires just 900 matrix multiplications to perform all predictions. However, in comparison to DTW and Protractor3D, feature-based classifiers such as LR need a large number of training

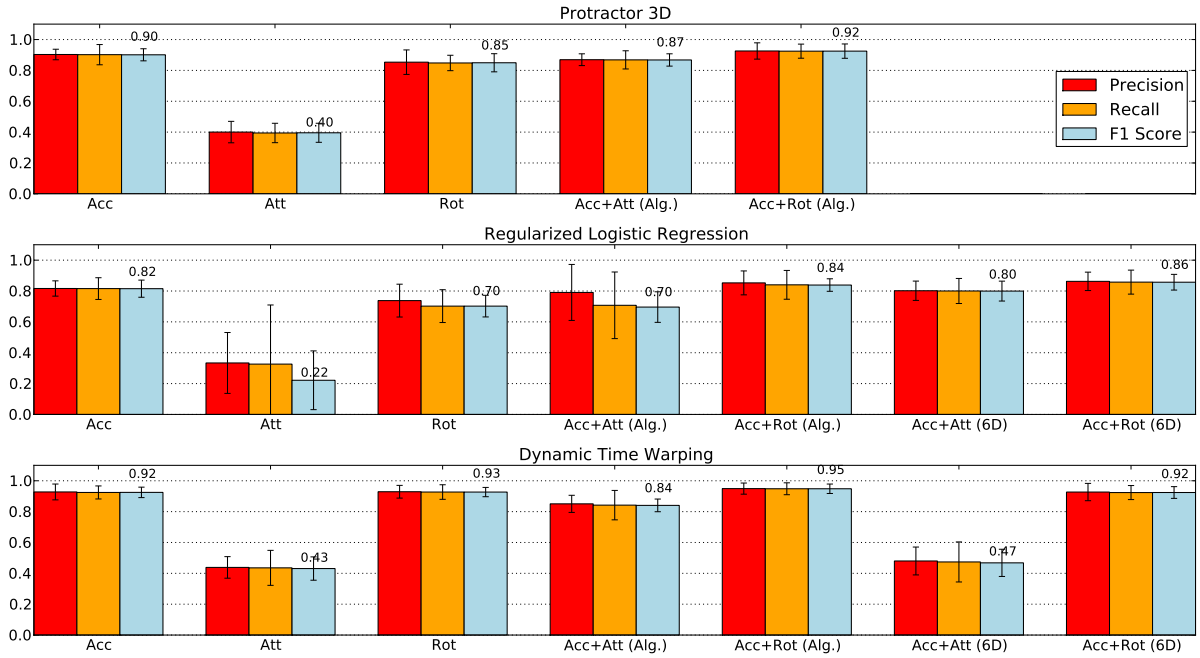


Figure 5.13: Mean recognition results for Protractor3D, Regularized Logistic Regression and DTW, for the data types accleration (*acc*), attitude (*att*), rotation (*rot*), acceleration+attitude *acc+att*), acceleration+rotation (*acc+rot*). "Alg." marks results the using the DCA. "6D" marks results using a 6D merged representation of accelerometer combined with gyroscope data. The error bars show the standard deviation. DTW with *acc+rot* using the combination algorithm achieved the best result with an F_1 Score of 0.95.

Algorithm	Approx. Number of required comparisons	Profiled Cumulative Execution Time (s)	Average Processing Time per Gesture (s)
LR	900	0.4	0.004
Protractor3D	405,000	881	0.98
DTW	405,000	3966	4.4

Table 5.1: The approximate number of operations and the execution time (including data loading and pre-processing) required by the analyzed algorithms for a run on our data set. The average processing time per gesture was calculated by dividing the total execution time by the number of test samples in the training set. DTW (without additional optimizations, i.e., (Rakthanmanon et al., 2012)) is by far the slowest algorithm as it needs to construct an $O(n^2)$ distance matrix in each comparison step. LR has the lowest measured execution time as performs only a single operation for each gesture in the validation set.

samples to work optimally, so developers face a trade-off between the time requested from users to enter training samples and the computational efficiency of the gesture recognizer.

Table 5.1 shows the measured execution times for each algorithm when running on our data set. We implemented all gesture recognition algorithms in Python 2.7.1 using the *NumPy*, *SciPy* and *rpy2* libraries. All computation was performed on a Mac Pro with 2.66GHz Intel Core i7 processors. In order to measure the classifiers' execution time, we used Python's *cProfile* library. All classifiers were run exclusively as single-threaded applications and did not make use of parallel processing.

The comparison in Table 5.1 shows that the average per-gesture execution time of LR is two orders of magnitude lower than Protractor3D and three orders of magnitude lower than DTW. The runtime advantage of LR is thus very clear but the problem remains that this algorithm requires an optimization library in order to train models. Thus, the most beneficial use of LR would be in deployment-stage applications, with appropriate models for gesture recognition already trained, since the classification step does not require an optimization library. Protractor3D compares favorably with DTW, requiring about 4s less computation time. The measured processing time per gesture could be further improved by decreasing the subsampling size and switching from Python to a lower-level language such as C.

5.5.3 Discussion

Confirming results obtained in previous work (Hoffman and Varcholik, 2010), our results indicate that combining accelerometer with gyroscope data leads to improvement in gesture recognition rates. The is reflected by the 2%, 3%, and 4% increases in F_1 score for Protractor3D, DTW and LR, respectively. Our results also indicate that the DCA we developed seems to be more beneficial for template-based approaches and not so much for feature-based classifiers such as LR.

We recommend that developers choose their gesture recognition algorithm by (1) the number of training samples available (template-based approaches perform well, even with few training samples), (2) the performance constraints of their target devices (we obtained the highest F_1 score when using DTW, but it was also the algorithm with the highest execution time by far) and (3) the time available for coding during each iteration (template-based methods are easier to implement and deploy on different types of devices, since they do not require specialized libraries).

5.6 Summary

First, we presented two motion gesture recognizers for accelerometer data, the $3 Gesture Recognizer and Protractor3D. User studies we

conducted show that they have an acceptable gesture recognition rate. The $3 Gesture Recognizer achieved an average correct recognition rate of 80%. Protractor3D improved upon $3 by adding invariance to rotational differences between gestures and template. The average correct recognition rate for Protractor3D was 91%. These algorithms are easy to implement and do not require sophisticated toolkits, thus that they are easy to deploy in a rapid-prototyping scenario.

Second, we presented the results of a user study on gesture-based authentication (GBA) for mobile devices using acceleration and rotation data . Our results indicate that GBA is feasible, usable and reasonably secure, with a high potential for improvement.

Last, we presented a detailed analysis on the effects of combining accelerometer and gyroscope data for motion gesture recognition. Recognizers, such as Protractor3D, that do not support data dimensions greater than 3, can be extended using an algorithm we developed. The results we obtained showed that adding gyroscope data will always improve gesture recognition. Concretely, the gesture recognition F_1 Score improved by 2%–4%, depending on the algorithm we tested.

Chapter 6

Enabling End-User Programming of Sensor-Based Interaction

"Beauty and brains, pleasure and usability—they should go hand in hand."

—Don Norman

In this section we will discuss two software tools for use with sensor-based input. For user interfaces based gesture input in particular, obtaining useful gesture recognition is only a necessary prerequisite for controlling further functionality. Since the gesture recognition algorithms developed in Chapter 5 return only the symbolic value of the recognized gesture, it is desirable to be able to quickly attach meaning to recognized gestures, e.g. by triggering actions such as changing to the next PowerPoint slide or causing a screen shot to be saved. It would thus be cumbersome if every practitioner or researcher had to implement the required functionality associated with recognized gestures from scratch.

Mayhem can be used to associate meaningful functionality with gestural input

This issue can be addressed with *Mayhem*, a software utility developed in conjunction with the Microsoft Applied Sciences Group[1] (Microsoft Inc., 2012c)[W]. Mayhem allows the creation of simple scripting tasks in Windows. Mayhem is a scripting environment for personal and home automation. Mayhem can be adapted easily to use mobile gestures as input or, in general, any kind of data from sensors located on mobile devices. Thus, scripts can be built to allow mobile device users to control the multitude of output modules provided by Mayhem. Mayhem is

[1]Mayhem was developed in collaboration with Paul Dietz, Jay Meistrich and Eli White.

based on an extremely simple programming paradigm and is targeted mainly at end users with no knowledge of programming.

Mayhem can be used by researchers and practitioners to rapidly build applications with gesture-based input. This is useful, e.g. for rapid prototyping cycles as well as building output functionality for use during user studies. Mayhem's system architecture is described in Section 6.1, and an example usage case is presented in Section 6.2.

In the remainder of this chapter we will discuss Mayhem in more detail and present a usage scenario.

6.1 Mayhem: a Scripting Environment for End Users

In this section, we will present a brief overview of the Mayhem end user programming environment and present a usage scenario that exemplifies the use of Mayhem as a development too for mobile sensor-based and gesture-based interaction.

6.1.1 Introduction and Related Work

the possibilities for automation in modern computing aren't easily accessible by end users

Ever since their invention, computers have been used extensively for control and automation tasks. Devices for control and automation are usually found in special locations, for instance as an engine control unit (ECU) embedded in motor cars, or in production environments, such as factories or chemical plants, amongst others. With the advent of always-on internet connections, integrated web cameras on a growing number of laptop computers, portable music players, smart phones and tablet PCs, the proliferation of social networking services, and the increasing popularity of internet-enabled set-top-boxes, a wide range of new opportunities for personal automation applications has appeared.

Home and personal automation, however, is presently far from widespread, although there are many cases in which such systems would be beneficial, for instance in order to regulate the efficient use of heating or to automate the sending of notifications of important, context-based events to friends or coworkers. Any type of automation or scripting usually requires the developer to have programming skills. Mayhem is, by contrast, a development environment that does not require the user to have programming skills, but still allows users to create complex functionality with a simple programming paradigm. Seen in the scope of this thesis, Mayhem is an ideal tool to connect the output of gesture recognizers or mobile sensors with meaningful

functionality. Using Mayhem, developers, scientists or practitioners can test new user interfaces and interaction styles rapidly and take advantage of the functionality provided by Mayhem's reaction modules easily.

Mayhem runs on any Windows PC with Windows 7 or later. Home PCs have become a very powerful and ubiquitous platform for automation tasks. PCs can communicate with and control a vast number of peripherals using USB, FireWire (IEEE 1394), (wireless) Ethernet, Bluetooth or, in some cases, legacy interfaces such as RS-232 or the parallel port (IEEE 1284). However, PCs, being present in 63.5 percent of households in the United States (National Economics And Statistics Administration and National Telecommunications and Information Administration, 2011), are, in our opinion, underutilized for personal automation tasks. Mayhem provides a very simple user interface that allows end-users to implement simple but compelling scripting tasks for Windows and home automation. By using prebuilt but configurable events and reactions, and using a single *Event-Reaction*-based connection semantic, Mayhem abstracts away from complex hardware interfaces and algorithms, making the implementation of automation tasks an exceptionally easy task for end users.

A wide body of previous work related to Mayhem exists. A multitude of visual programming environments (VPEs), for instance Max/MSP (Cycling74, 2012)[W], pd (Puredata, 2012)[W], Quartz Composer (Apple Computer Inc., 2012d)[W], PopFly (Microsoft Inc., 2012d)[W] or Kudu (Microsoft Inc., 2012b)[W] or App Inventor (Massachusetts Institute of Technology, 2012)[W] have been developed. Often, these environments are suitable for solving specific tasks, such as graphics output (Quartz Composer), rapid application development (Kodu, App Inventor), mashup creation (PopFly) or to be used by specific user groups, i.e. musicians or visual artists (Max/MSP and pd) or specialists in industry as in (Prähofer et al., 2008). Visual programming languages are not limited to desktop environments. Reactable (Jordà et al., 2007), for example, is a tangible and visual programming interface for music. Recent work on visual programming environments for mobile devices has also been conducted, for instance the urMus (Essl, 2010) project, which includes a visual audio synthesizer patch builder on iPhone and iPad.

Operating system wide scripting and automation services as well as macro recorders are related to Mayhem in functionality. Examples are AppleScript (Apple Computer Inc., 2012b)[W] or the Mac OS Automator (Apple Computer Inc., 2012c)[W]. In contrast to Mayhem, AppleScript requires some basic programming knowledge. Mac OS Automator has a strong focus on scripting application functions unique to Mac OS whereas Mayhem is more generic in scope.

Mayhem was developed because existing systems are still too complex for unskilled users

Although several of the existing systems could have been extended to support some of the functionality present in Mayhem, creating a completely new system allowed us much more design flexibility. Furthermore, existing visual programming environments often allow for the creation of very complex programs, in some cases even providing facilities to enter code in special nodes of the visual program. Although we do not exclude the possibility of adding more complex program structures to future versions of Mayhem, our goal for the first version of Mayhem is to make the creation of automation tasks as simple as possible, thus the only programming construct available in Mayhem is a simple Event-Reaction mechanism. To our knowledge, this minimalist approach is unique and sets Mayhem apart from other more complex systems. We believe that Mayhem's ease of use will make the domain of home and personalized automation accessible to a very wide range of users for whom existing systems are too complex to work with.

6.1.2 Concept, Design and Goals

automation tasks in Mayhem consist of *Events* and *Reactions*

Automation tasks in Mayhem consist of two basic building blocks: *Events* and *Reactions*. Events are software modules with the task of monitoring a state in the environment. For example, Events can listen for certain messages received from a sensor board, the completion of a timer, infrared signals from a universal remote, or detect motion in a certain region of a camera image. Reactions are software modules that perform a certain reaction upon being triggered (by Mayhem Events). Reactions can for instance post status updates on social media sites such as Twitter, initiate playback of sound or video files, send a text message or activate a household device interfaced with the PC.

In an example task scenario, a Mayhem user sets up his PC to prevent his cat from climbing on his precious living room couch. To do this, the user chooses an image recognizer Event, trained on the image of his cat and marks the outline of the couch in the Event configuration's preview window. He then connects this Event to an Reaction that will play a loud sound to scare away the cat when the feline comes too close to the couch. To verify if his plan worked, the user duplicates the preconfigured Event instance and attaches it to a "Record Video (10s)" Reaction, which will make a short recording each time the cat gets scared away (or not ...).

our goal was to make Mayhem easy to use for users without any programming experience

We intentionally chose the simplest possible semantic for coupling Events to Reactions—Reactions are Triggered by a unary Event. No state information other than that an Event has occurred is conveyed to the Reaction. Our motivation for the choice of this semantic is that we want to make Mayhem accessible to even the most unskilled users.

Moreover, introducing combinatorial operators such as those contained in more complex visual programming languages (Max/MSP, Quartz Composer, etc.), i.e. binary logic gates, multiplexers or function generators quickly leads to complex and confusing constructs, which conflict with our design goals for Mayhem, namely providing an extremely simple and easy-to-use tool for creating home automation tasks. However, most Events and Reactions have a configuration screen that provides the user with a reasonable amount of control over the parameters of Event or Reaction, depending on the type of functionality it implements.

6.1.3 User Interface Design

We chose a user-centered, iterative approach to design Mayhem's user interface. This involved paper-prototyping (Snyder, 2003), low-fidelity and high-fidelity prototyping stages. In total, we iterated over five revisions of the user interface. General design goals of Mayhem's interface were to enforce simplicity and to reduce visual clutter and to create a simple workflow for construction of automation tasks.

Figure 6.1: Graphical design for Events (left) and Reactions (right) in Mayhem. The graphical shape and distinct color of Events and Reactions implies the direction of events flowing from Event to Reaction and the establishment of a connection between Reaction and Event.

6.1.3.1 Graphical Representation of Automation Tasks

Automation tasks in Mayhem are represented as Mayhem *Connections*. In the remainder of the paper we will refer to Automation tasks as implemented in Mayhem as *Mayhem Connections*. Each Connection object assigns an Event to a Reaction. As can be seen in Figure 6.1, Events are graphically represented as right-facing arrows. Reactions, on the other hand, have a left-facing V-shaped notch. This visual design implies that when an Event and a Reaction are brought together, a connection between the Event and the Reaction is established and that events will flow from the Event to the Reaction.

The construction of Mayhem connections is accomplished by clicking on the "add event" and "add reaction" panes of the main user interface window. Following the click, a dialog appears that enables the user to

select the desired Event or Reaction (Figure 6.2). Once an Event or a Reaction is selected, a configuration dialog usually appears allowing customization of the functionality of that module.

6.1.3.2 User Interface

Mayhem is designed as a single window application. The central UI component is the *run list*, where users can manage a list of connections between Events and Reactions (Figure 6.2, "Run List"). By clicking on an Event or Reaction, a configuration dialog appears that allows the user to configure its behavior. A toggle switch to the side of the connection allows the user to set its activation status. Clicking on the "x" symbol deletes the corresponding connection from the run list. The items of the run list are stored persistently when the application is shut down, so the configured connections and their settings don't get lost when the application is started up again.

New connections are constructed by using the placeholder widgets above the run list (Figure 6.2, "Adding Events and Reactions"). By clicking on "Choose Event" or "Choose Reaction" a list of available Events or Reactions appears. When the user clicks on the desired Event or Reaction, a configuration dialog appears that allows her to configure its behavior. Once an Event or a Reaction has been selected, it fills the corresponding placeholder widget. When both an Event and a Reaction have been selected, this pairing is added to the run list as a new connection.

6.1.4 Implementation Details

Mayhem consists of a GUI, core and functionality layer

Mayhem consists of three implementation layers (Figure 6.3). A graphical user interface (GUI) written in C# and Windows Presentation Foundation (WPF) resides at the top level. An intermediate layer that is written in C# implements the Mayhem runtime environment, which includes the building of Mayhem Connections, event management and persistence functionalities. The core functionality of Mayhem modules is implemented in the lowest layer. Here, care is taken to make sure that functionality can be re-used for new modules or variants of existing ones.

we prefer to implement most of the functionality layer in C#

For reasons of simplicity we try to implement as much functionality at the lowest level in C#, but if special functionality (such as raw access to system input events) is required, Mayhem accesses native DLLs or custom code implemented in C++. For instance, all bindings to OpenCV

Figure 6.2: Mayhem's user interface consists of a single window. The central part of the UI is the run list, where the Mayhem connections can be managed. New connections can be constructed by clicking on "Choose Event" or "Choose Reaction" and selecting the desired Event and Reaction to combine. Clicking on the number on the lower right ("Toggle Notification Bar") opens a notification bar, that shows pending Mayhem model notifications. Some Mayhem modules post notifications in case they are not able to function correctly, i.e., if no camera is found when a "Take Picture" reaction is activated.

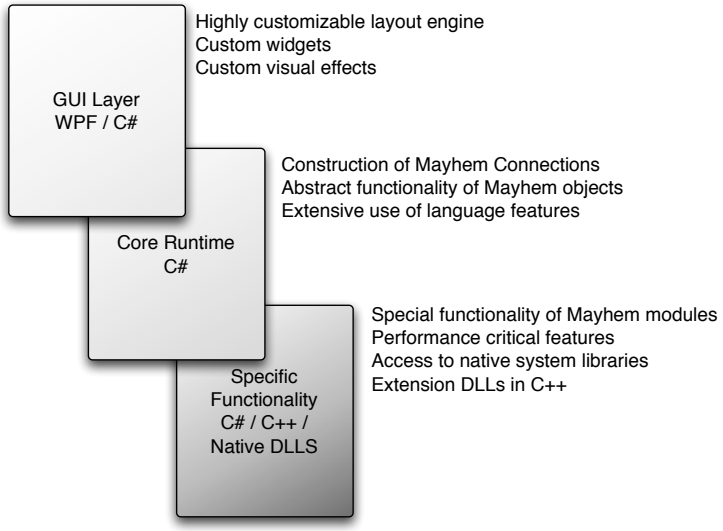


Figure 6.3: The three implementation layers of Mayhem.

are implemented at the lowest level as a C++/CLI[2] DLL accessible by the .Net Framework Common Runtime Framework (.Net CLR).

C# has many built-in features that were useful for implementing Mayhem

We chose to implement Mayhem's GUI using WPF as this gave us the needed flexibility and customizability to design the GUI to our requirements. Using C# it is possible to leverage many existing Windows technologies. Furthermore, the C# language has a number of event-oriented constructs (i.e. delegates and event listeners) that are particularly useful for the implementation of the automation semantics we defined for Mayhem. In addition, integrating lower-level extensions, such as C++ DLLs is a very easy task using this language. Thus, using C# allows developers to easily extend the functionality of Mayhem or to integrate their existing work into Mayhem.

Because the separation between the three layers is kept as high as possible, it will be possible to make versions of Mayhem that are able to run on other devices in the Windows ecosystem, such as smart phones and also other platforms supporting a .Net compatible runtime, for instance Linux or Mac OS X with the Mono Framework (Xamarin, 2012)[W].

[2]C++/CLI is an extension of the C++ language that is used to compile code towards the .Net runtime. In particular, C++/CLI allows the creation of "managed" objects in addition to standard ("unmanaged") C++ objects. Managed objects are subject to garbage collection by the .Net Runtime. C++/CLI is thus the ideal "glue language" for building low-level extensions to the .Net Framework. For more information on C++/CLI, please refer to (Sivakumar, 2004)[W].

6.1.5 Open Source Project

We have published Mayhem as an open source project[3] (Mayhem, 2012)[W]. From this website, users can download the current build of Mayhem as well as add-on packages containing more Event and Reaction modules. Mayhem supports on-the-fly updates and module extensions using NuGet (Outercurve Foundation, 2012a)[W].

The goal of the open source project is to make Mayhem available to as many users as possible, and, more importantly, to foster and encourage development of further functionality and improvements to Mayhem.

6.1.6 Discussion

Mayhem sets itself apart from similar system as it is not focused on specific task domain

Mayhem is an application aimed at facilitating the creation of home and personal automation projects for end users lacking programming skills. Mayhem distinguishes itself from similar systems because it is not targeted towards a specific expert user group, but to all users that are interested in creating simple automation tasks. We do not intend to limit Mayhem's range of functionality to one single domain, such as the generation of audio, but want to support the broadest range of functions possible for personal automation. Furthermore, by paying special attention to the design of Mayhem's user interface, and using a user-centered design approach, we try to avoid several of the barriers that learners of end-user programming systems are typically confronted with (Ko et al., 2004):

- **Selection Barriers:** Mayhem offers an easy-to-browse catalog of available Actions and Triggers and does not duplicate functionality.

- **Coordination Barriers:** All interconnections between functional modules in Mayhem are standardized. There is only a single semantic defined to interconnect Actions and Triggers. Also, the message behavior of Triggers is unambiguous for the user, as only a binary event is sent to the connected Action.

- **Use Barriers:** There is no need to construct complex functionality. The complexity is contained within individual Actions or Triggers, and users employ this complexity only on the abstract Trigger–Action relationship.

[3]The development repository is hosted on CodePlex (Dietz et al., 2012)[W]. The ownership of Mayhem has been transferred from Microsoft Inc. to the OuterCurve Foundation (Outercurve Foundation, 2012b)[W], where Mayhem is to be hosted in the "research accelerator gallery".

By minimizing these barriers, we also aim to make it far less likely that the users encounter *gulfs of execution and evaluation* (Norman, 2002).

We believe that Mayhem has the potential to become a valuable tool for supporting personal automation in the future. Mayhem purposely has an extremely simple semantic to connect Actions and Triggers. HCI literature has long supported the premise that the value of an application's functionality is closely related to the application's usability (Goodwin, 1987). As the number of users grows and the users become experienced with the concepts and the usage of Mayhem, we intend to provide more complex connection semantics and add more advanced features to Mayhem, depending on the growing requirements of our user base.

6.1.6.1 Future Work

Our vision for future versions of Mayhem is to convert the current singular application to a service-oriented architecture. We thus envision the entire computing ecosystem of a home user; PCs, Tablets, Smartphones or even advanced appliances, will run a Mayhem node that will provide Triggers and Actions. A centralized Mayhem interface that is abstracted from individual devices could then be implemented to run in the user's local cloud. Smartphones and other devices (even cars) enabled with localization functionalities could provide Mayhem users with functionality that extends well beyond their homes.

6.2 Example Usage Scenario for Mayhem

In the following we present an example usage scenario for Mayhem. The task is to make a media player running on Windows 7 controllable via gestural input from a mobile device. To accomplish this, we will use Mayhem to connect Android mobile application that performs motion gesture recognition with the controls of the media player application[4].

For our usage scenario, we developed a new Mayhem event that listens on a UDP socket for a message from the phone. Upon reception of a certain gesture ID the event triggers a connected media player reaction that controls one or several functions of the media player. The reactions for controlling the media player are already included in Mayhem.

[4]The operating system hooks for media player control in Windows will control any generic media player application, the specific application we use is VLC 2.0.0 (VideoLAN Organization, 2012)

6.2.1 Mobile Application

The mobile application uses a Java implementation of Protractor3D (Section 5.3) for gesture recognition. Delimiting the start and the end of a gesture is accomplished with a *push-to-gesture* button. The application stores gesture templates in a SQLite database.

the app's functions are: gesture entry, training sample entry and library management

The user can access three basic functions via the application menu: recognizing motion gestures, inputting training samples, and managing the gesture library. Figure 6.4 shows a wireframe representation of the gesture application's user interface.

the gesture ID is transmitted over wifi via UDP broadcast

When a gesture has been successfully recognized, the mobile application transmits a UDP broadcast datagram containing the recognized gesture ID via the phone's wireless connection. The companion Mayhem event listens for these broadcast messages and triggers when the correct gesture has been recognized.

6.2.2 Mayhem Event

To receive notifications via UDP datagrams, we had to implement a new Mayhem Event in C#. Implementing an Event basically starts with subclassing from `MayhemCore.EventBase`. If a graphical configuration is to implemented for the Event, the `MayhemWpf.ModuleTypes.IWpfConfigurable` interface needs to be implemented by the Event. Each Mayhem Event has two C# attributes. `[DataContract]` enables the persistent storage (serialization) of certain fields belonging to the Event, and the `[MayhemModule]` attribute sets metadata about the Event that is later displayed in the Mayhem GUI.

```
[DataContract]
[MayhemModule("Phone Gesture",
"Monitors a UDP port for a gesture event sent by a phone")]
public class PhoneGestureEvent : EventBase, IWpfConfigurable{
        ...
}
```

The basic functionality of our "Phone Gesture" Event is to monitor a UDP socket. The socket was implemented using the Singleton pattern[5], since multiple "Phone Gesture" Events can listen on the same socket to trigger multiple Reactions simultaneously.for reception of a predefined gesture ID code. When data is received by the socket, it calls the Event's `SocketReceived` method, to notify the module that data has been received. The Event checks if the received string matches the pre-defined gesture id and triggers its associated Reaction by calling `Trigger()`:

[5]At this point, we omit the detailed socket code for reasons of simplicity. Please refer to the source code provided on the Dissertation DVD.

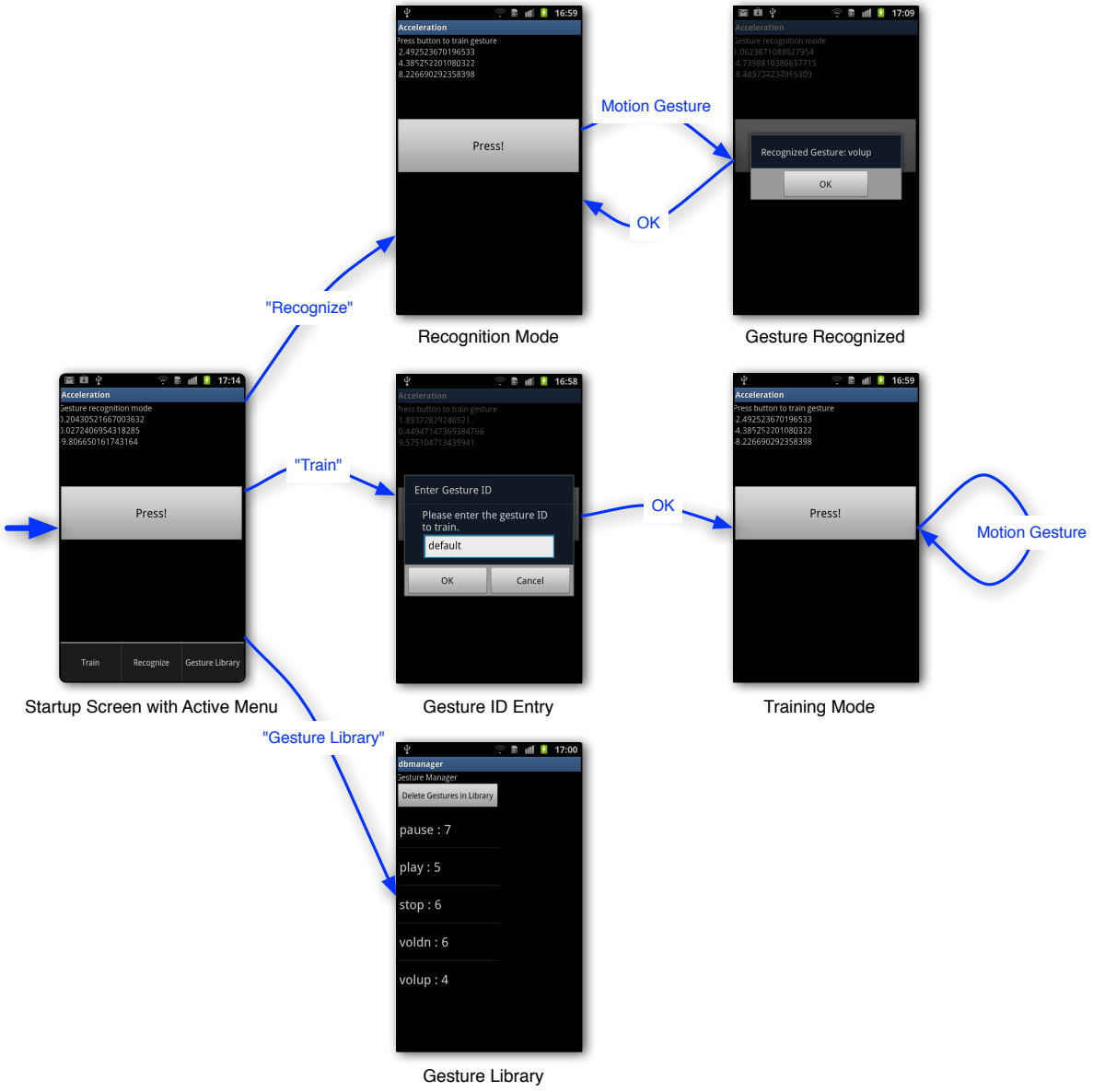


Figure 6.4: Wireframe of the Android motion gesture application's user interface. From the startup screen (gesture recognition mode) the user can select either recognition mode ("recognize"), training mode for gesture entry ("train") or view the contents of the gesture library ("gesture library"). In training and gesture recognition mode a push-to-gesture button ("Press!") is used to delimit the start and the end of gesture entries.

```
private void SocketReceived(object sender, SocketReceivedEventArgs args){
    string message = args.message;
    if (message.Equals(listenForMessage)){
        Trigger();
    }
}
```

Our "Phone Gesture" Event has a minimal user interface to input the desired gesture ID for triggering. It is implemented in Windows Presentation Foundation (WPF), a GUI toolkit for Windows. Figure 6.5 shows the UI displayed by the Event, which presents a simple text box to enable the input of the gesture ID on which the Event triggers.

Figure 6.5: The UI for our "Phone Gesture" Event features a single text box for entry of the gesture ID on which the Event triggers.

6.2.3 Setting up Events and Reactions to Control the Media Player

We want to use Mayhem to pause and resume media playback, and also to mute and unmute the audio volume. To do this we can use the "Media Play/Pause" and the "Volume: Mute" Reactions, which are already included in Mayhem.

One instance of "Phone Gesture" is set up to listen for a "play" gesture from the mobile application, and is connected to "Media Play/-Pause". A second instance of "Phone Gesture" is configured to listen for a "mute" gesture from the phone and is connected to "Volume: Mute". Figure 6.6 shows the completed configuration.

6.2.4 Discussion

In the scenario described previously, we used Mayhem to enable controlling the media player using motion gestures input via an application running on a mobile phone.

Implementing the Mayhem Event to listen for datagrams sent from the phone requires programming skills. However, once this Event has been developed, users without knowledge of programming can use it with Mayhem to create their own gesture-based controllers for other applications in a very easy way. For instance, Mayhem also has pre-built

Figure 6.6: Mayhem configuration that is used to control the media player via phone gestures.

reactions that can be used to control the display of slides in PowerPoint.

The example application shows how complex functionality provided by Mayhem's Reactions and Events can enable end users without any programming skills to create new functionality that would have been impossible for them without using Mayhem. The application scenario also illustrates the need for a critical mass of Events and Reactions to be implemented for Mayhem in order for the application to be appealing and useful for its target audience.

6.3 Summary

In this chapter, we presented Mayhem. Mayhem is an end-user programming environment target at end-users with no programming experience. Because of its simple event–reaction semantics, Mayhem allows any combination of an event and a reaction, which makes it easy to integrate new functionality in the application. Mayhem has been released as an open-source application with the goal of fostering a community

of developers and users that will help expand the scope of functions end users will be able to perform with PCs.

To demonstrate how sensor-based interfaces can be prototyped easily without writing additional code, we developed a mobile gesture recognition application for Android, which, when coupled with a Mayhem event we developed, can be used to trigger events provided by Mayhem. In this way, an end-user can make easy customizations of his operating system to support gesture-based input via his mobile phone.

Chapter 7

Future Work

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

—Alan Turing

In this thesis, we presented a series of contributions in the domain of sensor-based user interfaces for mobile devices. For most of these contributions, a number of further aspects remain open for future study, and novel user interface concepts can be developed based on the ideas and insights gained through the present work. This chapter presents recommendations for future work related to the main themes covered in this thesis: *Continuous Interaction and State-Space Systems* (Chapter 3), *Around-Device and Sensor-Based Interaction* (Chapter 4) and *Motion Gestures* (Chapter 5). The discussion of future work is thus presented order of these main themes.

In Section 7.1, we discuss further research into using state-space systems for tilt-based pointing as well as using them to process sensor data for around-device interaction (ADI). Following that, we consider future work on ADI in Section 7.2, in particular discussing two input paradigms for extending the input capabilities of *PalmSpace* (Section 4.4). Finally, in Section 7.3.2, we present a scheme for a machine learning-based method for the segmentation of motion gestures performed with mobile devices.

7.1 Further Examination of the Usability Benefits of State-Space Systems for Mobile Input

In Chapter 3, we used state-space systems to implement automatic zooming for an evaluation of user interfaces for tilt-based mobile map

navigation. Of course, substantially more research on the application of state-space systems to mobile user interfaces is still possible in the future. In the following, we propose two research ideas for future research on this topic.

7.1.1 Tilt-Based Pointing

Tilt-based user interfaces for mobile devices have recently become popular through gaming applications. Nevertheless, other uses for tilt have also been examined, such as controlling a cursor on a public display (Boring et al., 2009) or fusing tilt input with additional input modalities (Hinckley and Song, 2011). In future work, it would therefore be interesting to explore the use of state-space systems to model tilt-based cursors and compare such a system to more common input mappings for tilt-based pointing, such as linear or quadratic mappings. Because physical properties, such as acceleration, friction and momentum can be simulated using state-space systems, mappings could be created that appear more intuitive to the user.

state-space systems could be used to model more efficient mappings for tilt-based pointing

The intuition behind this is that many users have already interacted with physical tilt-based "interfaces", i.e., tilt-based marble mazes, so it is plausible to assume that they expect the cursors behavior to exhibit the traits of a real ball, with acceleration, friction and momentum, albeit expressed with more subtlety than a fully correct simulation, in order to make pointing more effective.

To compare state-space systems with other mapping alternatives, we suggest conducting Fitts' Law evaluations. The hypothesis here would be that using a mapping based on a state-space system would significantly increase the throughput of the tilt-based pointing task. Previous work in this area has covered an analysis of the influence of physically modeled auditory feedback on tilt-based input (Rath, 2007).

7.1.2 State-Space Systems for Around-Device Interaction

Modeling user input for around-device interaction through state-space systems may be beneficial. For around-device interaction, a large amount of sensor data needs to combined to generate the state of the system. This makes defining direct mappings from sensor values to user interface parameters difficult. In addition, the reliability of user interfaces that use a large number of sensors for input is directly dependent on the reliability of the sensors used. In mobile usage scenarios, sensor data may not be 100% accurate or available at all times.

Thus, state-space systems may be a useful approach to cope with unreliable sensor data, the prime example being the Kalman Filter (Kálmán, 1960), which maintains the most likely state, e.g., a tracked object's position, based on (error-prone) sensor readings and previous states (Marsland, 2009).

state-space systems can compensate for intermittent or noisy sensor readings

The main advantage of state-space systems is that they can model a closed-loop system. This gives them the ability to keep generating new states, that can be used as inputs to UI parameters, even when sensor readings are noisy or intermittent. This, e.g., was some times the case when tracking of the user's palm in *PalmSpace* (Section 4.4) was lost, halting any interaction with the system until the hand was re-acquired. In this case, a state-space system could use information about the user's previous movements to keep generating updates that appear plausible, in order to mask the momentary loss of tracking.

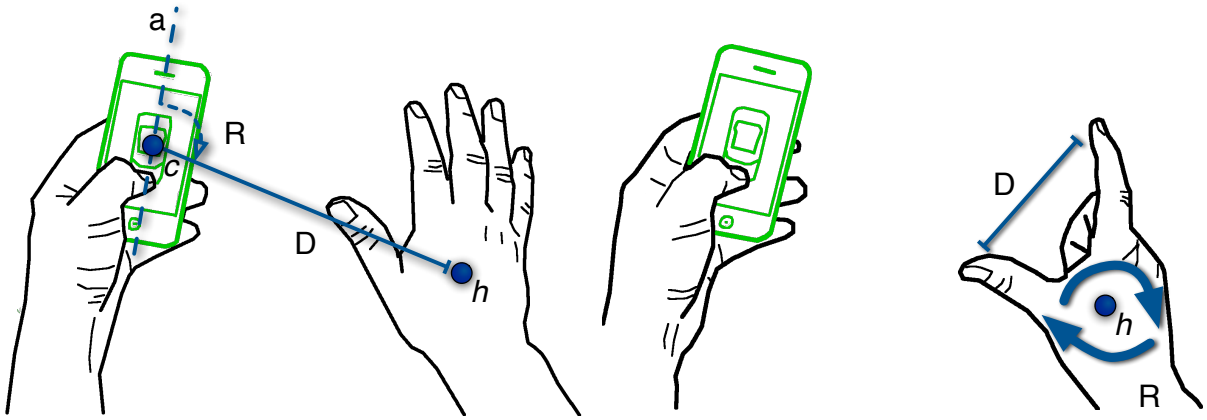
Around-device interaction could also be used to control 2D or 3D virtual entities that are subject to physical phenomena. Simulation engines supporting rigid-body dynamics, fluid dynamics, mass-spring systems or soft-body physics exist, e.g., (bulletphysics.org, 2012; Havok.com Inc., 2012; NVidia Inc., 2012)[W]. These physics engines can thus also be regarded as (very complex) state-space systems and need to be taken into account when evaluating state-space systems for around-device interaction.

7.2 Future Projects in Around-Device Interaction

HoverFlow (Section 4.3) and *PalmSpace* (Section 4.4) were developed to explore the design space of around-device interaction, and to demonstrate what types of around-device interaction (ADI) can be achieved with the given sensor technology. ADI has recently become a popular field in research and development, with technology companies (Ident Technology AG, 2012)[W], car manufacturers (Herrmann et al., 2010) and the home entertainment industry (Tomsguide.com, 2012)[W] investing a substantial amount of effort.

ADI with mobile devices needs to be further explored in future work

The design space for ADI remains, at present, less explored for interaction with mobile devices than with fixed devices. The relevant projects in this thesis can be considered as an excellent starting point for research on future ADI interface concepts. In the following, we will present some suggestions for research into new mappings for 3D object viewing based on hand gestures, extending the ideas realized in *PalmSpace* prototype. After that, we present a suggestion on how to realize around-device output through wearable and steerable mobile projection.



(a) Device-Centric **(b)** Hand-Centric

Figure 7.1: Two possible paradigms for extending the design space of mobile around-device interaction for 3D object viewing proposed in *PalmSpace* (Section 4.4). **(a)** The *device-centric* approach makes all interaction relative to the center point of the mobile device or the virtual object that the user is currently controlling. **(b)** By contrast, in the *hand-centric* model all inputs are relative to the user's hand.

7.2.1 Additional Input Mappings for Interaction Using Mobile Depth Cameras

PalmSpace is an interface for 3D object based on around-device hand gestures. In order to determine rotation values, *PalmSpace* tries to fit a plane to the user's outstretched palm. Due to limitations in the depth imaging hardware, gathering rotational information by other means, e.g., fingertip tracking, was not possible with the desired reliability. In addition, we did not implement 3D translation due to the high noise content in the depth readings of the SwissRanger 4000 depth camera we used.

Given mobile depth cameras with a smaller size, improved noise characteristics and a wider field of vision than what is available at present, *PalmSpace* could easily be extended to allow 3D translation. It also plausible that the outstretched palm will no longer be needed in the future, and that rotation could be inferred by analyzing the 3D point cloud of the user's hand captured by the depth camera.

For future 3D object viewing interfaces, we propose two general input paradigms, *device-centric* and *hand-centric*. These two paradigms are illustrated in Figure 7.1 and are discussed in the following two sections.

7.2.1.1 Device-Centric Input

all input is relative to a center point on the device

In the device-centric paradigm, all input is relative to the center of the device or the center of the virtual object (Figure 7.1 (a), c) that is being viewed. The distance from the device to the center of the user's hand (Figure 7.1 (a), D, and h), can be mapped to object translation or zoom. Rotation (Figure 7.1 (a), R) is calculated relative to the device by calculating the (3D) angle between the line $\overline{ch}$ and the device centerline axis a, effectively turning the hand into a rotation "handle".

The advantage of the device-centric paradigm, is that using the hand as a rotation handle allows the user to vary the control gain depending on the distance from the hand to the device. Similar to the original interface in *PalmSpace*, the user can use proprioception to correctly position his hands while keeping his focus on the object on the mobile device. A possible disadvantage of this paradigm is that certain rotations will require the hand to be moved in front of the display, thus occluding the mobile device's display. Also, a range of rotations won't be possible without clutching, since the user's non-dominant arm obstructs a part of the interaction space.

7.2.1.2 Hand-Centric Input

All input is centered on the hand, specifically its center point (Figure 7.1 (b), h) in the hand-centric paradigm. The size (Figure 7.1 (b), D) of pinch gestures can be mapped to object translation or scaling. Alternatively, free-form 3D object translation could be implemented by using h to directly map the position of the virtual object. The rotation (Figure 7.1 (b), R) of the hand around h can be mapped to object rotation.

The advantage of the hand-centric paradigm lies in its direct input property. Hand rotation is directly mapped to object rotation and the size of a pinch gesture is directly mapped to object translation or scaling. Compared to device-centric rotation, there is no (abstract) object handle. Possible disadvantages of the hand-centric paradigm are that the range of unclutched rotation may be more limited by physiological constraints than device-centric and that the translation or zooming range, when using a pinch gesture is more limited compared to the device-centric paradigm.

7.2.1.3 Discussion

In the preceding sections, we suggested two possible paradigms to follow in extending the around-device interaction work of *PalmSpace*. We

would like to note that at this time, these paradigms are not yet evaluated and may not be the optimal way to design for 3D input in around-device space. *Hand-centric* and *device-centric* are, however, meant to serve as a basis for initiating future research into this topic.

the design of novel ADI techniques is highly dependent on the development of sensing technology

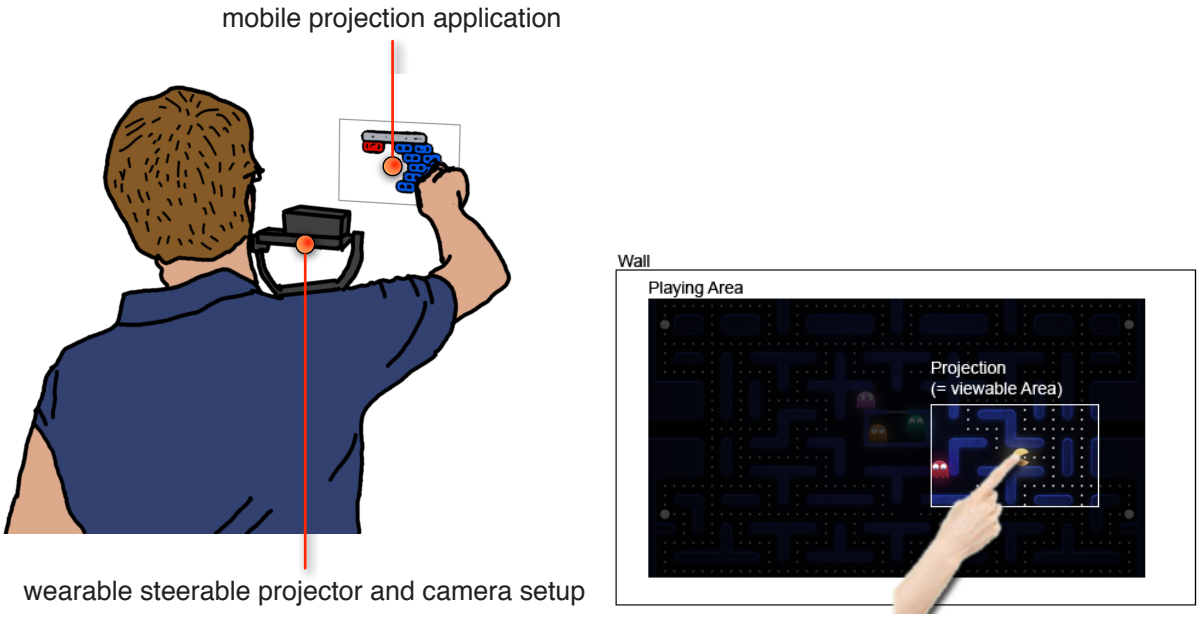
A further point to note is that realization of the two paradigms is extremely dependent on the development of depth imaging technology. Without the resolution necessary to detect individual fingertips, a free-space pinch gesture (as suggested in the *hand-centric* paradigm), for example, could not be properly implemented. Thus, most of the proposed input techniques that comprise the two paradigms can, at present, only be properly implemented and evaluated using an optical tracking system, such as OptiTrack (NaturalPoint Inc., 2012)[W].

7.2.2 Around-Device Output Using a Wearable and Steerable Projector

In Chapter 4, we focused mainly on the input side of around-device interaction. Around-device output, however is also an interesting subject for future research. We propose the use of wearable, steerable mobile projectors as one possible channel for around-device output for mobile (and wearable) devices.

Steerable and wearable projectors are able to generate output on a large area in front of the user during mobile use. Projecting information on the users themselves is also possible (Harrison et al., 2011). Furthermore, such mobile projectors have the ability to follow the users *locus of attention* (Raskin, 2000), i.e., by tracking the position of the user's hands (Kratz et al., 2012b).

We feel that body-worn, and especially steerable projectors solve one of the fundamental weaknesses of current mobile projection interfaces: the fact that the user needs to control aim the projection itself. We believe that this distracts the users from their main tasks too heavily and that user's would be much more comfortable and efficient using mobile projection interfaces if the need to control the actual projection didn't exist. For instance, (Cauchard et al., 2011) suggest that typical configurations of handheld pico projectors are unsuited to many projection tasks, because they couple the device orientation to projector orientation in a fixed way and argue for steerable handheld projection. Using a mobile, steerable projector, we aim to address a number of research questions that have not been addressed in previous work.



(a) Shoulder-mounted and steerable projector **(b)** Peephole Pac-Man

Figure 7.2: Using a stabilized mobile steerable projection allows the projected image to follow the user's locus of attention. **(a)** conceptually shows a shoulder-worn projector and camera setup that racks the user's right hand. **(b)** an example gaming application for our proposed set-up, *Peephole Pac-Man*. By using a steerable projector which follows the user's hand, a playing area much larger than the actual projection is realized.

7.2.2.1 Fitts' Law for Peephole Pointing

Peephole pointing (Yee, 2003) allows interaction with large workspaces that contain more information than can be displayed on a single screen. This technique provides a movable window that reveals a portion of the larger content the user is working with. Peephole pointing is thus a suitable technique for use with mobile projectors due to their rather limited projection throw angle.

For peephole pointing, we hypothesize that systems such as a wearable steerable projector will lead to a higher throughput compared to solutions where the mobile projector is controlled by hand. Our intuition here is that pointing with the finger is likely to be faster than pointing through a proxy such a mobile projector. Of course, a prerequisite for good performance in a Fitts' Law evaluation is a well-functioning gimbal with sufficient speed to track fast hand movements[1].

Peephole interfaces have been previously evaluated in a Fitts' Law context by (Cao et al., 2008; Rohs and Oulasvirta, 2008), but mobile projection has, at this time, not been covered yet.

[1]The rotating mass of a gimbal can be significantly reduced by projecting on a steerable mirror, rather then moving the projector directly.

7.2.2.2 Focus-In-Context Displays

Focus-in-context interfaces usually consist a large scale, low resolution (or low detail) representation of the scene with a smaller, higher-resolution window that is movable across the low-resolution content to reveal in high resolution and high detail content in a local scope. A possible focus-in-context interface scenario for a wearable steerable projector could be the use of our system in conjunction with a large projected image created by a full-size projector in a room.

7.2.2.3 Interaction on the Body and in the Environment

A wearable, steerable projector affords seamless switching between input on the body and in the environment. This allows the implementation of interesting design ideas, such as moving virtual objects between a projected surface on the body to a projected in the environment and vice-versa. For example, a picture from a user's mobile could be transferred to an interactive surface by dragging its projection on the user's palm to the surface. This process, mediated by the steerable projector, would appear seamless to the user.

7.3 Automatic Segmentation Strategies for Motion Gesture Recognition

Segmentation of gesture entries for user interfaces based on motion gesture recognition is still a problem that hasn't been solved satisfactorily. Ideally, we should not require a "push-to-gesture" button to the segment the start and end of a gesture. Algorithms that analyze the incoming sensor data from the accelerometer and gyroscope should instead determine when a gesture entry has started and when it has stopped.

7.3.1 Discussion of Previous Approaches

an explicit prefix gesture may be difficult to change once it has been incorporated in the mobile UI

DoubleFlip (Ruiz and Li, 2011) attempt to address this problem by using a single, very reliable prefix gesture to delimit the entry of further gestures. The detection of this prefix gesture is very robust, since its classifier was built from a very large data set. The problem with this approach is that, while it is more elegant than an on-screen delimiter button, it remains a manual delimitation step. Furthermore, it is unclear if it is easy to switch to an alternative delimiting gesture, since the

factors influencing the detection accuracy of individual motion gestures have not yet been studied formally.

threshold-based techniques are difficult to adapt to a wide range of use environments

(Ashbrook, 2010b) suggests using a threshold on the variance of the last N samples as a decision criterium for gesture segmentation. This technique may be useful in controlled settings and for predesigned gesture vocabularies, where the correct threshold setting can be determined by prior evaluation. It is plausible to assume that this strategy will not perform well when deployed in the field, for example in situations where there is substantial "noisy" motion coming from the user's environment, i.e., when using a mobile device while using public transport.

7.3.2 A Machine Learning Approach to Segmenting Motion Gestures

We believe that machine learning can be used to create more effective, and, what is more important, more general motion gesture segmentation algorithms, if a sufficiently large amount of training data is available to create a meaningful model of the characteristics of the start and the end of motion gesture entries.

Metric	Value
average	286
median	266
standard deviation	109
max	1766
min	24

Table 7.1: A set of exploratory statistics on the number data samples per gesture entry. The basis of this data is the gesture entry data set also used in Section 5.4.

To this end, we have conducted some exploratory studies on the data set used in Section 5.4. To recapitulate, this data set consists of a total of 3507 gesture entries sampled at 100 Hz. The sensor values measured were acceleration, angular rotation and attitude (a compound value calculated from the device's accelerometer, gyroscope and magnetometer). Table 7.1 shows a set of exploratory statistics on the number data samples per gesture entry. This data can be used to set the number of samples used to define the beginning, middle, and end of a motion gesture entry, as depicted in Figure 7.3.

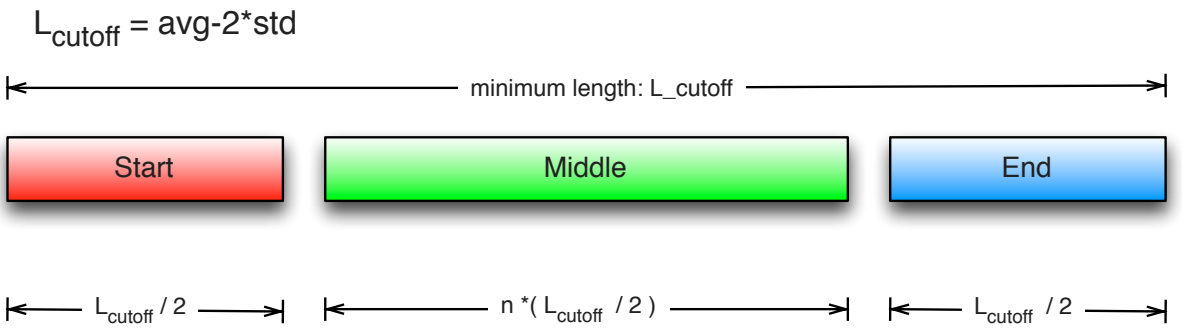


Figure 7.3: The proposed segmentation strategy is to build a classifier for each part of a motion gesture entry. To do this, we segment a data set of example gesture entries into three parts, *start*, *middle* and *end*. We propose the use of empirical measurements to determine L_{cutoff} in order to set the correct number of sample points for each the segment types.

Concretely, our proposed segmentation of the gesture data is based on calculating a value L_{cutoff}, with

$$L_{\text{cutoff}} = \mu - 2\sigma \tag{7.1}$$

and μ, σ obtained from our analysis of the gesture data entry set (Table 7.1). L_{cutoff} is also used to filter out gestures that are unusually short , i.e. to be used for building a segmentation model a gesture must have a length of L_{cutoff} or greater.

Segmenting the gesture data set like this, we obtain three labelled data classes, *start*, *middle* and *end*. To validate our assumptions that the data would be separable we conducted a Principal Component Analysis (PCA) (Jolliffe, 2002) on the labeled data, using all gestures in the data set as well as randomly chosen subsets of 250 and 512 gesture entries. Figure 7.4 shows 2D and 3D plots of the PCA. Visually, it indicates that *start* and *end* gesture segment classes can be separated by a classifier from *middle* with relatively good accuracy since the labeled points are spatially well separated. There appears, however, to be substantial intermixing of *start* and *end*, which may pose a problem (at least when using linear classifier on the data at this degree of dimensionality reduction).

To obtain some initial classification results, we trained a Support Vector Machine (SVM) classifier with a Radial Basis Function (RBF) kernel with the segmented and labeled data and performed a 6-fold cross-validation. The gesture segments were correctly classified with an accuracy of 88.1%.

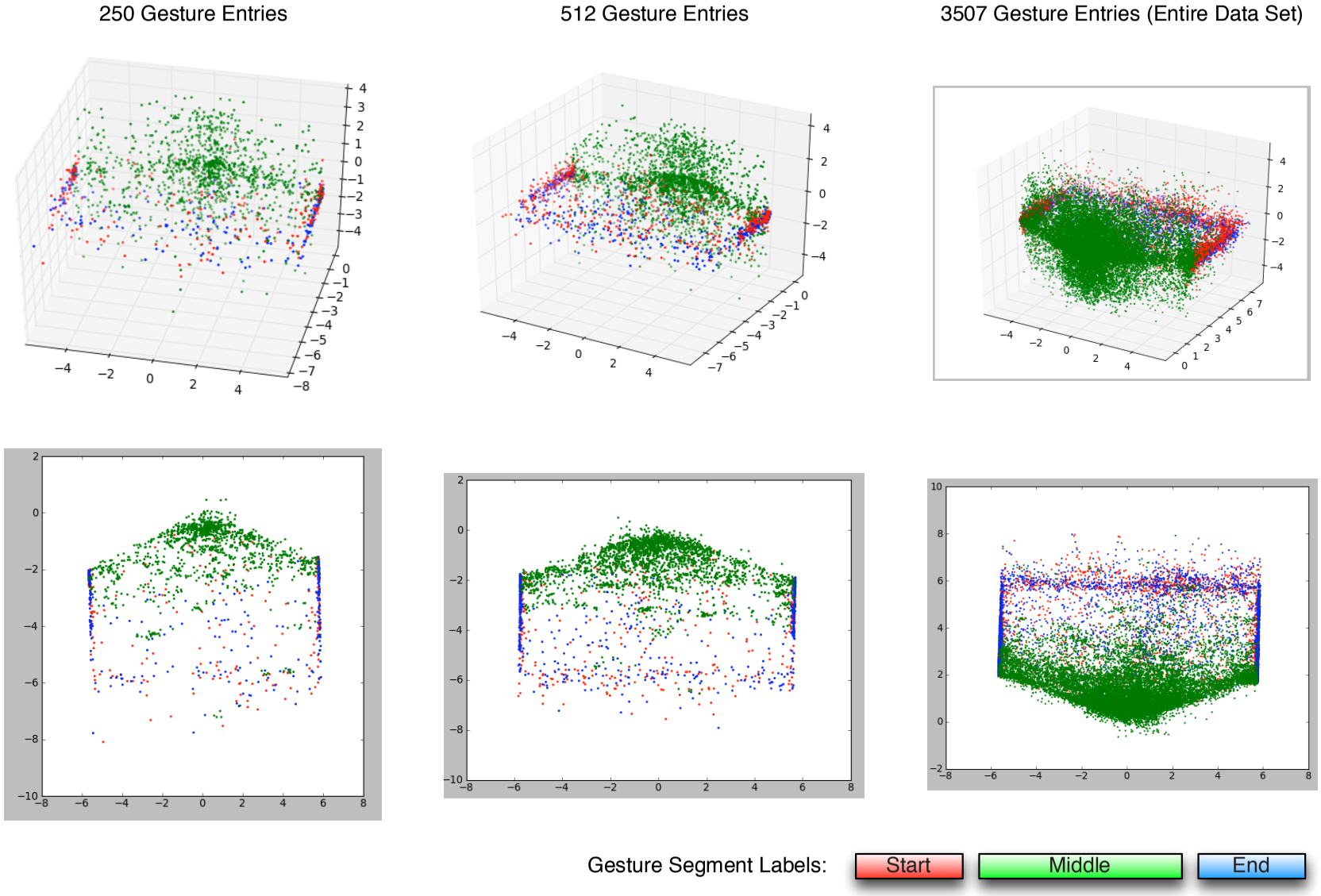


Figure 7.4: We segmented each gesture entry from the data set used in Section 5.4 in to three parts, *start*, *middle* and *end*. This figure shows 3D and 2D plots (top and bottom rows, respectively) of PCA we conducted on the segmented and labeled gesture entry data. From left to right, the plots show the results for a subsets of 250 and 512 entries, as well as the entire gesture set in the right column. The plots provide a visual indication that the *start* and *end* gesture segment classes can be separated by a classifier from *middle*.

7.3.3 Aspects that Require Further Work

The results presented in this section are exploratory in nature and should be regarded as a promising basis for future research. In particular, there are two aspects that need to be addressed to improve their significance:

- Recordings of the sensor readings immediately before and after a delimited gesture are missing from the data set analyzed. These time intervals may include important information that could improve the classification of the gesture segments. A new and extensive data set will need to be gathered.

- Since we want to delimit gestures from other everyday movements, a further data label, *noise*, will need to be added to the existing classifier, in order to show that it is possible to actually delimit a gesture entry from uninteresting motions of the mobile device. This new label, too, will require the recording of substantially more sensor data from users of mobile devices.

Chapter 8

Conclusion

In this dissertation we demonstrate that the input expressiveness for mobile user interfaces can be improved by making better use of the sensors available on the mobile devices. This has the effect of increasing the usability of mobile user interfaces, measured by quantitative and qualitative indicators. To come to this conclusion, we developed and analyzed several novel interaction concepts making use of sensor data from sensors that are currently built in to mobile devices, as well as sensors that are likely to be incorporated into future mobile devices.

In this chapter, we summarize the work presented in this dissertation and enumerate the contributions we achieved. We conclude by contextualizing the main contributions of this dissertation and discussing its impact for the future of mobile HCI.

8.1 Summary and Contributions

In Chapter 1, we give a brief overview of the technological development of mobile interaction. We observe that the development of mobile user interfaces is closely coupled with the evolution of the sensory capabilities of the device. This motivates us to conduct research on sensor-based mobile user interface concepts in this dissertation, because the development of new sensing technologies for mobile devices will continue into the foreseeable future. We identify three areas of interest for research into sensor-based interface concepts on mobile devices: *continuous interaction*, *around-device interaction* and *motion gestures*.

8.1.1 Continuous Input

In Chapter 3 we examine mobile interfaces supporting *continuous interaction*. The idea here is to use the additional sensing capabilities of mobile devices to help users perform more efficiently in tasks that require repetitive input actions, e.g., map scrolling on a touch-based mobile device. *State-space systems* can be used to map a low number of input parameters to a large number of output parameters. Such dynamic models also allow modeling the parts of the user interface as a physical process, which can improve the user's mental model of the user interface during interaction. We employ a state-space system to create a *Speed-Dependent Automatic Zooming* (SDAZ) engine for our investigation on how to improve mobile map interfaces controlled by tilting the device.

Semi-Automatic Zooming addresses several usability problems of standard SDAZ interfaces

We choose to study tilt-based mobile map navigation with automatic zooming because on touch-based devices, map navigation interfaces without automatic zooming require repetitive scrolling and zooming gestures and can thus lower the productivity of the user. On the other hand, we feel that fully automatic zooming can by itself cause user errors, such as the hunting effect (Section 3.1.1), in some situations. Thus, we propose *Semi-Automatic Zooming* (SAZ), a technique that allows the user to control the base zoom level of an automatic zooming interface manually. SAZ avoids the hunting problem by allowing the user to retain a given zoom level even when the map is not scrolling. In addition, the work in Section 3.1 contributes a software testbed for mobile map navigation that uses real map data in the form of locally stored map tiles. We extend a state-space model used by (Eslambolchilar and Murray-smith, 2004) to support two-axis tilt input for SDAZ. The testbed scales to allow map navigation across a large geographic area.

To evaluate SAZ, we present the results of a study comparing SAZ, SDAZ and multi-touch. Our results show significantly lower task completion times and higher USE ratings for SAZ in comparison to SDAZ. There is no significant difference between SAZ and multi-touch, probably owing to user familiarity with touch input as well as the excellent support for multi-touch in the testing apparatus (an Apple iPhone 3GS). We notice that the tilt-based input for map navigation has a significant learning curve, with users' performance improving noticeably towards the end of the trials. As such, we suggest using SAZ as a useful companion technique for complex map navigation tasks on devices with multi-touch support, in situations that do not afford multi-touch use (i.e, when only one hand is available to hold and use the mobile device), and as the main technique on devices without multi-touch input capabilities.

In consideration of the positive results we obtained for SAZ for tilt-based input, we decided to evaluate SAZ and SDAZ for flick-based input. The results of this work are presented in Section 3.2. Our motivation here is to combine the advantages of SAZ with multi-touch, which, as our previous study indicates, is more familiar to use for the users. We obtain insights into the correct mappings for flick-based input for map scrolling and modifications to the original tilt-based state-space model (Section Section 3.2.3) to support tilt-based input. We describe the results of a preliminary user study (similar to the one in Section 3.1.3), comparing two flick-based techniques with tilt-based SAZ and multi-touch. Against our expectations, the flick-based techniques perform worse than SAZ and multi-touch. In future work, a more extensive study with a further iteration on the implementation will need to be conducted to verify or invalidate this observation.

embedded sensors on current mobile devices can be used to build expressive UIs and are currently underused

Looking at the results from Chapter 3, we can argue that mobile user interfaces using sensors providing a continuous data stream, such as accelerometers, can be used to drive expressive interactions, such as mobile map navigation. The work also shows how the embedded sensors on mobile devices are under-used in terms of enhancing the input capabilities of the user interface. A further observation we can make is that, while state-space systems are state-of-the-art and can yield very impressive results in terms of realism, they can be difficult to calibrate and to adjust to particular user interface requirements. Future work should therefore focus more generally on the use of state-space systems for mobile user interfaces, in order to be able to generate recommendations for the choice of model and parameters for particular sensor types and application areas.

8.1.2 Around-Device Input

Around-Device Interaction increases the expressiveness and possibilities for input on mobile devices

In Chapter 4 we look at sensor technologies that could be incorporated into future mobile devices. We develop novel mobile UIs using these sensor technologies and evaluate their usability in comparison to current UIs. The goal of these evaluations is to enable us to determine what sensor types can be used in future mobile devices and how the usability of these devices can be improved by new types of mobile UIs. To this end, we also introduce the concept of *Around-Device Interaction* (ADI) in Section 4.1. The idea of ADI is to increase the expressiveness and possibilities of mobile input by enabling input on the entirety of the device's surface and also the space surrounding the device. A fundamental motivation of ADI is to break through the constraints of the 2D touch found on most contemporary mobile devices.

HoverFlow is a proof-of-concept that demonstrates the feasibility of around-device interaction

HoverFlow is the first prototype we built to evaluate-the concept of ADI. The prototype demonstrates that it is possible to implement ADI by using a simple set of sensors. *HoverFlow* contributes a set of coarse input gestures for front-of-device interaction, useful in situations where touch manipulation of the device is not possible.

The concept of around-device gestures is expanded upon by *PalmSpace* (Section 4.4). Here, we use a depth camera attached to a mobile device which provides depth information with a significantly higher resolution than that of the sensors with which *HoverFlow* was equipped. The aim of *PalmSpace* was to allow precise gestural input in the around-device space, allowing the simultaneous manipulation of user interface parameters with free-space gestures. To accomplish this, we developed a palm-based user interface metaphor, based on tracking the pose of the user's outstretched palm. As a task to evaluate, we choose 3D rotation input, since this type of task is relatively difficult to perform on touch screens due to occlusion and only two degrees of freedom for input.

we demonstrate that depth cameras could be a useful input device for future mobile devices

We contribute insights into the correct depth camera orientation for around-device interaction (to the side of the mobile device) and empirically confirm our hypothesis that rotation tasks can be performed with a lower task completion time with around-device gestures than with direct input on the touch screen. On a technical level, we show that it would be feasible and beneficial to incorporate depth cameras on future mobile devices, once this technology has been sufficiently reduced in size.

Regarding interaction on the surface of mobile devices, we explore dual-sided pressure-based mobile interaction using the *iPhone Sandwich*, a prototype mobile device with these input capabilities which we patented (Essl et al., 2011). One of the advantages of the iPhone Sandwich is that it allows a high local expressivity for local input, affording interactions that are not possible with traditional single-sided touch input. Interactions resembling everyday actions such as squeezing, twisting, grabbing and turning become possible on the mobile device. We contribute a number of application scenarios in this design space.

we resolve contradictions in the literature concerning the correct mapping of pressure to input variables

Noticing contradictory statements concerning the correct mapping of pressure input to user interface output in the literature, we study the characteristics of measuring circuits and mapping functions for pressure-based input in Section 4.5 as well as the influence of handheld device pose on the accuracy of pressure input. Furthermore, we observe that the performance of pressure input, measured as the target acquisition time for our task, does not differ significantly when pressure is applied to the mobile device lying on the table, from both sides of the device or from the rear of the device. The target acquisition time for pressure input from the front, however, is significantly lower

than for the other poses, indicating that the other compared poses are preferable.

we contribute a rear-of-device virtual trackball that avoids discontinuities using a gaussian mapping

Input of 3D rotation is a difficult task on single-sided touch-based devices. In Section 4.6 we present the results of a study comparing front and rear-based virtual trackballs with a tilt-based input technique. We did not find a significant difference between front and rear trackball input for 3D object rotation. However, we did contribute a rear-of-device virtual trackball implementation using a Gaussian mapping functions that avoids the discontinuity at the edge of standard virtual trackballs using a spherical mapping, and allows occlusion-free rotation of 3D objects.

8.1.3 Motion Gestures

We present a number of contributions in the domain of motion gestures for mobile devices in Chapter 5. Since motion gestures are currently under-used in mobile user interfaces, one our goals is to develop gesture recognition algorithms that are easy to implement and to incorporate into mobile applications, while at the same time retaining an acceptable gesture recognition accuracy. By doing this, we hope to help practitioners and researchers to incorporate gesture recognition into their applications at an early stage of in the development of their projects, promoting the use of gesture-based input in mobile user interfaces. This effort is starting to bear fruit, as recent publications in human-computer-interaction, e.g., (Alexander et al., 2012), have made use of *Protractor3D* (Section 5.3). In addition, there have been numerous requests from industry expressing interest in using the open-source versions of the *$3 Gesture Recognizer* and *Protractor3D* (Kratz, 2012a,b)[W] in their projects.

The *$3 Gesture Recognizer* (Section 5.2) is an easy-to-implement template-based gesture recognizer we developed by extending earlier work by (Wobbrock and Wilson, 2007). The *$3 Gesture Recognizer* needs only few training gestures, can be incorporated rapidly into mobile applications and has sufficient accuracy for prototyping applications.

Improving upon the *$3 Gesture Recognizer* Protractor3D (Section 5.3) significantly increases gesture recognition accuracy by solving the template–gesture rotation problem (Section 5.3.2.1), while at the same time retaining the desirable properties of its predecessor.

gestures for authentication can be easier to memorize than PINs and faster to execute on mobile devices

We also examine a novel application area for motion gestures, *gesture-based authentication* (GBA) (Section 5.4). Accelerometers and gyroscopes in modern mobile device have sufficient fidelity and update rates to be used for authentication purposes. The sensor data provides features that can be used to distinguish between motions of individual users, because users perform motions differently due to physiological differences. We aim to prove the feasibility of such a system and to analyze its resistance to forgery attempts. To do this, we analyze three different forgery scenarios. We demonstrate that GBA is reasonably secure even for attacks with direct visual information of the performed gesture. Since the false acceptance rate lies below 5% when the GBA system is set to be reasonably usable (in terms of the tradeoff between false acceptance and false rejection rates), we recommend using GBA for rapid "low-risk" authentication tasks, e.g., for posting updates to social network sites, where PIN-entry would be cumbersome. Future refinement of the algorithms will most likely lead to a higher level of security. Embedding GBA capabilities on a low level of the hardware, i.e., the SIM card of a mobile device, would open very interesting possibilities. We have laid the foundations for such work to proceed.

Because a growing number of mobile devices is being equipped with a gyroscope as well as an accelerometer, we investigate how to make the additional data of the gyroscope available to the gesture recognition algorithms. In Section 5.5 we present an algorithm for combining accelerometer and gyroscope data that can be used with algorithms such as *Protractor3D*, that only support three dimensional input data. We present the results of a study showing the effect of combining accelerometer and gyroscope data for *Protractor3D* and other popular gesture recognition algorithms. Our results indicate that combining accelerometer and gyroscope data increases the gesture recognition F_1 score 2%–4%, depending on the gesture algorithm employed.

8.1.4 Empowering End Users

Mayhem allows end users to build custom user interfaces controlling complex functionality

Creating gesture-based user interfaces is at present only possible for those users who have programming skills. To make the scripting of complex functionality, including gesture-based input, accessible to end users, we developed *Mayhem* (Section 6.1). *Mayhem* packages modules that implement complex functionality, and makes these functions available in a scriptable environment targeted at end users. *Mayhem* has recently been released as an open-source project and is currently in the process of building up a user community.

8.2 Recapitulation and Contextualization of Contributions and Results

In this dissertation we contribute a range of user interface concepts for sensor-based interaction aimed at enhancing the scope of mobile interaction input possibilities for current and future mobile devices. We have shown that the expressiveness of mobile input can be significantly increased when making non-trivial use of the devices' onboard sensors. We have made contributions in three promising areas of sensor-based input for mobile devices: *continuous interaction*, *around-device interaction* and *motion gestures*. These contributions presented in this dissertation cumulate in an in-depth view on the possibilities of sensor-based mobile user interfaces and form a prediction of the development of mobile user interfaces in the coming years.

In the following, we list all individual contributions and then present a contextualization of them within the SIGCHI HCI curriculum system (Hewett et al., 2009)[W].

1. Continuous Interaction and State-Space Systems:

   (a) *Semi-Automatic Zooming* (SAZ): mobile tilt-based map navigation interface based on Speed-Dependent Automatic Zooming (SDAZ) that has the option of a manual override of the base zoom level.

   develops tilt-based map navigation interface as exploration of non-trivial usage of accelerometers on mobile devices

   Extension of a state-space model to support 2D map scrolling with automatic zooming and tilt input. Helps overcome several limitations of SDAZ, shows significant decrease in task completion time vs. SDAZ.

   (b) *Flick-and-Zoom* (FAZ): mobile map interface for map navigation based on SDAZ with flick-based input.

   Insights into mappings for flick-based input for map navigation. Creation of a state-space model for automatic zooming and flick input. Presents preliminary findings on user performance with FAZ.

2. Around-Device Interaction (ADI)

   (a) *Introduction to the concepts of ADI. Characterization of the ADI design space.*

   (b) *HoverFlow*: prototype ADI interface supporting coarse hand gestures.

   develops concepts for future mobile user interfaces based on sensor technologies yet to be incorporated into mobile devices

   Proof of concept implementation for ADI. Demonstrates sufficiently accurate around-device gesture recognition with a simple set of sensors.

(c) *PalmSpace*: precise around-device gesture-based input based on a mobile depth camera.

Development of a gesture-based input technique for 3D rotation. User study shows improved task time performance of around-device gestures in comparison to existing touch screen 3D rotation techniques.

(d) *iPhone Sandwich*: a mobile dual-sided multi-touch device with pressure sensing.

Development of the iPhone Sandwich prototype. Conception of application scenarios for dual-sided pressure-based input. Gains new insights into mapping of pressure sensor data to user interface inputs; corrects misconceptions about this in the literature. Evaluates the influence of different grip poses on the effectiveness of pressure input. Development of a dual-sided virtual trackball that enables occlusion-free 3D object rotation.

provides algorithms that simplify the integration of gesture recognition into mobile applications

3. Motion Gestures

(a) *$3 Gesture Recognizer*: a simple-to-implement motion gesture recognizer for rapid prototyping applications.

Extension of existing recognition technique to cope with 3D input data. Evaluation of performance in a user study. Technique is easier to implement and requires less training data than previous gesture recognizers, while retaining a comparable recognition accuracy.

(b) *Protractor 3D*: improvement upon the $3 Gesture Recognizer that is invariant to rotational differences between gesture inputs and model templates.

Application of a closed-form solution to the gesture–template rotation problem, resulting in significant accuracy gains of the $3 Gesture Recognizer*, and keeping its beneficial properties.*

(c) *Gesture-Based Authentication* (GBA) : user authentication on mobile devices through motion gestures.

Shows that GBA is feasible on devices equipped with accelerometers and gyroscopes. Shows that GBA is resistant towards direct visual attacks. Demonstrates that such a system is usable and accepted by users.

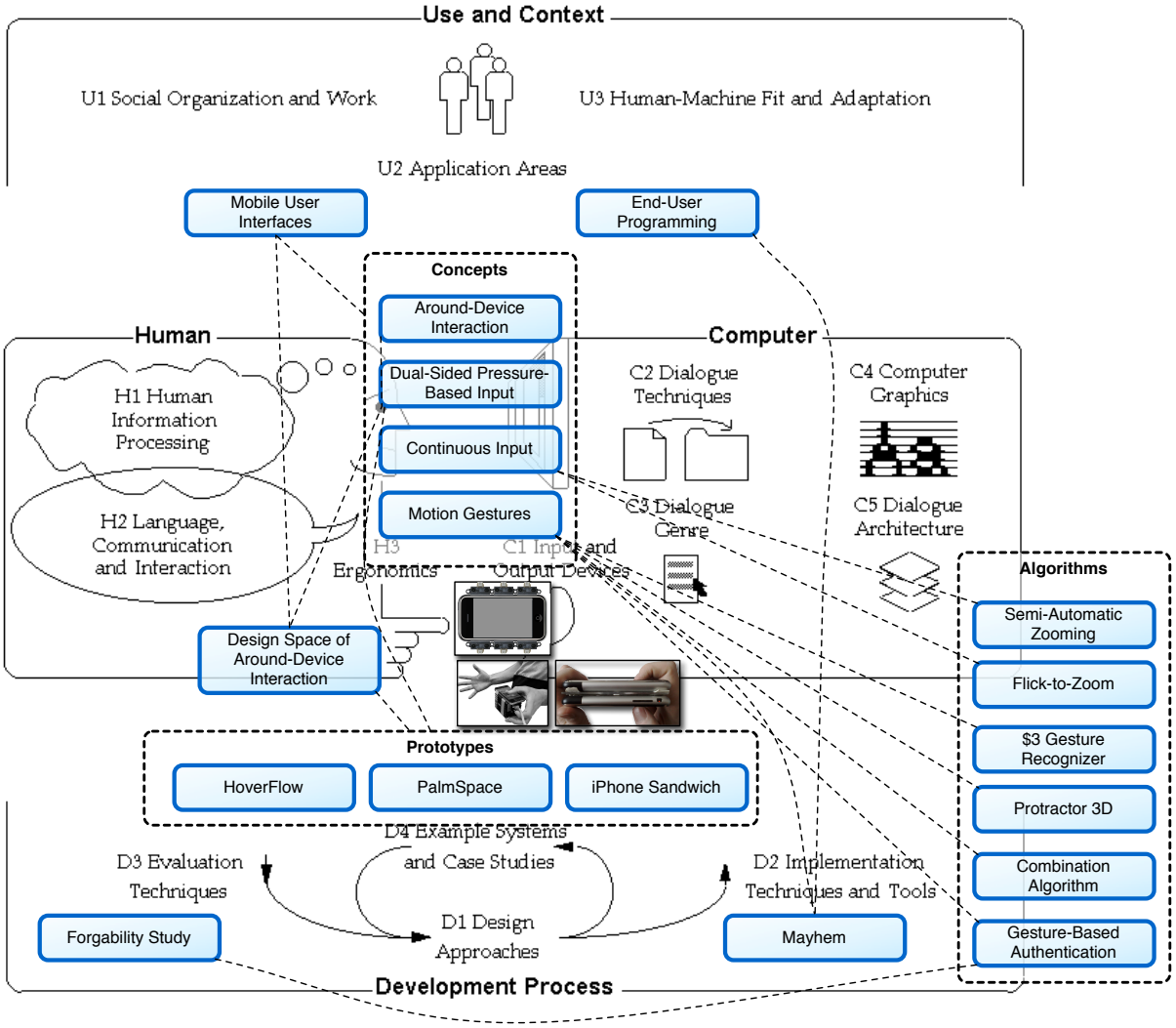


Figure 8.1: The contributions of this dissertation mapped over a graphical representation of the ACM's classification system for HCI. The main classes of contributions and their interrelation is shown.

(d) *Combining Accelerometer and Gyroscope Data for Motion Gesture Recognition*: analysis of the effects of combining accelerometer and gyroscope data

Introduces an algorithm to combine accelerometer and gyroscope data for use with algorithms such as Protractor3D. Evaluates effect combining these two data types on a large data set with several gesture recognition algorithms.

4. Empowering the End User

bridges the gap towards end users, giving them access to some of the advanced concepts presented in this dissertation

(a) *Mayhem* : a programming environment for end users.

Demonstrates how Mayhem enables end users without programming skills to use advanced concepts such as gesture recognition in their

own applications, empowering them to create complex interactions by themselves.

The majority of the contributions in this dissertation have been published in the proceedings of peer-reviewed academic conferences: (?Kratz and Rohs, 2010b,a, 2011; Kratz et al., 2010a, 2012a; Stewart et al., 2010).

In order for the reader to better understand the context of the contributions of this dissertation, we map them over a graphical representation of the ACM's classification system for HCI in Figure 8.1. The contributions are grouped into three main classes: *concepts, algorithms* and *prototypes*. Moreover, the figure highlights the relationships between the individual contributions and concepts.

8.3 Closing Remarks

The use of mobile devices is currently growing rapidly throughout the world, especially in developing countries. At the same time, the number of tasks that users perform on mobile devices is increasing steadily. Because of this shift of focus away from stationary computing, mobile interaction has become an important field within HCI. The contributions in this dissertation provide developers of mobile user interface with a number of novel interface concepts, that will allow them to make better use of the sensory capabilities of mobile devices. Furthermore, the contributions can guide mobile phone manufacturers in choosing sensor technologies to embed in future mobile devices, with the objective of increasing those devices' usability, usefulness and joy of use.

Sven Kratz
Sunnyvale,CA, January 11, 2013

Appendix A

Contents of CD-ROM

The CD-ROM accompanying to this dissertation contains material and media accompanying this work. The root directory of the CD-ROM contains four folders with the following contents:

- **Papers:** The papers containing the most of the relevant contributions to this dissertation.
- **Source Code and Software:** The source code for the $3 Gesture Recognizer and Protractor 3D as well as the Gesture Recombination Algorithm and a Python implementation of regularized logistic regression. Furthermore, the current Mayhem installer, as well as the UDP listener event used in the example in Section 6.2 is included.
- **Study Data:** Some raw data and evaluation scripts the user studies discussed in this work.
- **Web References:** Snapshots of online references at time of access.
- **Videos:** Demonstration videos of HoverFlow, PalmSpace and the iPhone Sandwich.

Bibliography

Waleed H. Abdulla, Davd Chow, and Gary Sin. Cross-words reference template for DTW-based speech recognition systems. In *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*, volume 4, pages 1576–1579. IEEE, 2003.

Jason Alexander, Teng Han, William Judd, Pourang Irani, and Sriram Subramanian. Putting your best foot forward: investigating real-world mappings for foot-based gestures. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 1229–1238, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4.

Dzimitry Aliakseyeu, Pourang Irani, Aandres Lucero, and Sriram Subramiam. Multi-flick: an evaluation of flick-based scrolling techniques for pen interfaces. In *Proceedings of ACM CHI 2008 Conference on Human Factors in Computing Systems*, pages 1689–1698, Florence, Italy, 2008. ACM Press New York, NY, USA.

Lisa Anthony and Jacob O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, pages 245–252. Canadian Information Processing Society, 2010.

Caroline Appert and Jean-Daniel Fekete. OrthoZoom scroller: 1D multi-scale navigation. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 21–30. ACM, 2006.

Daniel Ashbrook. MAGIC: a motion gesture design tool. *Proc. CHI 2010*, pages 2159–2168, 2010a.

Daniel Ashbrook. *Enabling Mobile Microinteractions*. PhD thesis, Georgia Institute of Technology, 2010b.

Daniel Ashbrook, Patrick Baudisch, and Sean White. Nenya: Subtle and eyes-free mobile input with a magnetically-tracked finger ring. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2043–2046. ACM, 2011.

Rafael Ballagas, Michael Rohs, Jennifer G. Sheridan, and Jan Borchers. Sweep and Point & Shoot: Phonecam-based interactions for large public displays. In *CHI '05: Extended abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1200–1203, New York, NY, USA, 2005b. ACM Press. ISBN 1-59593-002-7.

Rafael .A. Ballagas, Sven Kratz, Jan Borchers, Eugen Yu, Steffen P. Walz, Claudia .O. Fuhr, Ludger Hovestadt, and Martin J. Tann. REXplorer: a mobile, pervasive spell-casting game for tourists. In *CHI'07 extended abstracts on Human factors in computing systems*, pages 1929–1934. ACM, 2007.

Lucas Ballard, Daniel Lopresti, and Fabian Monrose. Forgery quality and its implications for behavioral biometric security. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(5):1107–1118, 2007.

Olivier Bau. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proc. UIST 2008*, pages 37–46, Monterey, CA, USA, 2008. ACM Press New York, NY, USA.

Patrick Baudisch and Gerry Chu. Back-of-device interaction allows creating very small touch devices. In *Proc. of CHI '09*, pages 1923–1932, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7.

Patrick Baudisch and Ruth Rosenholtz. Halo: A Technique for Visualizing Off-Screen Objects. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 481–488, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-630-7.

Hrvoje Benko, Andrew D. Wilson, and Patrick Baudisch. Precise Selection Techniques for Multi-Touch Screens. In *Proceedings of ACM CHI 2006 (CHI'06: Human Factors in Computing Systems)*, pages 1263—-1272, Montreal, Canada, April 2006.

Sebastian Boring, Marko Jurmu, and Andreas Butz. Scroll, tilt or move it: using mobile phones to continuously control pointers on large public displays. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group*, pages 161–168. ACM Press New York, NY, USA, 2009.

Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. Touch projector: mobile interaction through video. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2287–2296. ACM, 2010.

Stephen Brewster, Faraz Chohan, and Lorna Brown. Tactile feedback for mobile interactions. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 159, 2007b.

Ivo Brodien. Semi-Automatic Zooming: Eine Erweiterung des Speed-Dependent Automatic Zooming zur Kartennavigation für mobile Endgeräte mit Beschleunigungssensor – Entwurf, Implementierung und Evaluierung (Diplom Thesis). *Quality and Usability Group, Deutsche Telekom Laboratories, TU Berlin*, 2009.

Thorsten Büring, Jens Gerken, and Harald Reiterer. Zoom Interaction Design for Pen-Operated Portable Devices. *International Journal of Human-Computer Studies*, 66(8):605–627, 2008. ISSN 1071-5819.

Alex Butler, Shahram Izadi, and Steve Hodges. SideSight: multi-touch interaction around small devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 201–204. ACM, 2008b.

William Buxton. Chunking and phrasing and the design of human-computer dialogues. In *Proceedings of the IFIP World Computer Congress*, number 1986, pages 475–480. Citeseer, 1986.

Xiang Cao, Jackie J. Li, and Ravin Balakrishnan. Peephole pointing: modeling acquisition of dynamically revealed targets. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1699–1708. ACM, 2008.

Alvaro Cassinelli, Stephane Perrin, and Masatoshi Ishikawa. Smart laser-scanner for 3D human-machine interface. In *CHI'05 extended abstracts*, pages 1138–1139, 2005.

Jessica R. Cauchard, Mike Fraser, Teng Han, and Sriram Subramanian. Steerable projection: exploring alignment in interactive mobile displays. *Personal and Ubiquitous Computing*, pages 1–11, 2011.

Jared Cechanowicz, Pourang Irani, and Sriram Subramanian. Augmenting the mouse with pressure sensitive input. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 1385, 2007a.

Michael Chen, Joy S. Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. In *Proc. of SIGGRAPH '88*, pages 121–129, New York, NY, USA, 1988. ACM. ISBN 0-89791-275-6.

Sung-Jung Cho, Changkyu Choi, Younghoon Sung, Kwanghyeon Lee, Yeun-Bae Kim, and Roderick Murray-Smith. Dynamics of tilt-based browsing on mobile devices. *CHI '07 extended abstracts on Human factors in computing systems - CHI '07*, page 1947, 2007.

Andy Cockburn and Joshua Savage. Comparing speed-dependent automatic zooming with traditional scroll, pan and zoom methods. In

Bath, UK: People and Computers XVII: British Computer Society Conference on Human Computer Interaction, pages 87–102, 2003.

Andy Cockburn, Joshua Savage, and Andrew Wallace. Tuning and testing scrolling interfaces that automatically zoom. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 71–80, 2005b.

Gabe Cohn, Dan Morris, and Shwetak N. Patel. Your noise is my command: Sensing gestures using the body as an antenna. In *Proc. CHI 2011*, pages 791–800, 2011.

Gabe Cohn, Dan Morris, and Shwetak N. Patel. Humantenna: Using the Body as an Antenna for Real-Time Whole-Body Interaction. In *Proc. CHI 2012*, Austin, TX, USA, 2012. ACM.

Lorrie F. Cranor and Simson Garfinkel. *Security and usability: Designing secure systems that people can use*. O'Reilly Media, Inc., 2005. ISBN 978-0596008277.

Fabrice Decle and Martin Hachet. A study of direct versus planned 3D camera manipulation on touch-based mobile phones. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, page 32. ACM, 2009.

Elmo M.A. Diederiks, Erwin R. Meinders, Edwin Van Lier, Ralph H. Peters, Josephus H. Eggen, and Jasper I. Van Kuijk. *Method of and system for controlling an ambient light and lighting unit*. US Patent App. 10/519,066, June 2003.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2 edition, October 2000. ISBN 0471056693.

Florian Echtler. Tracking mobile phones on interactive tabletops. . *Workshop on Mobile and Embedded Interactive*, pages 388–391, 2008.

Parisa Eslambolchilar and Roderick Murray-smith. Tilt-based automatic zooming and scaling in mobile devices - a state-space implementation. In *Proceedings of Mobile HCI 2004*, pages 120–131. Springer, 2004.

Parisa Eslambolchilar and Roderick Murray-Smith. Control centric approach in designing scrolling and zooming user interfaces. *International Journal of Human-Computer Studies*, 66(12):838–856, December 2008c. ISSN 10715819.

Georg Essl. UrMus–an environment for mobile instrument design and performance. In *Proceedings of the International Computer Music Conference*, pages 76–81, 2010.

Georg Essl and Michael Rohs. The Design Space of Sensing-Based Interaction for Mobile Music Performance. In *Proceedings of the 3rd International Workshop on Pervasive Mobile Interaction Devices (PERMID)*, Toronto, Ontario, Canada, May 2007.

Georg Essl, Michael Rohs, and Sven Kratz. Demo: Squeezing the Sandwich: A Mobile Pressure-Sensitive Two-Sided Multi-Touch Prototype. In *Demonstration at the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST), Victoria, BC, Canada*, Victoria, BC, Canada,, October 2009.

Georg Essl, Sven Kratz, and Michael Rohs. Electronic device and method for controlling an electronic device, EP000002282256A1, February 2011.

Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM, 2010.

Elisabeth Farella, Sile O'Modhrain, Luca Benini, and Bruno Ricco. Gesture signature for ambient intelligence applications: a feasibility study. *Lecture Notes in Computer Science*, 3968:288, 2006.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. ISSN 0001-0782.

Roger Fletcher. *Practical methods of optimization, volume 1*. Wiley, 1987. ISBN 978-0471277118.

Sarah D. Goodman. Social Media: The Use of Facebook and Twitter to Impact Political Unrest in the Middle East through the Power of Collaboration. Technical report, California Polytechnic State University, 2011.

Nancy C. Goodwin. Functionality and usability. *Communications of the ACM*, 30(3):229–233, 1987.

Ettenne Grandjean. *Fitting the task to the man: a textbook of occupational ergonomics*. Taylor & Francis/Hemisphere, 1989.

Thomas Gruber, Adam John Cheyer, and Apple. Intelligent Automated Assistant, WIPO Patent Application Nr WO 2011/088053 A2, 2011.

Dennis Guse. Gesture-Based User Authentication on Mobile Devices using Accelerometer and Gyroscope (Master's Thesis). *Quality and Usability Group, Deutsche Telekom Laboratories, TU Berlin*, 2011.

Martin Hachet, Joachim Pouderoux, and Pascal Guitton. A camera-based interface for interaction with mobile handheld computers. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 65–72, New York, NY, USA, 2005a.

Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24. ACM Press/Addison-Wesley Publishing Co., 1998. ISBN 0-201-30987-4.

Chris Harrison and Scott Hudson. Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. *Proceedings of the UIST*, pages 205–208, 2008.

Chris Harrison and Scott E. Hudson. Abracadabra: wireless, high-precision, and unpowered finger input for very small mobile devices. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 121–124. ACM, 2009b.

Chris Harrison, Desney Tan, and Dan Morris. Skinput: Appropriating the body as an input surface. In *Proceedings of the 28th international conference on Human Factors in Computing Systems*, CHI '10, pages 453–462, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9.

Chris Harrison, Hrvoje Benko, and Andrew D. Wilson. OmniTouch: wearable multitouch interaction everywhere. In *Proc. UIST 2011*, pages 441–450, Santa Barbara, CA, USA, 2011. ACM Press New York, NY, USA.

S G Hart and L E Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Human mental workload*, 1:139–183, 1988a.

Knud Henriksen, Jon Sporring, and Kasper Hornbaek. Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics*, 10 (2):206–216, 2004.

Enrico Herrmann, Andrey Makrushin, Jana Dittmann, Claus Vielhauer, Mirko Langnickel, and Christian Kraetzer. Hand-movement-based in-vehicle driver/front-seat passenger discrimination for centre console controls. *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 7532:23, 2010.

Ken Hinckley and Hyunyoung Song. Sensor synaesthesia: touch in motion, and motion in touch. *Proc. CHI 2011*, pages 801–810, 2011.

Michael Hoffman and P Varcholik. Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices.

In *IEEE Virtual Reality Conference (VR)*, pages 59–66, Waltham, Massachusetts, USA, 2010. IEEE.

Eve Hoggan, Stephen A. Brewster, and Jody Johnston. Investigating the Effectiveness of Tactile Feedback for Mobile Touchscreens. In *Proceedings of ACM CHI 2008 Conference on Human Factors in Computing Systems*, pages 1573–1582, April 2008.

Jason I. Hong and James A. Landay. SATIN: a toolkit for informal ink-based applications. In *Proc. UIST 2000*, page 12, San Diego, CA, United States, 2007. ACM Press New York, NY, USA.

Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, 4(4):629–642, 1987.

Paul Horowitz and Winfield Hill. *The Art of Electronics*. Cambridge University Press, Cambridge, 2nd edition, 1989.

Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. *Human-Computer Interaction*, 1(4):311–338, 1985. ISSN 0737-0024.

Du Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

Takeo Igarashi. Speed-dependent automatic zooming for browsing large documents. *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 139–148, 2000.

Interlink Electrionics. FSR—force sensing resistor integration guide and evaluation parts catalog. Version 1.0, 90-45632 Rev. D, Available at Interlink Electronics, `http://www.interlinkelectronics.com`.

Walter Isaacson. *Steve Jobs*. Simon & Schuster, first edition, October 2011. ISBN 1451648537.

ISO/IEC. Ergonomic requirements for office work with visual display terminals (VDTs) – Requirements for non-keyboard input devices. ISO/IEC 9241-9, 2000a.

ISO/IEC. 18004:2000. Information technology – Automatic identification and data capture techniques – Bar code symbology – QR Code, 2000b.

ISO/IEC. 15420:2000. Information technology - Automatic identification and data capture techniques - Bar code symbology specification - EAN/UPC, 2000c.

Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975.

Robert J. K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin T. Solovey, and Jamie Zigelbaum. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceedings of CHI 2008*, pages 201–210, Florence, Italy, 2008b. ACM. ISBN 978-1-60558-011-1.

Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow. Implicit authentication for mobile devices. In *Proceedings of the 4th USENIX conference on Hot topics in security*, pages 9–9. USENIX Association, 2009.

I. T. Jolliffe. *Principal component analysis*, volume 2. Springer US, 2002. ISBN 978-0387954424.

Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 139–146. ACM, 2007.

Ronald Kainda, Ivan Flechais, and William A. Roscoe. Security and usability: Analysis and evaluation. In *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*, pages 275–282. IEEE, 2010.

Sanna Kallio, Juha Kela, Jani Mäntyjärvi, and Johan Plomp. Visualization of hand gestures for pervasive computing environments. *Proceedings of the working conference on Advanced Visual Interfaces (AVI)*, page 483, 2006.

Rudy E. Kálmán. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(Series D):35–45, 1960.

Alan C. Kay. A Personal Computer for Children of All Ages. *Byte*, 1972b.

Juha Kela, Panu Korpipää, Jani Mäntyjärvi, Sanna Kallio, Giuseppe Savino, Luca Jozzo, and Sergio Di Marca. Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing*, 10(5):285–299, 2006.

Eamonn Keogh and Chotirat A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

Hamed Ketabdar, Mehran Roshandel, and Kamer A. Yüksel. Towards using embedded magnetic field sensor for around mobile device 3D interaction. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 153–156. ACM, 2010a.

Hamed Ketabdar, Kamer A. Yüksel, and Mehran Roshandel. MagiTact: Interaction with mobile devices based on compass (magnetic) sensor. In *Proceeding of the 14th international conference on Intelligent User Interfaces*, pages 413–414. ACM, 2010b.

Niklas Kirschnick, Sven Kratz, and Sebastian Möller. Poster: An Improved Approach to Gesture-Based Authentication for Mobile Devices. In *SOUPS '10 Symposium on Usable Privacy and Security. ACM.*, 2010.

Yoshifumi Kitamura, Takashige Konishi, and Toshihiro Masaki. IllusionHole: a stereoscopic display for multiple observers. *Proceedings of SPIE*, 4297:360, 2001.

Andrew J. Ko, Brad A. Myers, and Htet H. Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 199–206. IEEE, 2004.

Eva Kollorz, Jochen Penne, Joachim Hornegger, and Alexander Barke. Gesture recognition with a Time-Of-Flight camera. *International Journal of Intelligent Systems Technologies and Applications*, 5(3):334–343, 2008.

Sven Kratz. *A Camera-Based Gesture Recognition System for Mobile Devices*. Diplom thesis, RWTH Aachen University, 2007b.

Sven Kratz and Michael Rohs. Unobtrusive Tabletops : Linking Personal Devices with Regular Tables. In *Multi-Touch and Surface Computing Workshop at CHI 2009*, pages 1–6, Boston, MA, USA, 2009b. ISBN 9781605582474.

Sven Kratz and Michael Rohs. Extending the virtual trackball metaphor to rear touch input. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 111–114, Waltham, Massachusetts, USA, 2010a. IEEE, IEE.

Sven Kratz and Michael Rohs. A $3 Gesture Recognizer - Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors. In *Proceedings of the 14th international conference on Intelligent user interfaces.*, Hong Kong, China, February 2010b.

Sven Kratz and Michael Rohs. Protractor3D : A Closed-Form Solution to Rotation-Invariant 3D Gestures. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2011)*, Palo Alto, CA, USA, 2011. ACM. ISBN 9781450304191.

Sven Kratz, Ivo Brodien, and Michael Rohs. Semi-Automatic Zooming for Mobile Map Navigation. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services MobileHCI 2010*, pages 63—-72, Lisbon, 2010a. ACM Press New York, NY, USA. ISBN 9781605588353.

Sven Kratz, Fabian Hemmert, and Michael Rohs. Natural User Interfaces in Mobile Interaction. In *Natural User Interfaces: the prospect and challenge of touch and gestural computing (Workshop at CHI 2010)*, Atlanta, Georgia, USA, 2010b.

Sven Kratz, Dennis Guse, Michael Rohs, Jörg Müller, Gilles Bailly, and Michael Nischt. PalmSpace: Continuous Around-Device Gestures vs. Multitouch for 3D Rotation Tasks on Mobile Devices. In *Proc. AVI 2012*, Capri Island, Italy, 2012a. ACM Press New York, NY, USA.

Sven Kratz, Michael Rohs, Felix Reitberger, and Jörg Moldenhauer. Attjector: an Attention-Following Wearable Projector. Kinect Workshop at Pervasive 2012, Newcastle, United Kingdom, 2012b.

Christian Kray, Michael Rohs, Jonathan Hook, and Sven Kratz. Group coordination and negotiation through spatial proximity regions around mobile devices on augmented tabletops. In *3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008.*, pages 1–8. IEEE, 2008.

Valerie Kroner. Flick-and-Zoom: Improving the Usability of Speed-Dependent Automatic Zooming for Map Navigation on Mobile Devices (Project Thesis). *Chair of Media Informatics, University of Munich*, 2011.

Christine Kuehnel, Tilo Westermann, Fabian Hemmert, Sven Kratz, Alexander Müller, and Sebastian Moeller. I'm home: defining and evaluating a gesture set for smar-home control. *International Journal of Human-Computer Studies*, 69(11):1071–5819, 2011.

M. J. Large, Tim Large, and Adrian R. L. Travis. Parallel Optics in Waveguide Displays: A Flat Panel Autostereoscopic Display. *Journal of Display Technology*, 6(10):431–437, 2010.

Johnny C. Lee. Hacking the Nintendo Wii Remote. *IEEE Pervasive Computing*, pages 39–45, 2008.

Johnny C. Lee, Paul H. Dietz, Darren Leigh, William S. Yerazunis, and Scott E. Hudson. Haptic Pen: A Tactile Feedback Stylus for Touch Screens. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 291–294, New York, NY, USA, 2004. ACM. ISBN 1-58113-957-8.

Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient L1 Regularized Logistic Regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 401. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Taehee Lee and Tobias Höllerer. Handy AR: Markerless inspection of augmented reality objects using fingertip tracking. In *11th IEE International Symposium on Wearable Computers*, pages 83–90. Wearable Computers, 2007 11th IEEE International Symposium, 2007.

Yang Li. Protractor: a fast and accurate gesture recognizer. In *Proceedings of the 28th international conference on Human factors in computing systems (CHI)*, pages 2169–2172, Atlanta, Georgia, USA, 2010b. ACM.

Chunyuan Liao, François Guimbretière, and Corinna E. Loeckenhoff. Pen-Top Feedback for Paper-Based Interfaces. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 201–210, New York, NY, USA, 2006. ACM. ISBN 1-59593-313-1.

Jiayang Liu, Lin Zhong, and Jehan Wickramasuriya. User evaluation of lightweight user authentication with a single tri-axis accelerometer. *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–10, 2009a.

Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 2009b.

A. Chris Long Jr., James A. Landay, Lawrence A. Rowe, and Joseph Michiels. Visual similarity of pen gestures. In *Proc. CHI 2000*, pages 360–367, The Hague, Netherlands, 2000. ACM.

Joseph Luk, Jérôme Pasquero, Shannon Little, Karon MacLean, Vincent Lévesque, and Vincent Hayward. A Role for Haptics in Mobile Interaction: Initial Design Using a Handheld Tactile Display Prototype. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, pages 171–180, Montreal, Canada, April 2006.

Arnold M. Lund. The need for a standardized set of usability metrics. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, volume 42, pages 688–691. Human Factors and Ergonomics Society, 1998.

Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman & Hall/CRC, 2009. ISBN 978-1420067187.

Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. *Pervasive computing*, pages 144–161, 2007.

David C. McCallum, Edward Mak, Pourang Irani, and Sriram Subramanian. PressureText: Pressure Input for Mobile Phone Text Entry.

In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4519–4524, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-247-4.

Sachi Mizobuchi, Shinya Terasaki, Turo Keski-Jaskari, Jari Nousiainen, Matti Ryynanen, and Miika Silfverberg. Making an Impression: Force-Controlled Pen Input for Handheld Devices. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1661–1664, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7.

Roderick Murray-Smith, John Williamson, Stephen Hughes, and Torben Quaade. Stane: synthesized surfaces for tactile input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1299–1302, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1.

NASA. *Man-Systems Integration Standards*. Rev.˜B, 1995.

National Economics And Statistics Administration and National Telecommunications and Information Administration. Exploring the Digital Nation: Computer and Internet Use at Home. Technical report, U.S. Department of Commerce, 2011.

Hartmut Neven Sr and Hartmut Neven. Image-based search engine for mobile phones with camera, US Patent 7,565,139, July 2009.

Tao Ni and Patrick Baudisch. Disappearing mobile devices. *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 101–110, 2009.

Donald A. Norman. *The Design of Everyday Things*. Basic Books, 2002. ISBN 0465067107.

Ian Oakley and Sile O'Modhrain. Tilt to Scroll: Evaluating a Motion Based Vibrotactile Mobile Interface. In *IEEE Proceedings of the World Haptics Conference*, Pisa, Italy, 2005.

Dynal Patel, Gary Marsden, Steve Jones, and Matt Jones. An evaluation of techniques for browsing photograph collections on small displays. *Mobile Human-Computer Interaction–MobileHCI 2004*, pages 261–311, 2004a.

Shwetak N. Patel, Jeffrey S. Pierce, and Gregory D. Abowd. A gesture-based authentication scheme for untrusted public terminals. In *Proc. UIST 2004*, pages 157–160, 2004b.

Brandon Paulson, Pankaj Rajan, Pedro Davalos, Ricardo Gutierrez-Osuna, and Tracy Hammond. What!□ no Rubine features?: using geometric-based features to produce normalized confidence values

for sketch recognition. In Beryl Plimmer and Tracy Hammond, editors, *VL/HCC Workshop: Sketch Tools for Diagramming*, pages 57–63, Herrsching am Ammersee, Germany, 2008.

Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (7):677–695, 1997.

Herbert Prähofer, Dominik Hurnaus, Roland Schatz, Christian Wirth, and Hanspeter Mössenböck. Software support for building end-user programming environments in the automation domain. In *Proceedings of the 4th international workshop on End-user software engineering*, pages 76–80, 2008.

Qian Qin, Michael Rohs, and Sven Kratz. Dynamic ambient lighting for mobile devices. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, pages 51–52, Santa Barbara, CA, USA, 2011. ACM.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, 77 (2):267—-296, 1990.

Mahfuz Rahman, Sean Gustafson, Pourang Irani, and Sriram Subramanian. Tilt techniques: investigating the dexterity of wrist-based input. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1943–1952. ACM, 2009.

T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

Gonzalo Ramos and Ravin Balakrishnan. {Zliding}: Fluid Zooming and Sliding for High Precision Parameter Manipulation. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 143–152, New York, NY, USA, 2005. ACM. ISBN 1-59593-271-2.

Gonzalo Ramos, Matthew Boulos, and Ravin Balakrishnan. Pressure widgets. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 487–494, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8.

Gonzalo a. Ramos and Ravin Balakrishnan. Pressure marks. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 1375, 2007a.

Jef Raskin. *The humane interface: New directions for designing interactive systems*. ACM Press/Addison-Wesley Publishing Co., March 2000. ISBN 0-201-37937-6.

Matthias Rath. Auditory velocity information in a balancing task. In *Proc. of the 13th International Conference on Auditory Display (ICAD)*, volume 7, pages 372–379, Montreal, Canada, 2007.

Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *Proc. National Conference on Artificial Intelligence 2005*, volume 20, page 1541, Menlo Park, CA, 2005. MIT Press.

Milan Redzic, Ciaran O. Conaire, Noel E. O'Connor, and Conor C. Brennan. Multimodal Identification of Journeys. In *2010 Workshop on Database and Expert Systems Applications (DEXA)*, pages 283–287. IEEE, 2010.

Jun Rekimoto. Tilting operations for small screen interfaces. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 167–168. ACM Press, 1996b. ISBN 0-89791-798-7.

Jun Rekimoto and Carsten Schwesig. {PreSenseII}: Bi-Directional Touch and Pressure Sensing Interactions with Tactile Feedback. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1253–1258, New York, NY, USA, 2006. ACM. ISBN 1-59593-298-4.

Xiangshi Ren, Jinbin Ying, Shengdong Zhao, and Yang Li. The Adaptive Hybrid Cursor: A Pressure-based Target Selection Technique for Pen-based Interfaces. In *Proceedings of the Interact 2007 Conference (INTERACT)*, pages 310–323, 2007.

Michael Rohs. Real-World Interaction with Camera Phones. In *International Symposium on Ubiquitous Computing Systems (UCS 2004), Tokyo, Japan, http://www.vs.inf.ethz.ch/publ/papers/rohs2004-visualcodes.pdf Also published under Revised Selected Papers, pp. 74-89, LNCS 3598, Springer,*.

Michael Rohs and Antti Oulasvirta. Target acquisition with camera phones when used as magic lenses. In *Proc. CHI 2008*, pages 1409–1418, Florence, Italy, 2008. ACM, ACM Press New York, NY, USA.

Ilya Rosenberg and Ken Perlin. The UnMousePad: an interpolating multi-touch force-sensing input pad. In *ACM Transactions on Graphics (TOG)*, volume 28, page 65. ACM, 2009.

Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25 (4):329—-337, 1991.

Jaime Ruiz and Yang Li. DoubleFlip: a motion gesture delimiter for mobile interaction. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2717–2720. ACM, 2011.

Jaime Ruiz, Yang Li, and Edward Lank. User-defined motion gestures for mobile interaction. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 197–206. ACM, 2011.

Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

Erica Sadun. *iPhone Developer's Cookbook*. Pearson Education, 2008. ISBN 0321555457.

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

Stan Salvador and Philip Chan. FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space. In *KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, 2004.

Lawrence K. Saul and Sam T. Roweis. An introduction to locally linear embedding. *Unpublished article. Available at: http://www. cs. toronto. edu/\~ roweis/lle/publications. html*, 2000.

Abraham Savitzky and Marcel J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964b.

Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a Wii controller. *Proceedings of the 2nd international conference on Tangible and embedded interaction - TEI '08*, page 11, 2008.

Carsten Schwesig, Ivan Poupyrev, and Eijiro Mori. {Gummi}: A Bendable Computer. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 263–270, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8.

James Scott, Lorna M. Brown, and Mike Molloy. Mobile Device Interaction with Force Sensing. In *Pervasive '09: Proceedings of the 7th International Conference on Pervasive Computing*, pages 133–150, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01515-1.

Sharp. Sharp GP2D120 Distance Sensor, 2008.

Erh-li Early Shen, Sung-sheng Daniel Tsai, Hao-hua Chu, Yung-jen Jane Hsu, and Chi-wen Euro Chen. Double-side multi-touch input for mobile devices. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 4339–4344, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-247-4.

Kang Shi, Pourang Irani, Sean Gustafson, and Sriram Subramanian. {PressureFish}: A Method to Improve Control of Discrete Pressure-Based Input. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1295–1298, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1.

Ken Shoemake. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Graphics Interface*, volume 92, pages 151–156, 1992.

Katie Siek, Yvonne Rogers, and Kay H. Connelly. Fat finger worries: How older and younger users physically interact with PDAs. *Human-Computer Interaction-INTERACT 2005*, pages 267–280, 2005.

Carolyn Snyder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, first edition, 2003. ISBN 978-1558608702.

M Srinivasan and J Chen. Human Performance in Controlling Normal Forces of Contact with Rigid Objects. In *Proceedings of Advances in Robotics, Mechatronics, and Haptic Interfaces*, volume 49, pages 119–125, 1993.

Thad Starner, Jake Auxier, Daniel Ashbrook, and Maribeth Gany. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. In *International Symposium on Wearable Computers*, pages 87–94, 2000.

Craig Stewart, Michael Rohs, Georg Essl, and Sven Kratz. Characteristics of Pressure-Based Input for Mobile Devices. In *Proceedings of the 28th international conference on Human factors in computing systems CHI 2010*, pages 801—-810, Atlanta, Georgia, USA, April 2010. ACM Press New York, NY, USA.

Masanori Sugimoto and Keiichi Hiroki. HybridTouch: an intuitive manipulation technique for PDAs using their front and rear surfaces. In *MobileHCI '06: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 137–140, New York, NY, USA, 2006. ACM. ISBN 1-59593-390-5.

Adrian Travis, Tim Large, Neil Emerton, and Steven Bathiche. Collimated light from a waveguide for a display backlight. *Opt. Express*, 17 (22):19714–19719, 2009.

Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

Jake J. van Wijk and Wim A. A. Nuij. A model for smooth viewing and navigation of large 2d information spaces. *Visualization and Computer Graphics, IEEE Transactions on*, 10(4):447–458, 2004.

Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

Daniel Vogel and Patrick Baudisch. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 657–666. ACM, 2007.

Daniel Wagner and Dieter Schmalstieg. Artoolkit on the pocketpc platform. In *IEEE International Augmented Reality Toolkit Workshop*, pages 14–15. IEEE, 2003b.

Andrew Wallace. *The calibration and optimisation of Speed-Dependent automatic zooming*. Honours thesis, University of Canterbury, Christchurch, New Zealand, 2003.

Andrew Wallace, Joshua Savage, and Andy Cockburn. Rapid visual flow: how fast is too fast? *Proceedings of the fifth conference on Australasian user interface-Volume 28*, pages 117–122, 2004.

Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

Daniel Wigdor and Dennis Wixon. *Brave NUI world: designing natural user interfaces for touch and gesture*. Morgan Kaufmann, 2011. ISBN 978-0123822314.

Daniel Wigdor, Darren Leigh, Clifton Forlines, Samuel Shipman, John Barnwell, Ravin Balakrishnan, and Chia Shen. Under the table interaction. *Proc. UIST 2006*, pages 259–268, 2006.

Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Barnwell, and Chia Shen. Lucid touch: a see-through mobile device. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 269–278. ACM, 2007b.

Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Barnwell, and Chia Shen. Lucid touch: a see-through mobile device. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 269–278. ACM, 2007c.

Jacob O. Wobbrock and Andrew D. Wilson. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. UIST 2007*, page 168, Newport, RI, USA, 2007. ACM Press New York, NY, USA.

Jacob O. Wobbrock, Brad A. Myers, and Htet H. Aung. The performance of hand postures in front-and back-of-device interaction for mobile computing. *International Journal of Human-Computer Studies*, 66(12): 857–875, 2008.

Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *Proc. CHI 2009*, pages 1083–1092, Boston, MA, 2009. ACM Press New York, NY, USA.

Larry S. Yaeger, Brandyn J. Webb, and Richard F. Lyon. Combining neural networks and context-driven search for online, printed handwriting recognition in the Newton. *AI Magazine*, 19(1):73, 1998.

Ka-Ping Yee. Peephole Displays: Pen Interaction on Spatially Aware Handheld Computers. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-630-7.

Shengdong Zhao, Pierre Dragicevic, Mark H Chignell, Ravin Balakrishnan, and Patrick Baudisch. earPod: Eyes-free Menu Selection with Touch Input and Reactive Audio Feedback. In *Proceedings of CHI 2007*, pages 1395–1404, 2007.

Web References

Analog Devices Inc. Analog Devices ADXL 330 Data Sheet. http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf, 2012. Accessed: 20.05.2012 (url.ADXL330).

Apple Computer Inc. Event Handling Guide for IOS. https://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/MotionEvents/MotionEvents.html#//apple_ref/doc/uid/TP40009541-CH4-SW1, 2012a. Accessed: 22.02.2012 (url.appleSdkShake).

Apple Computer Inc. Apple script. http://developer.apple.com/applescript/, 2012b. Accessed: 20.01.2012 (url.applescript).

Apple Computer Inc. Mac OS X Automator. http://www.macosxautomation.com/automator/, 2012c. Accessed: 20.01.2012 (url.macosxautomator).

Apple Computer Inc. Quartz Composer. http://developer.apple.com/graphicsimaging/quartzcomposer/, 2012d. Accessed: 19.01.2012 (url.quartzcomposer).

Arduino. Arduino BT (Bluetooth). http://arduino.cc/en/Main/ArduinoBoardBluetooth, 2012. Accessed: 01.03.2012 (url.arduinobt).

Blender Foundation. Blender. http://www.blender.org/, March 2012. Accessed: 17.03.2012 (url.blender).

bulletphysics.org. Bullet Game Physics Simulation. http://bulletphysics.org, 2012. Accessed: 01.05.2012 (url.bullet).

CBS News. Number of Cell Phones Worldwide Hits 4.6B. http://www.cbsnews.com/stories/2010/02/15/business/main6209772.shtml, 2010. Accessed: 18.05.2012 (url.mobilephoneusage).

Cycling74. Max/MSP. http://cycling74.com, 2012. Accessed: 19.01.2012 (url.cycling74).

Paul Dietz, Sven Kratz, Jay Meistich, and Eli White. Mayhem Codeplex site. http://mayhem.codeplex.com, 2012. Accessed: 21.01.2012 (url.mayhemforge).

eyeSight Tech. eyeSight's Advanced Gesture Recognition and Machine Vision Software Based Technology. http://www.eyesight-tech.com/, 2012. Accessed: 12.03.2012 (url.eyesight-tech).

Amy Fowler. A swing architecture overview. http://java.sun.com/products/jfc/tsc/articles/architecture/, 1998. Accessed: 25.04.2012 (url.swing).

Google Inc. MotionEvent – Android Developers. http://developer.android.com/reference/android/view/MotionEvent.html, 2012a. Accessed: 21.04.2012 (url.androidpressure).

Google Inc. Google Goggles. http://www.google.com/mobile/goggles/, 2012b. Accessed: 18.05.2012 (url.googlegoggles).

Havok.com Inc. Havok. http://www.havok.com, 2012. Accessed: 01.05.2012 (url.havok).

Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong, and Verplank. ACM SIGCHI curricula for human-computer interaction, July 2009. Accessed: 15.05.2012 (url.hcicurriculum).

HTC. HTC EVO 3D. http://www.htc.com/www/smartphones/htc-evo-3d/, 2012. Accessed: 18.05.2012 (url.htcevo3d).

Ident Technology AG. Gesture control. http://www.ident-technology.com/technology/gesture-control.html, 2012. Accessd: 05.05.2012 (url.identtech).

Illusion Labs. Labyrinth. http://labyrinth.codify.se/, 2010. Accessed: 10.01.2012 (url.game_labyrinth).

Khronos Group. OpenGL ES. http://www.khronos.org/opengles/, March 2012. Accessed: 17.03.2012 (url.openglES).

Sven Kratz. Three dollar gesture recognizer: an easy-to-use gesture recognizer for motion gestures on smart phones. http://code.google.com/p/three-dollar-gesture-recognizer/, 2012a. Accessed: 15.05.2012 (url.3dollarOSS).

Sven Kratz. Protractor3D open source release. https://bitbucket.org/svenkratz/protractor3d/src, 2012b. Accessed: 15.05.2012 (url.protractor3dOSS).

Layar. Augmented Reality Browser: Layar. http://www.layar.com, 2012. Accessed: 12.03.2012 (url.layar).

LG Corp. LG Optimus 3D Android Mobile Phone - LG Electronics UK.html. http://http://www.lg.com/uk/mobile-phones/all-lg-phones/LG-android-mobile-phone-P920.jsp, 2012. Accessed: 12.03.2012 (url.lgoptimus).

Maciek Drejak Labs AB. Sleep Cycle Alarm Clock. http://www.sleepcycle.com/, 2012. Accessed: 18.05.2012 (url.sleepcycle).

Massachusetts Institute of Technology. MIT App Inventor. http://appinventoredu.mit.edu/, 2012. Accessed: 20.01.2012 (url.appinventor).

Mayhem. Mayhem. http://makemayhem.com/, 2012. Accessed: 21.01.2012 (url.makemayhem).

Mesa Imaging AG. SwissRanger—Driver Downloads. http://mesa-imaging.ch/drivers.php, 2012a. Accessed: 28.05.2012 (url.mesadriver).

Mesa Imaging AG. SwissRanger SR4000 — miniature 3D time–of–flight range camera. http://mesa-imaging.ch/prodview4k.php, 2012b. Accessed: 12.03.2012 (url.sr4000).

Michael Tyson and collaborators. RouteMe Toolkit for IOS. https://github.com/route-me/route-me, 2012. Accessed: 28.01.2012 (url.routeme).

Microsoft Inc. KINECT for Windows. http://www.microsoft.com/en-us/kinectforwindows/, 2012a. Accessed: 01.06.2012 (url.kinect4w).

Microsoft Inc. Kodu. http://research.microsoft.com/en-us/projects/kodu/, 2012b. Accessed: 20.01.2012 (url.kodu).

Microsoft Inc. Microsoft Applied Sciences Group. http://www.microsoft.com/appliedsciences, January 2012c. Accessed: 19.01.2012 (url.microsoftASG).

Microsoft Inc. Popfly. http://popflyteam.spaces.live.com/, 2012d. Accessed: 20.01.2012 (url.popfly).

NaturalPoint Inc. OptiTrack. http://www.naturalpoint.com/optitrack/, 2012. Accessed: 07.03.2012 (url.optitrack).

Nintendo. Nintendo 3DS. http://www.nintendo.com/3ds/, 2012. Accessed: 18.05.2012 (url.nintendo3ds).

Nokia Inc. Nokia 5500 Sport. http://europe.nokia.com/find-products/devices/nokia-5500, 2012. Accessed: 18.05.2012 (url.nokia5500).

NVidia Inc. Physx — GeForce. http://www.geforce.com/hardware/technology/physx, 2012. Accessed: 01.05.2012 (url.physx).

openstreetmap.org. OpenStreetMap — The Free Wiki World Map. http://www.openstreetmap.org, 2012. Accessed: 29.04.2012 (url.openstreetmap).

OSGeo Project. PostGIS. http://postgis.refractions.net/, 2012. Accessed: 23.05.2012 (url.postgis).

Outercurve Foundation. Nuget. http://nuget.org/, 2012a. Accessed: 22.01.2012 (url.nuget).

Outercurve Foundation. Outercurve foundation. http://www.outercurve.org, 2012b. Accessed: 21.01.2012 (url.outercurve).

Artem Pavlenko. mapnik. http://www.mapnik.org, 2012. Accessed: 23.05.2012 (url.mapnik).

PCWorld. Confirmed: Google goggles will reach other platforms. http://www.pcworld.com/article/184011/confirmed_google_goggles_will_reach_other_platforms.html, 2009. Accessed: 18.05.2012 (url.googlegoggles.future).

Puredata. Puredata. http://www.puredata.info, 2012. Accessed: 19.01.2012 (url.pd).

William Putnam and R. Benjamin Knapp. Input/data acquisition system design for human computer interfacing, unpublished lecture notes. http://www.cs.princeton.edu/~prc/MUS539/Sensors.pdf, October 1996. Accessed: 17.03.2012 (url.PutnamKnapp96).

SAMH Engineering Services. SHAKE SK6: Sensing Hardware Accessory for Kinaesthetic Expression. http://shake-drivers.googlecode.com/files/SHAKE%20SK6%20User%20Manual%20Rev%20J.pdf, April 2009. Accessed: 18.05.2012 (url.shakesk6).

Nishant Sivakumar. A first look at c++/cli. http://www.codeproject.com/Articles/6882/A-first-look-at-C-CLI, 2004. Accessed: 21.01.2012 (url.cppcli).

Stackoverflow user KennyTM. Answer to question: is there any way at all that i can tell how hard the screen is being pressed? http://stackoverflow.com/questions/2103447/is-there-any-way-at-all-that-i-can-tell-how-hard\-the-screen-is-being-pressed/2103715, 01 2010. Accessed: 21.04.2012 (url.iphonepressure).

The MathWorks Inc. MATLAB. http://www.mathworks.com/products/matlab/, 2012. Accessed: 23.05.2012 (url.matlab).

Tomsguide.com. Samsung Launches Smart TVs With Gestures, Voice Control. http://www.tomsguide.com/us/samsung-led-plasma-smart-tv,news-14381.html, March 2012. Accessed: 05.05.2012 (url.samsunggesture).

VideoLAN Organization. VLC media player. http://www.videolan.org/vlc/, 2012. Accessed: 06.05.2012 (url.videolan).

Wii Brew. Wiimote/Extension Controllers/Wii Motion Plus. http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Wii_Motion_Plus, 2012. Accessed: 19.05.2012 (url.motionplus).

Wikipedia. Bump (application). http://en.wikipedia.org/wiki/Bump_(application), 2012. Accessed: 23.04.2012 (url.bump).

Wikipedia. Wikipedia: Motoral DynaTAC. http://en.wikipedia.org/wiki/Motorola_DynaTAC, 2012. Accessed: 18.06.2012 (url.dynatac).

Wikipedia. Wikipedia: Camera Phone. http://en.wikipedia.org/wiki/Camera_phone, 2012. Accessed: 18.05.2012 (url.wikipedia.camphone).

Xamarin. Mono project. http://www.mono-project.com/, 2012. Accessed: 18.04.2012 (url.mono).

Larry Yaeger. Larry Yaeger's Homepage: Apple Newton Handwriting Recognition. http://www.beanblossom.in.us/larryy/ANHR.html, 1996. Accessed: 18.05.2012 (url.inkwell).

Index

Typeset January 11, 2013.

Revision 341.