# Capturing the Expressiveness of Touch: Detect, Improve, and Extend

Der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University vorgelegte Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

von

Dipl.-Inform. Dipl.-Wirt.Inform. Maximilian Heinrich Gerhard Möllers
aus Düsseldorf, Deutschland

# Contents

# List of Figures

# List of Tables

# Abstract

Human-Computer-Interaction always tries to reduce user input to a very small amount of information (here touch to 2D point). This allows for easy input processing and easy systems development because the state machine has fewer transitions. It can also help the user, because a simpler UI can be easier to learn. However, we are missing out on a lot of expressiveness in the input from the user, and we will show how we could capture this expressiveness and how we could use it to give more precise input and more natural interfaces.

Touch is a means of communicating user intent, and we will show throughout this thesis that a simplification to a 2D point is a significant bottleneck for the interaction between a human and a computer, both on a micro and a macro level, and what we should do instead to grasp the user's intent and support more natural interactions.

As an introduction, we take a close look at what happens before a touch is performed and how this touch is then typically interpreted. Our first project shows a technical solution to get very precise touch information even on large tabletops, necessary for any deeper analysis of touches. We then show how in touch sequences one touch is affected by its predecessor, and that we can exploit this systematic error to improve touch accuracy. We also show extensions of current touch models to take into account body posture, relative location, etc. when interacting with larger tabletops. User location also plays an important role if we extend our direct manipulation surface to a 3D display. We show that the direct manipulation conflicts with the perspectively correct 3D rendering and how we can solve this conflict to have error-free direct manipulation.

So far, we have only covered flat, horizontal surfaces, but touches can be performed on any surface, and actually, people have more non-flat and/or mobile input devices at their disposal than tabletops: keyboards, mice, smartphones, remote controls, etc. Similar to our tabletop section, we start out with a technical solution to detect touches on arbitrary objects in 3D space. This raw touch data, however, contains a lot of misguided information because contact with the user's palm or fingers that are only holding a mobile device create the same input as the actual touch input. To reduce this problem, we propose an algorithm that infers the hand posture from touch data on an arbitrary object, making it easier to understand the user's intent.

As a closure for this thesis, we extend the existing touch-based GUI metaphor to support ad hoc interactions with arbitrary objects. We show how we can repurpose everyday objects as input controllers and remove the necessity of dedicated input devices to control our computers.

# Überblick

Beim Erstellen neuer Nutzerschnittstellen versucht das Forschungsfeld der Mensch-Computer-Interaktion die Nutzereingabe auf einen wesentlichen Kern zu reduzieren— bei Toucheingaben typischerweise ein 2D-Punkt. Dies erleichtert die Erkennung der Nutzereingabe und vereinfacht auch die Entwicklung von interaktiven Systemen, da die Eingaben weniger unterschiedlich sein können. Des Weiteren kann es den Nutzern helfen, das System besser zu verstehen, da es auf wenige und simple Eingaben reagiert. Auch wenn dies gute Gründe für die bisherige Entwicklung sind, sollten wir nicht die Gelegenheit verschenken und viele Aspekte der Ausdrucksvielfalt des Menschen ignorieren. In dieser Arbeit zeigen wir wie man diese Ausdrucksvielfalt besser erkennen kann und dass dies zu präziseren Eingaben und natürlicheren Schnittstellen fhürt.
In der Einführung zeigen wir zunächst, was eigentlich eine Toucheingabe ist: Was passiert bevor ein Mensch einen Touch ausführt, und wie wird die Eingabe von einem interaktiven System erkannt und interpretiert. Danach zeigen wir im ersten Projekt, wie wir auf einem Tabletop extrem präzise Toucheingaben ermöglichen können. Auf diesem System bauen einige der folgenden Kapitel auf: Im zweiten Kapitel zeigen wir, dass die vorherige Fingerposition einen Einfluss auf die darauf folgende Toucheingabe hat. Wir verdeutlichen auch, dass man diese Kontextinformation nutzen kann, um Fehler bei Toucheingaben zu vermeiden. Zuletzt zeigen wir in dem Kapitel, dass sogar die gesamte Körperhaltung Einfluss auf das Touchverhalten hat. Im dritten Kapitel erweitern wir unser System um einen weiteren Eingabekanal—die Position des Kopf des Nutzers. Das ermöglicht uns, unser 2D-Display zu einem 3D-Display zu erweitern. Wir zeigen dass die Metapher der direkten Manipulation, die typisch für Touchsysteme ist, im Konflikt mit einer perspektivisch korrekten 3D-Darstellung steht. Wir stellen Lösungsvorschläge vor, die zu einer fehlerfreien Interaktion führen.
Nachdem wir uns mit der Erweiterung des Touches um kontextuelle Informationen (z.B. Vorgängerposition) und weitere Eingabakanäle (z.B. die Kopfposition) auf planaren Oberflächen beschäftigt haben, werden wir in den folgenden drei Kapiteln dieselbe Fragestellungen schließlich auf beliebig geformten Oberflächen betrachten. Zunächst stellen wir auch hier eine technische Möglichkeit vor, die es erlaubt, beliebige Objekte im Raum zu erkennen und Touches darauf festzustellen. Diese Touchdaten sind jedoch nur der Anfang, da z.B. das Festhalten des Objektes selbst schon als Toucheingabe interpretiert würde. Wir stellen vor, wie wir aus den Touchdaten ermitteln können, wie das Objekt gehalten wurde. Diese Information ermöglicht uns, die Berührungen zu filtern, die nur Teil der Haltegeste sind. Im letzten Kapitel erweitern wir den Touch um die semantischen Informationen des Objektes. Wir erschaffen damit eine neue Interaktionsmetapher für ad-hoc Zweckentfremdung von Alltagsgegenständen. Wir zeigen, wie diese Gegenstände als Eingabekontroller genutzt werden können und somit dedizierte Eingabegeräte überflüssig machen.

# Acknowledgements

First of all, I want to thank my advisor, Jan Borchers, for his continued support of my work. He was the one that got me into HCI in the first place and his chair the primary reason I decided to pursue a Doctoral degree. He also inspired me to extend my research on tabletops to arbitrary surfaces and was always willing to provide comments on my papers and research ideas.
I would also like to thank my co-advisor, Mark Hancock, for the inspirational conversations. His ideas helped to shape this work.
I would also like to acknowledge all of the master, diploma students, and student assistants that performed a tremendous amount of work to support me and helped my visions come true: Norbert Dumont and Patrick Zimmer for their exceptional work on my tabletop research ideas as well as Christian Corsten and Ignacio Avellino for the arbitrary surfaces. Additionally, Radek Paluszak for his support in performing the user studies.
My colleagues played a very important role in this process, especially Thorsten Karrer, Leonhard Lichtschlag, Chatchavan Wacharamanotham, Moritz Wittenhagen, and Christian Corsten. They were always there to give advice on my problems and helped me to shape up my ideas for paper submissions and, most importantly, made the five years a really great and fun experience. I can only hope to have the same quality of people to work with in the future.
A very special thank you goes to my wife and kids who reminded me that life is not only about becoming a Doctor, while still supporting me in achieving this goal.
Finally, I would like to dedicate this thesis to my parents whose determination in achieving their goals encouraged me over and over again to do my best, which made this all possible. Thank you for your love and support!

# Conventions

The following conventions will be used throughout this thesis:

Technical terms or jargon that appear for the first time will be set in *italics*.

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

Some of the material in this thesis has also been previously published in conferences or journals, and this will be noted in the text.

This thesis is written in American English.

# Chapter 1

# Introduction

Over the last years, touch has become the dominant input modality in end-user devices such as tablets, smartphones, in-car entertainment, etc. due to the benefits of direct manipulation and physical metaphors that ease learning how to use such a device. Research extended the touch-capable device landscape to vertical and horizontal surfaces and continues to explore other form factors. What all those systems have in common is that in their first iterations they interpreted a touch as a simple 2D point similar to a mouse cursor due to the ease of development and a lack of better understanding of what a touch means. This simplification of the user's intent to a 2D point significantly narrows the communication channel between Human and Computer. It causes erroneous touch detection and imprecise touch locations. To deal with those problems, current mobile phones have a deeper understanding of how people interact with them and can thus provide more precise input. Although more precise input (on mobile phones) is a significant step forward, it is only an improvement on a rather low communication level, leaving out improvements that are based on more abstract levels, such as user intent and application context.

Touches are considered to be just 2D points leading to erroneous touch detection, imprecise touch location, and missing out opportunities for other interaction style.

Thus, the overall goal of the thesis is to improve the communication between humans and computers in the context of touch interfaces. We will show how the computer can get a better understanding of the user's input to provide more reliable input processing and enable the aforementioned new interaction techniques. We will also show new interaction techniques and metaphors that allow for a more natural input that requires less knowledge on the user's side but more on the side of the system.

The thesis improves and extends the use of touches.

To be more specific, we will show how we can not only extend the improvements from the small form factors to larger surfaces, but also how we can capture the user's intent based not only on the touch position but on its body posture and the task and context. This deeper understanding then allows us to leave behind dedicated input devices and, instead, repurpose everyday objects to convey our intentions to a computer. This ad hoc interaction opens up a new design space, and we will show first results on how to apply this new interaction metaphor to create more flexible user

Create new interaction techniques based on object affordances.

interfaces. This metaphor leaves behind the traditional cursor-based desktop interactions and replaces them with interaction artifacts that react to touches as expected from the existing object affordances.

We will now state what the current understanding of a touch is and raise our overarching question for the thesis.

## 1.1 What is a Touch?

We first take a look at what happens before a touch is performed and see how the touch is typically captured in current systems.

### 1.1.1 Mental Models

In general, touch is a means of communicating intent to a (computer) interface. But how does one know where to touch, what to touch, or whether to touch at all? Don Norman presented in his seminal work *The Psychology of Everyday Things* (Norman [1988]) a way to understand how humans shape their actions based on their targeted outcome. He calls his model *The Seven Stages of Action* (c.f. Figure 1.1).

Seven Stages of Action explain how users interact with the world.

At first, the person has a specific *goal* in mind, e.g., he wants to know the stock price for his IBM shares. He then needs to form an *intention* for an action that will help him reach his goal. In our example, he could look up the stock price on an app on his smartphone. This intention is then translated to *sequence of actions*: pick up the phone, open the app, and take a look at the price. The person then *executes* this action sequence. The world, here his smartphone, responds to the input by displaying the appropriate data. The user now first needs to *perceive* this change in the world by looking at it. He then *interprets* what he sees, in our case the IBM stock price. He then *evaluates* and compares the outcome with his goal: knowing the IBM stock price. In our example, the user was able to successfully achieve his goal, but there were different gaps to cross throughout the seven stages. As this thesis strongly focusses on input, we will now take a closer look at how the user ended up with the action sequence he executed.

Users can have problems finding the correct action sequence.

Starting at the top, the user needs to have a general understanding of how he can change the world to end up in his desired state. If he does not know that his smartphone can tell him the stock price, he would need to find another solution by asking a friend, calling his favorite broker, or going to Wall Street. But if the user is stuck at this stage, we as a designer of a user interface cannot do much more than creating an all-knowing artificial intelligence that can point him into the right direction—using google with "How do you do X?" comes quite close in this regard.

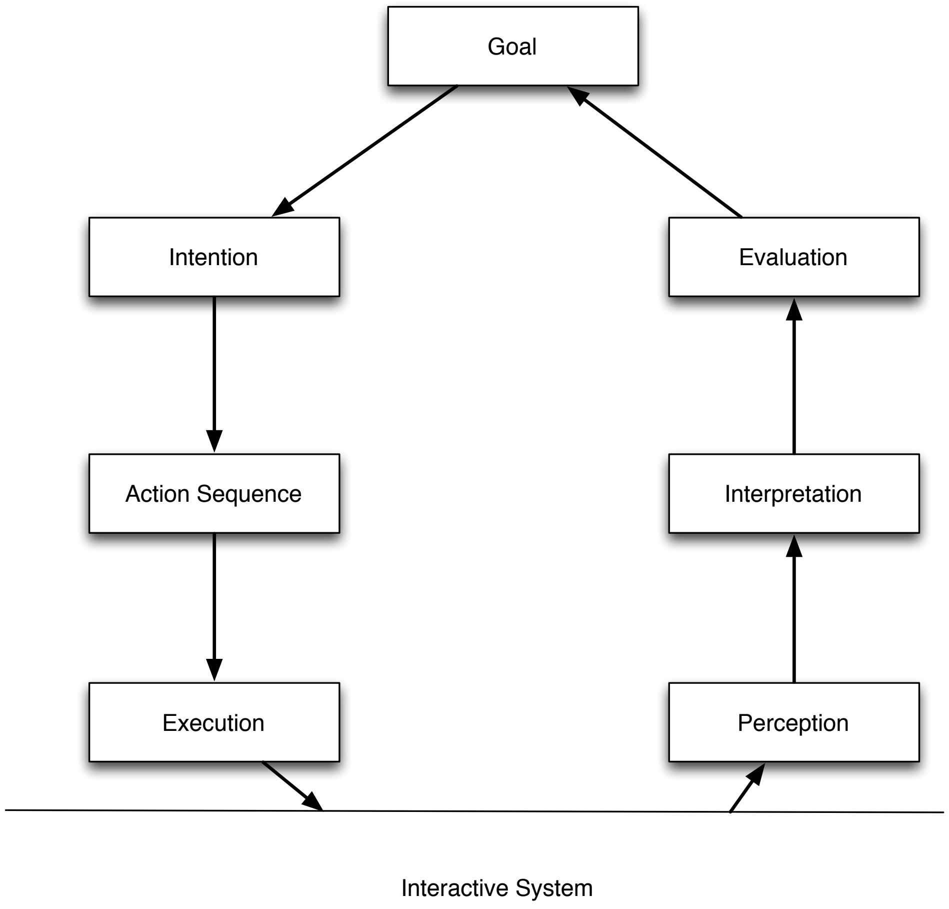


**Figure 1.1:** The Seven Stages of Action. **Left column:** The user wants to achieve his goal and has to prepare his actions. **Right column:** The interactive system responds, and the user now has to decide whether he achieved his goal.

Affordances can ease this.

However, if the user knows that a given system should be able to support him in achieving his goals, the developer of the system can design it in such a way that the system's visual and tactile appearance suggests the correct usage. Norman calls this affordances of a system:*"The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used. [...] Affordances provide strong clues to the operations of things. Plates are for pushing. Knobs are for turning. Slots are for inserting things into. Balls are for throwing or bouncing. When affordances are taken advantage of, the user knows what to do just by looking: no picture, label, or instruction needed."* (Norman [1988]).

Although Norman introduced affordances as properties of physical "things", they are easily applied to digital components of a graphical user interface, such as buttons, sliders, etc., that typically rely heavily on physical metaphors. We will revisit the concept of affordances in Chapter 7, where we take a look at how we can repurpose everyday objects as input controllers. This idea is based on the fact that people already know how to use an everyday object, so there is no need to explain them the user interface;

the challenge is to find a meaningful interpretation of the user's input.

Direct manipulation via touch instead of indirect manipulation via pointing device.

Another term that needs to be mentioned is *direct manipulation*, which was coined by Shneiderman [1983]. He introduced it for graphical user interface to differentiate between editors that allow you to change text directly at the cursor and editors that you need to tell “delete third word in line 5”. Today it can be applied to differentiate between indirect input methods, such as the mouse (where you move the mouse on your desk and a cursor on the screen gets moved), and a direct method such as panning a website on a smartphone where you are directly interacting with the digital content based on some physical metaphor. Other examples are rotating objects with two fingers and zooming in and out. Again, interaction methods based on these kind of physical metaphors can help to convey the behavior of the user interface to the user, making it easier for him to come up with successful action sequences based on these intentions.

After action sequences are found, their execution is performed in Norman's Model. For our later analysis, however, we need to take a closer look at how an action sequence such as “grab phone and touch the app button” is transformed into a series of muscle movements which in turn are then executed. There exists a vast body of psychology research that gives a lot of insight—the interested reader might want to start with the book from Heuer and Keele [1996]— but here we want to highlight two important findings. Fitts [1954] found out that there is a rather strict connection between time to acquire, i.e, touch, a target and its distance from our current hand position and the width of target w.r.t. the distance vector. His model can be used to predict targeting times and thus can support interface design. For example, the corners of a screen have an “infinite” width and are thus the fastest targets to point at. This is the reason why operating systems use them for frequently used commands: start button in Windows, close window button, the “magic corners” from Mac OS X, etc.

Fitts' Law predicts targeting times.

Larger and more complex body parts target less accurate than smaller ones.

There is also research that not only tells us about how fast we are but how accurate we can perform touches. For example, Hammerton and Tickner [1966] showed that the larger the body part that we use to perform input, the worse the accuracy: Thumb only is more accurate than using only the wrist joint which is more accurate than using the elbow joint. This can directly be applied to our current touch interfaces (see Figure 1.2): Mobile phones are primarily used with one thumb or one finger, tablets are used primarily with one finger that is controlled via finger, wrist and small elbow joint movements, and our biggest form factor—tabletops— can sometimes require movement of the whole body when touching a far away spot. Thus we can expect that larger form factors result in less accurate input, and Chapter 3 will tell us how accurate people actually do perform on a tabletop.

People use the most convenient way to perform successive actions.

Interestingly, our body is “aware” of these inaccuracies and tries to optimize our movement. Rosenbaum et al. [1990] showed that people tend to plan actions ahead and use the most convenient way from a motoric and muscular perspective to perform successive actions. We will revisit this concept in

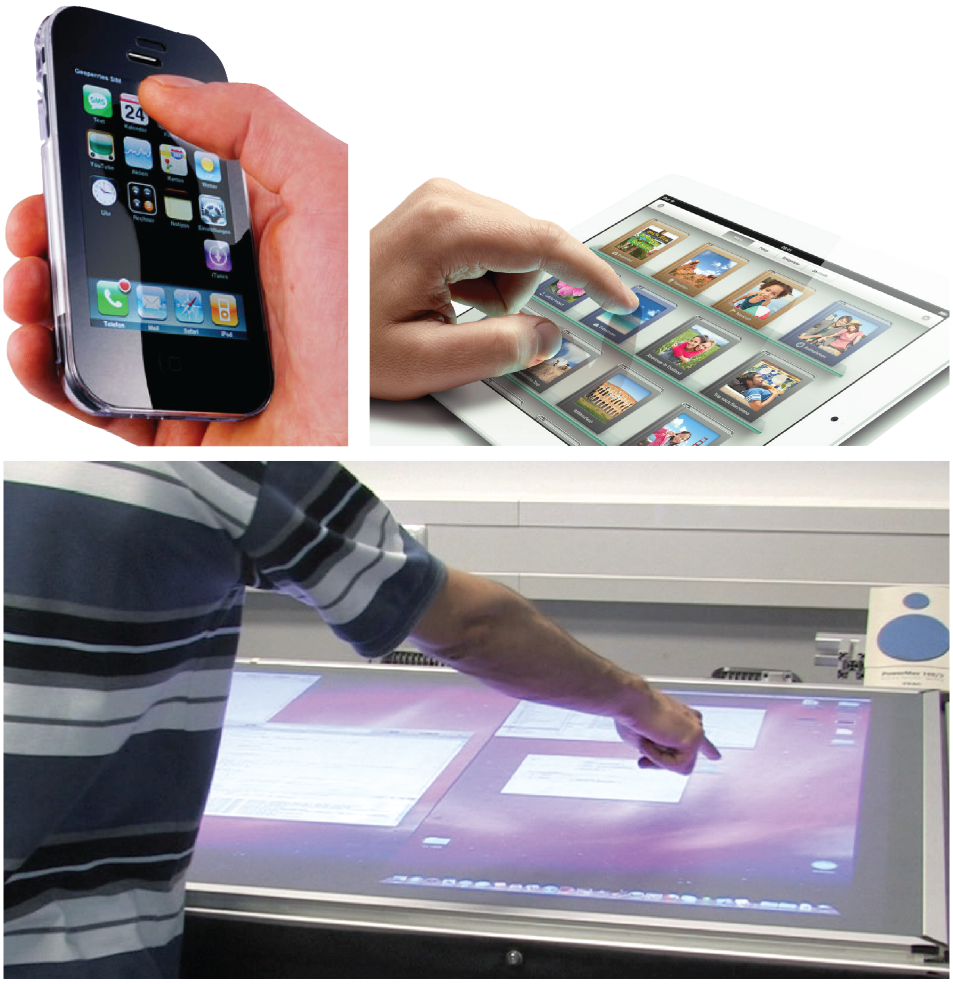


**Figure 1.2:** Different form factors of touch-capable devices. The larger the device, the more body parts are involved in the interaction and the less precise is the interaction with the device.

the second part of Chapter 3, where we show how one touch depends on its predecessor and successor.

After this detailed look at the human side of how a touch is performed, we now describe how current systems detect a touch.

### 1.1.2 Touch Detection

Touch detection is mostly done with vision-based or capacitive systems.

There exists a variety of ways to detect touch input, the two most prominent being *capacitive* and *vision-based*. Capacitive detection uses a 2D grid of thin wires that act as antennas and sense the change in capacitance on and above the grid. If a finger (or another object with a high capacitance) comes close to the grid, a touch is detected. In vision-based systems, the surface is watched by a camera, e.g., in an interactive tabletop the camera is usually beneath the surface and looking to the top. Vision-based systems typically illuminate the scene with infrared light as it is strongly reflected by the hands and less so from the background and makes it easier to distinguish the hand from the background.

Both need to analyze a 2D image.

We will go into the technical details in the next chapter, but what capacitive and vision-based methods have in common is that they both return a 2D image that contains contact information for each pixel. In the case of the vision-based approach, it is an intensity value of the infrared signal for each pixel (Figure 1.3), in the case of capacitive tracking, it is the capacitance at each grid point.

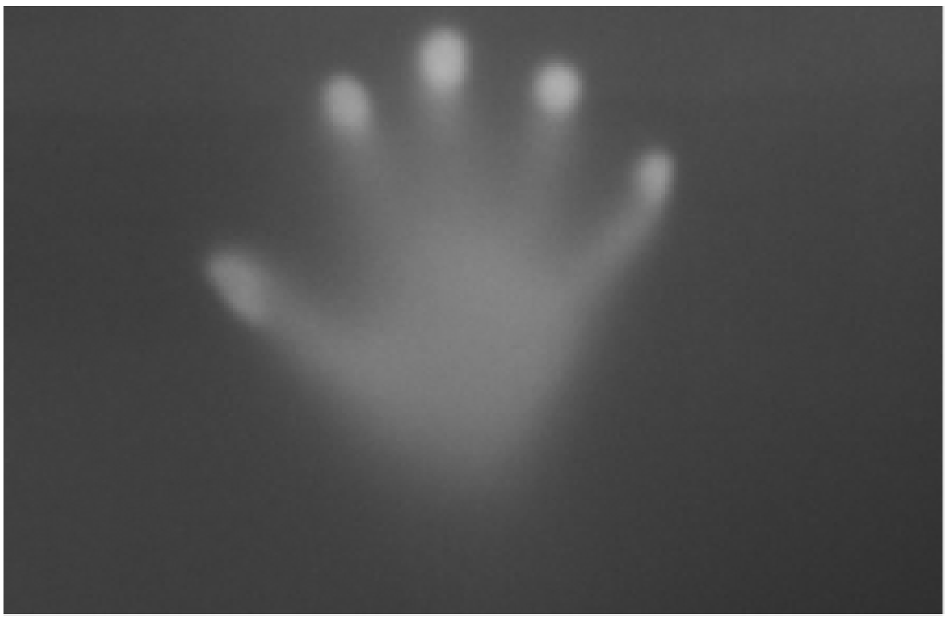


**Figure 1.3:** A camera image from a vision-based touch detection system. The finger tips are brightest because they are closest to the light source and the camera.

Standard computer vision methods give us touch ellipses.

We now can use standard computer vision methods to detect the touches (c.f. Figure 1.4): subtract the background, threshold the intensity values to a binary image to remove noise, retrieve connected components, filter out components based on criteria such as size or shape, and compute the principal axes (e.g. by using *Principal Component Analysis*, Bishop [2007]). This way we end up with a contact ellipses for each touch. The next steps are highly hardware and software dependent, not only regarding how it is processed but also which part, i.e., onboard hardware controller, input driver, a multi-touch frame work, or the application itself, perform them. Typical steps include the detection of multi-touch gestures and tracking one touch point from frame to frame.

Different properties of a touch can be used. Not only the center point.

Table 1.1 shows an overview of what properties we could use to capture the user's intent. Interestingly, on an application level, the touch is often only considered as a 2D point and, other properties such as orientation and contact area size are ignored. Given robust ellipse detection, the orientation can be used to, e.g., select an entry from a pie menu (c.f. Figure 1.5), as was done first by Wang and Ren [2009]. Also, if we were taking into account the orientation, we could perform object rotation with one finger. Instead,

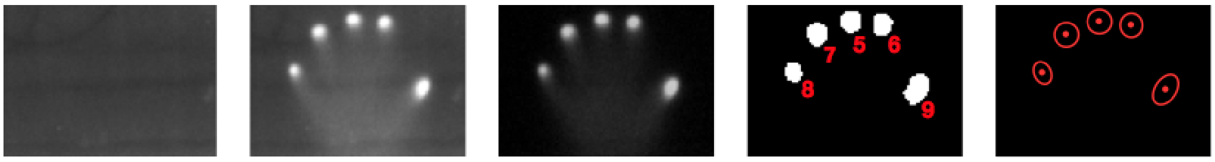


**Figure 1.4:** A typical image processing pipeline. We store the background image without any hands (1) during setup. We get a new frame from the camera (2), delete the background (3), retrieve connected components (4), and fit ellipses (5).

| Input Aspect | Finger Property |
|---|---|
| **Position** | Coordinate Value (x, y) |
| **Motion** | Velocity<br>Acceleration |
| **Contact** | Size of Contact Area<br>Shape of Contact Area<br>Orientation<br>Pressure |
| **Event** | Tap<br>Flick |

**Table 1.1:** Different properties of a touch. Taken from Wang and Ren [2009].

current systems ask us to use two fingers: one for the rotation anchor, the other for the rotation angle. This is one simple example where we can see that current systems are not fully exploiting the expressiveness that a touch provides.

Good reasons exist for this reduction.

There are good reasons why touch system have evolved this way, i.e., use only the $(x, y)$ of the touch as a concept for the user, indicated by the cursor. Current (desktop) applications still need to be used with a standard 2-button mouse that does not have a rotation input. (i) It is easy to build hardware to reliably detect the $(x, y)$ input. (ii) It is easy to interpret this input by the software. (iii) This also means that the applications is built upon a simpler state machine because the transitions are limited more. (iv) A simpler state machine can make it easier for the user to learn how to operate the applications. No matter how comprehensible this development over the last decade is, it should not hinder us to explore more fitting uses of touch input that do not restrict themselves to the 2D point and take into account more contextual and additional information apart from the touch itself. This will be the focus of this thesis and will guide us from a very technologically-minded interaction model to a more natural interaction.

Better reasons exist for an extension

### 1.1.3 Summary

Concluding this fundamentals section, we have introduced the seven stages of action that are one model to understand how a person changes the world around him. We highlighted *affordances* which are system properties that

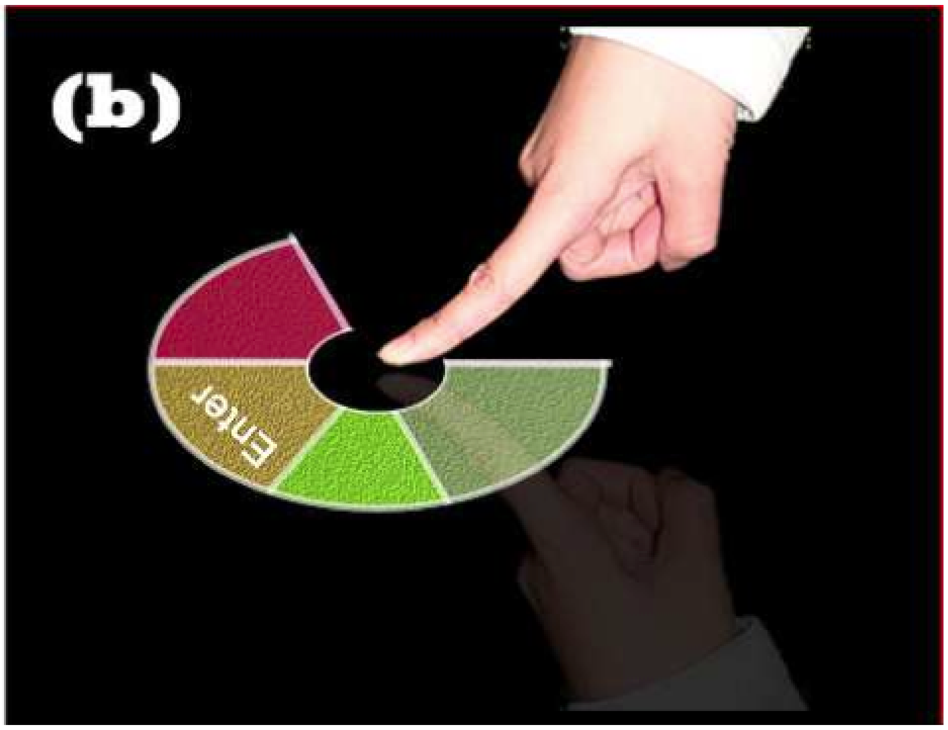


**Figure 1.5:** Finger orientation selects entries from a pie menu. Taken from Wang and Ren [2009].

suggest a specific system usage to the user to guide him in the right direction. We also presented some work from psychology that describe the planning of the actual movement and showed some examples for single touch interactions with different form factors. We also presented a typical touch detection pipeline and raised the overarching question of our thesis: "What is a touch?".

## 1.2 Contributions

The main contributions of this thesis are as follow:

- Tracking hardware for very precise touch detection on large tabletops as well as a system for tracking arbitrary objects in 3D space without markers while detecting touches on them.
- A method to infer the hand posture from touch data on arbitrary objects.
- A deeper understanding of how people interact with a tabletop and how we can exploit this behavior to improve touch accuracy.
- An interaction technique that exploits the deeper understanding of touch on a 3D enhanced tabletop.

- A new interaction metaphor based on repurposing everyday objects as input controllers with a series of evaluations.

By studying this thesis, the reader will not only gain a considerable amount of knowledge on how humans perform touch and how it is currently processed and used in various contexts, but also find inspiration for future interaction techniques and applications, opening up interesting research venues and allowing him to better understand the user's (touch) input.

## 1.3 Structure

The thesis is structured in three different chapter categories: "detect, improve, and extend". In "detect" chapters, we propose novel tracking and touch detection methods to capture the raw touch data. In "improve" chapters, we take into account contextual data, such as body posture or the location of the previous touch, to get a better understanding of the users input gesture and increase touch accuracy. In the "extend" chapters, we add semantic information, such as the object the user is interacting with, to create more sophisticated applications based on touch interfaces.

We will first focus on touches on tabletops and afterwards extend our work to arbitrary objects.

**Chapter 2** shows a technical solution to get very precise touch information even on large tabletops, necessary for any deeper analysis of touches.

**Chapter 3** takes a closer look at touch sequences on tabletops and how one touch is affected by its predecessor and how we can exploit this systematic error to improve touch accuracy. Here, we also extend current touch prediction models to take into account the full body posture and show how large tabletops differ from smaller surfaces.

**Chapter 4** gives insight into how the direct manipulation metaphor needs to be adapted if we not only consider the user's touch as input but also take part of the user's body, i.e., its head position, into account when creating a 3D display.

The following chapters then extend our focus from interactive tabletops to arbitrary interactive objects.

**Chapter 5** shows a technical solution to detect touches on arbitrary objects in 3D space.

**Chapter 6** shows how we can filter the raw touch data coming from an arbitrary object by inferring the hand posture.

**Chapter 7** as the final chapter gives an outlook into the future and presents a metaphor to interact with an arbitrary object in an ad hoc fashion. It shows findings on what objects are actually apparent in the user surrounding, how the objects' shapes afford the (touch) interaction, and gives insight into the different ways to design an application in this context.

**Chapter 8** summarizes the work presented and provides an outlook to future extensions of our ideas.

Having laid out this basic knowledge, the next chapter will show a way to combine high-precision touch detection with large tabletops, necessary for deeper analysis and understanding of how people perform touches.

# Chapter 2

# Detecting Touch on Tabletops

In the previous chapter, we have laid out the basics of detecting a touch. However, the typical solutions for detecting touch on larger interactive tabletops do not provide us with a high input resolution and have problems robustly detecting the orientation of the finger. We will now present our approach in detail. This work has been carried out with the Diploma student Norbert Dumont. More information can be found in his thesis, Dumont [2012].

## 2.1 Related Work

Opted for vision-based touch detection.

There are several options to create a multi-touch capable display: (i) capacitive sensing, where the proximity of fingers change the capacity of thin metal threads made from, e.g., copper, (ii) resistive, where pressure connects two conductive layers, (iii) sound-based, where fingers block sound waves, and (iv) vision-based, where one or more (infrared) images are taken from the side of or behind the screen. As (i) and (ii) require industrial production capabilities and are not available in large table sizes, and (iii) is less flexible for multi-touch input, we decided to employ a vision-based approach. Schöning et al. [2010] presented a detailed overview of all the detection techniques.

Use Diffuse Illumination to differentiate background from foreground.

In vision-based systems, the main challenge is to differentiate the fingertips from the background. To make this easier, infrared light is used as (ambient) lighting which increases the contrast between both because infrared light is strongly reflected from the hands and less so from the background. This technique was named *Diffuse Illumination (DI)* by Matsushita and Rekimoto [1997].

FTIR gives higher contrast images, but do not provide hover.

Instead of flooding the scene, we can also flood the transparent surface of

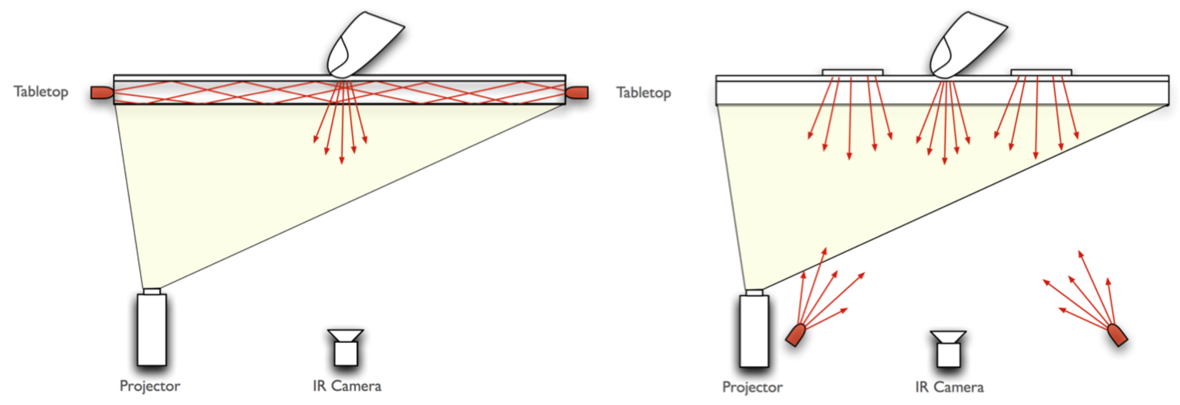


**Figure 2.1:** Touch detection setups. **Left:** Frustrated Total Internal Reflection, **Right:** Diffuse Ilumination. Taken from Weiss [2012].

the tabletop with infrared light and employ the effect of *Frustrated Total Internal Reflection (FTIR)* as introduced by Han [2005]. In FTIR, an acrylic plate is flooded with infrared light from the side. The light is totally reflected inside the plate (Figure 2.1). As soon as a finger touches the surface the light is no longer totally reflected at the acrylic-air passage but frustrated by the acrylic-skin combination. This way, the infrared light is shining downwards, where an infrared camera would see a bright spot on an otherwise dark image. Both techniques are shown in Figure 2.1, and the resulting raw camera images are shown in Figure 2.2. The typical contact area of a finger can be approximated with an ellipse, and its main axis is the (undirected) finger orientation.

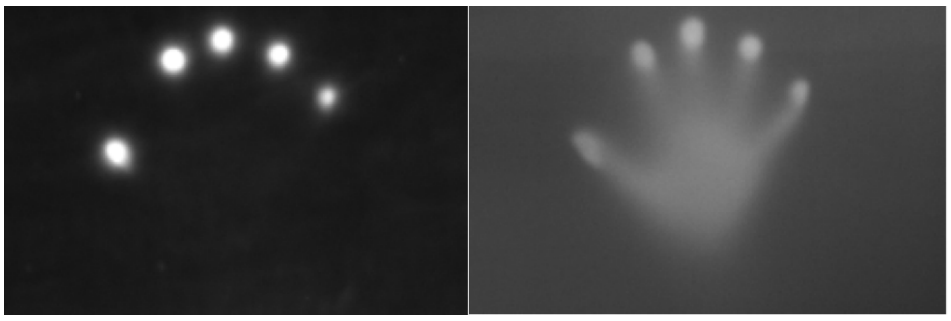


**Figure 2.2:** Camera images. **Left:** With FTIR only the finger tips touching the surface are visible. **Right:** With DI we can also see the hovering rest of the hand but have a lower contrast on the finger tips themselves.

Analyze multiple height levels of the touch to get the finger posture.

Before showing our approach to get the directed orientation, we want to mention other solutions to a more detailed finger tracking. FTIR provides contact information on one height level, a natural extension thus is to take a look at different height levels. Z-Touch by Takeoka et al. [2010] uses multiple lasers on different height levels that detect intersecting fingers, similar to a light curtain in an elevator( c.f. Figure 2.3). From this data, it can not only infer the orientation but even the posture of the touch and take it into account when designing interaction techniques.

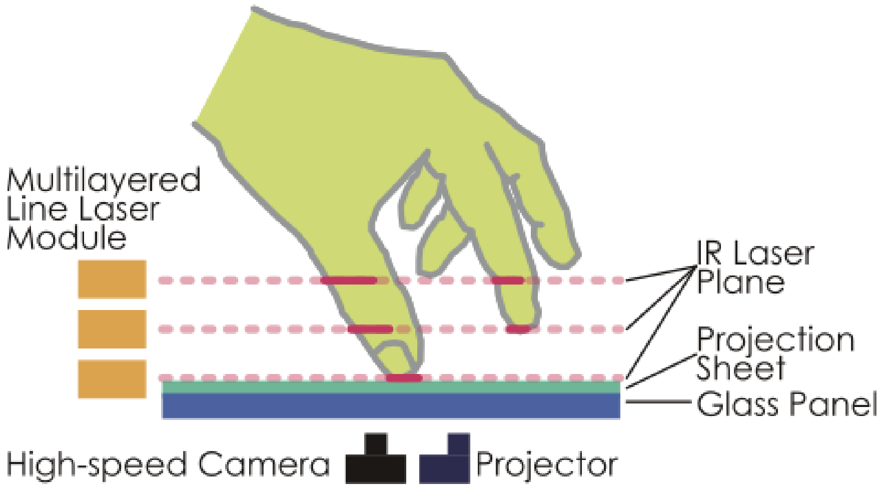


**Figure 2.3:** Setup for the Z-Touch system. Multiple lasers scan for touch on different height levels and can infer the finger posture. Taken from Takeoka et al. [2010]

Depth-cameras can detect touches but are not yet available with high resolutions.

Another vision-based option is to use depth-sensing to get more than just the center point of a touch as, e.g., was done by Wilson [2010] . An overhead 3D camera, e.g., MS Kinect provides us with depth information for each pixel (c.f. Figure 2.4), and simple thresholding based on the touch surface and the thickness of the finger will return touch information. Other approaches use more sophisticated models, such as Wang et al. [2011], and are able to detect position, orientation, and gestures of the whole hand in real-time. Current depth-cameras, however, only provide low resolution images and thus does not give high precision.

## 2.2 Implementation

Our method uses diffuse illumination and steerable high-res cameras.

For our research goal of understanding touch on interactive tabletops, we need a high precision touch detection as well as a robust detection of the finger orientation. None of the aforementioned support both at the same time, so we came up with our own method. It is based on a) Diffuse Illumination tracking to get the finger orientation and b) uses steerable high-res cameras with a narrow focus that we can point at the area of the touchpoint. We will explain both parts in detail after introducing our general tabletop setup.

A wireframe rendering can be seen in Figure 2.5. The table is constructed from aluminum frames and uses three FullHD DLP projectors to provide a resolution of $3240 \times 1920$ px on our 140x80 cm display, yielding a 60 dpi display, which is sufficient for reading text in a font of at least 10 pt—common desktop displays have 80–100 dpi.

We use foil mirrors that allow an affordable, yet very planar reflection of the

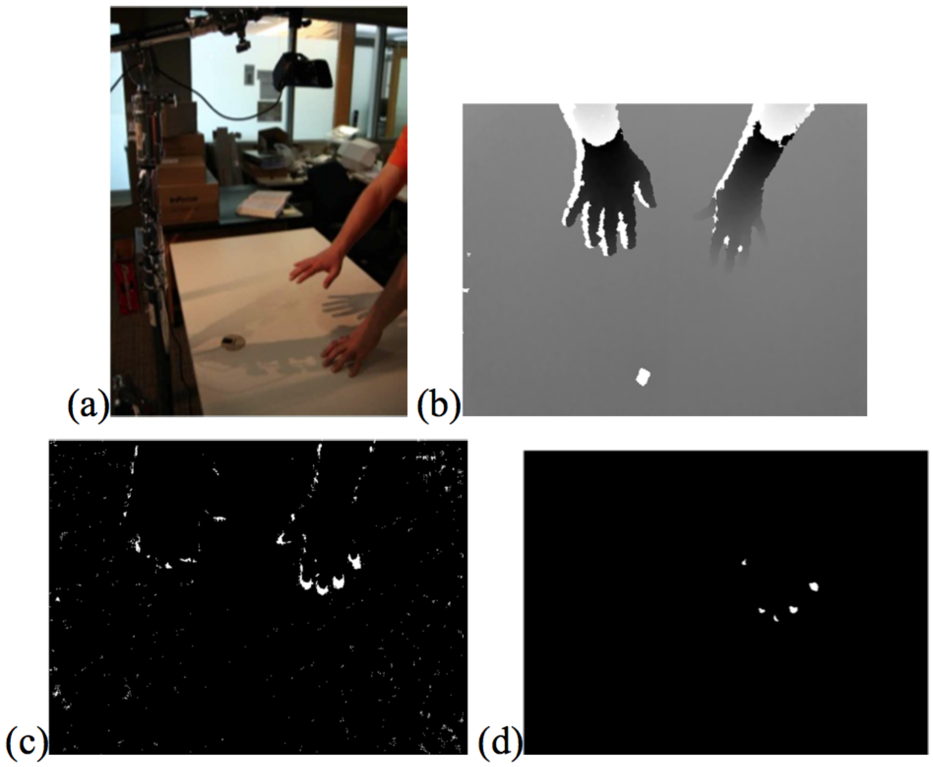


**Figure 2.4:** (a) Setup for a depth-camera, (b) Raw data from the camera, (c) Touch image after depth thresholding, (d) Final image after low pass filter. Taken from Wilson [2010].

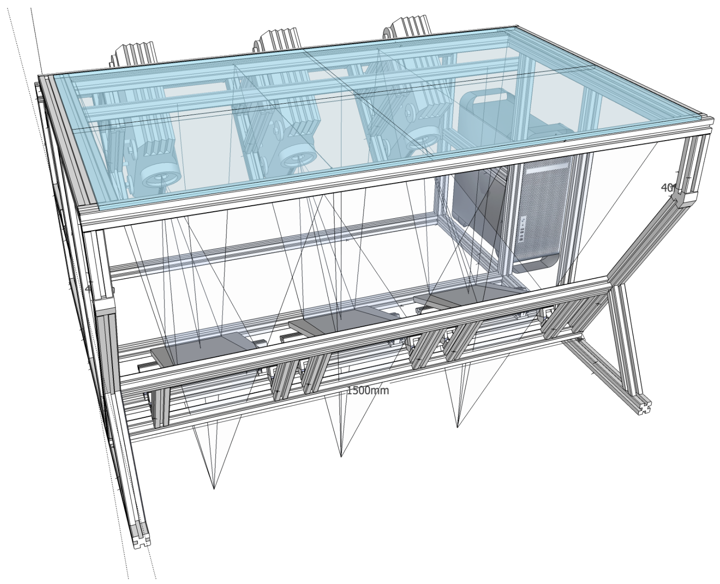


**Figure 2.5:** Wireframe sketch of our multi-touch enabled desk. The three projectors are aimed at the mirrors in the bottom. The two cameras (not shown here) are placed between the mirrors and look upwards.

image upwards onto the surface of the table. We decided for this mirror setup to decrease the height of the table because the projectors have a minimum projection distance.

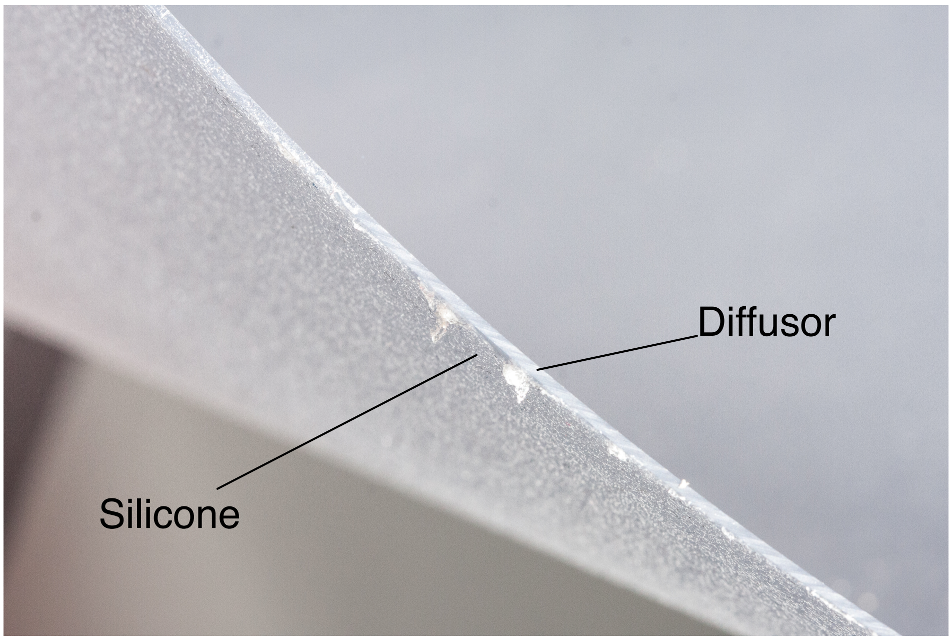


**Figure 2.6:** Closeup of the acrylic diffusor covering the tabletop. The small texture of silicone emulates touches of the user.

The surface is a 1.2 cm thick sheet of *Plexi Glass*. This acrylic sheet is framed by aluminum panels containing high powered infrared LED strips providing the lighting necessary for FTIR tracking. The frames also serve as a passive cooler for the 5.7 ampere that flow through the LEDs.

Directly touching the acrylic would refract the light and scatter it downwards to the cameras. But we need a diffusor to project on. This scratch resistant diffusor is placed on the acrylic but prevents the FTIR effect to happen because the skin is no longer touching the acrylic. Therefore, we roll a 1 mm layer of textured silicone onto the downside of the diffusor (Figure 2.6).

The silicon emulates a touch and creates the FTIR effect.

This way, pushing the diffusor will make the silicon ”push” the acrylic, thus resulting in the FTIR effect because silicone and human skin share similar refraction properties. The silicone needs to be textured in order to not stick to the acrylic where it would then permanently create the FTIR effect, making it impossible to detect actual touches in that area. The texture is a byproduct of rolling the silicon and not, e.g., casting it on to the diffusor.

We also added 24 plates with infrared LED arrays to the backside of the bottom of the table (Figure 2.7). These LEDs provide the lighting for the DI system. However, a careful placing of the LEDs and the cameras is necessary in order not to have specular reflections from the acrylic sheet visible in the camera image. That would make it impossible to see touches in this area. That is why we chose to put the LEDs to the backside, close to the cameras, so their reflections show up only close to the backside of

24 LED arrays are placed on the backside and beneath of the table for diffuse illumination.

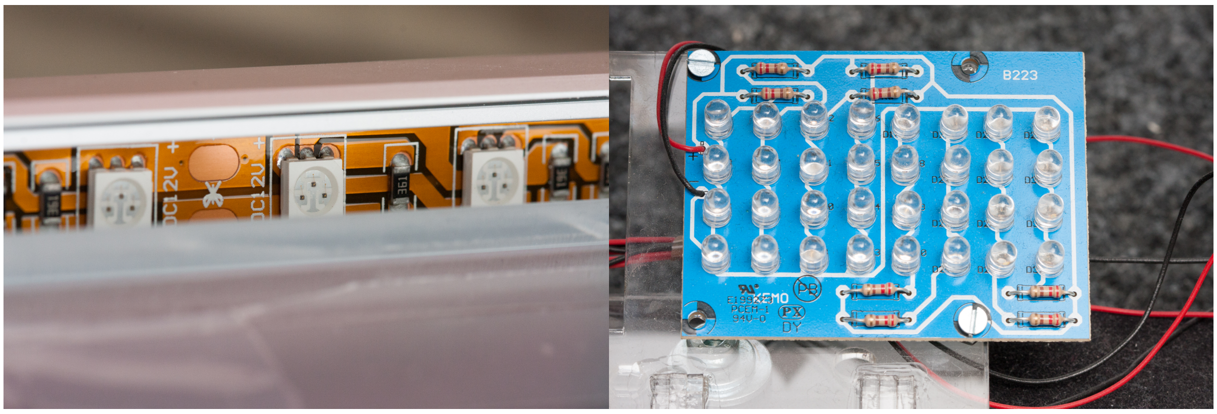


**Figure 2.7: Left:** Strip of infrared LEDs that emit directly into the side of the acrylic. **Right** Array of infrared LEDs that provide a diffuse illumination from beneath the table.

the table (c.f. Figure 2.8).

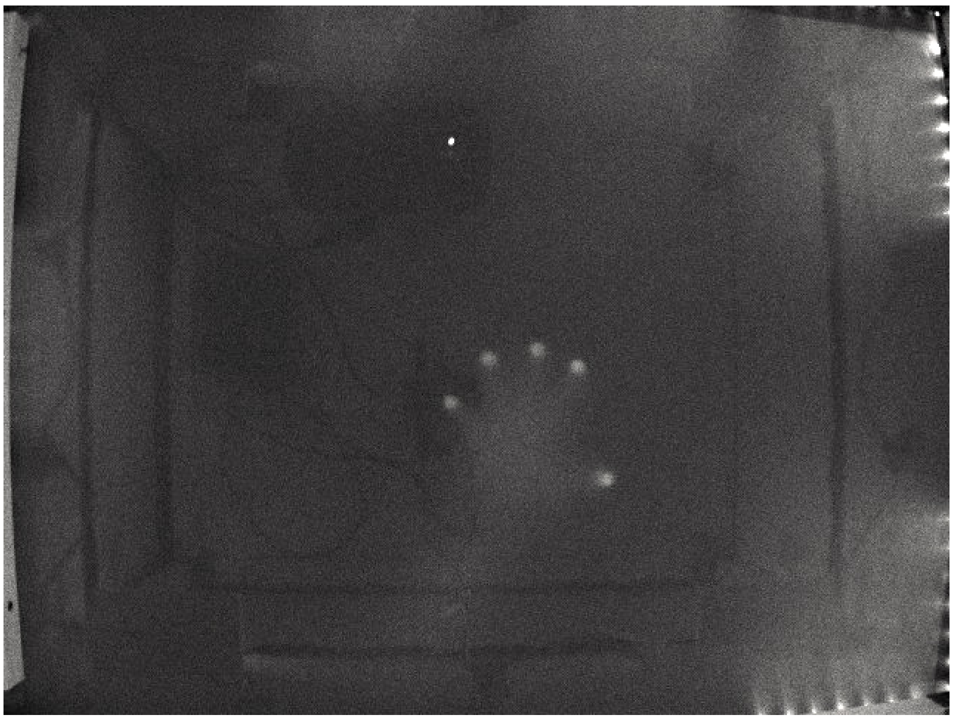


**Figure 2.8:** The view from the camera beneath our multi-touch tables. This image is background subtracted, thresholded, and then processed using a connected component analysis. This gives us the five bright spots—the fingers touching the surface.

Thus, our table supports DI and FTIR tracking and we can switch between according to the needs of the study, i.e., higher contrast finger tracking or information about the hovering hand.

Higher input resolution necessary for our studies.

This multi-touch table provides a surface large enough for our study and is a typical setup that can be found in other research labs over the world. However, the two cameras only give an input resolution of 23 dpi. This is sufficient for standard touch tracking, but we doubted it would net us significant study result (as it was three times less than the output resolution of  60 dpi, i.e., every touch point is hitting three display pixels).

In the upcoming studies in Chapter 3, we are only interested in a small region of the display at a time, i.e., the area around the currently displayed target that the user needed to touch. We can exploit that by making the camera only look at this location, which would then give us its full resolution at the rather small area. We used narrower lenses and put the camera on a self-built mount that can rotate the camera along two axes (Figure 2.9). The system uses stepping motors and hall sensors for accurate tilting. The motors are controlled via arduino micro controllers. More implementation details can be found in the master thesis of Dumont [2012].

Narrow lenses and servomotors are used to aim the camera at small parts of the table.

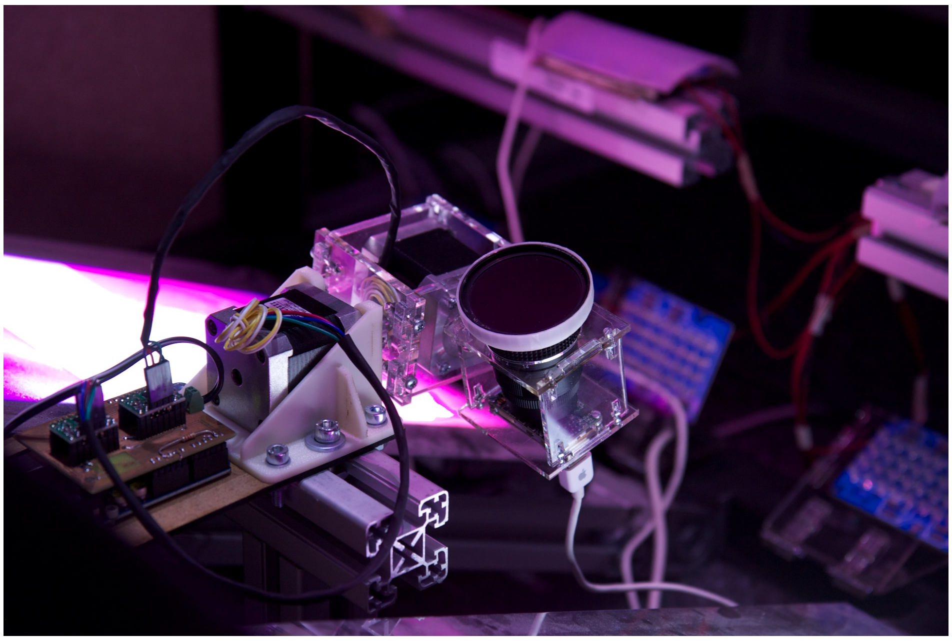


**Figure 2.9:** A camera on a custom mount. It allows for rotation along two axes and can thus be pointed at arbitrary regions of the table.

Using two of these cameras with automated aiming, we were able to cover the whole table but have an input resolution of 160 dpi for each of the targets. Pointing the camera at a new touch area takes only a fraction of a second. The merging of camera images, the system calibration, as well as perspective correct unwarping of the images is performed by the chair's own MultiCameraTracking framework. More details about it can be found in Voelker [2010].

Multiple cameras are supported by the MultiCamera-Tracking framework.

We employed Diffuse Illumination (DI) to detect the finger orientation with minimal effort: DI gives us the shadow of the full finger hovering the table, and we can use this information to detect the direction (c.f. Figure 2.10) by comparing the average brightness in front of the ellipse and behind it.

The hovering hand tells us where the finger is pointing.

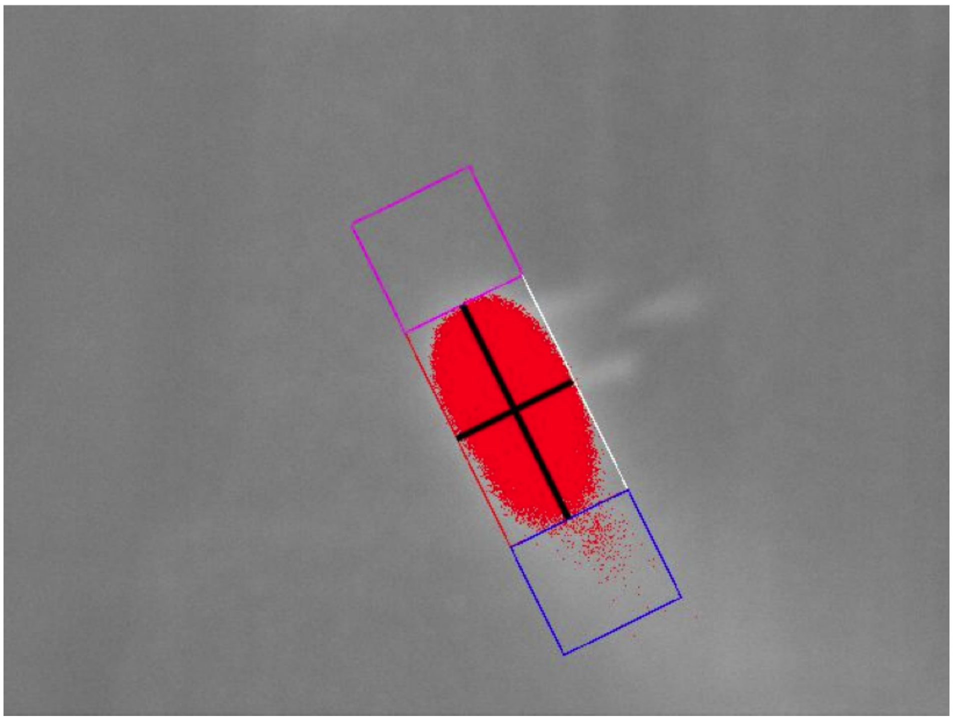


**Figure 2.10:** Spot as seen by the camera. Main and minor axis can be seen, and the area in front of the finger is darker than the one behind it, thus we can detect the direction.

## 2.3 Limitations & Future Work

Currently, we need to know the general touch region beforehand.

Our current system is a great setup for the studies ahead, yet there might be usage scenarios where the system falls short. Pointing the camera at a touch area takes up less than half a second, but this makes our system not usable for a real application scenario where user input is not restricted or at least known ahead. One solution would be to use a lot of cameras that are fixed to one area, for our setup this would mean 112 cameras instead of two to achieve the same resolution. This could be compensated by higher resolution cameras, but both options would be expensive and require significant processing power to look for touches in all the spots at the same time.

Combine high- and low-res cameras to track the whole screen while being able to focus on the touches after a short time.

Another option that we would like to pursue in a future work is to have a small amount of standard resolution cameras that are fixed and can together see the whole table. Additionally, we would have another set of high resolution cameras that can be pointed at a fraction of the screen. If the standard resolution cameras would detect a touch, the high-res camera dedicated for this part of the screen would be pointed at the touch area and would return more accurate touch data over time.

## 2.4 Chapter Summary

In this chapter, we presented an overview of current touch detection techniques and showed our solution that provides us with high precision touch input on a large input surface while preserving orientation information. In the next chapter, we will use this system to take a closer look at how people perform touches in general and as part of touch sequences. We will use this contextual information to improve the touch accuracy of the system.

# Chapter 3

# Improving Touch on Tabletops

In the last chapter, we presented our study platform that we will now use to explore how people interact with touch devices. Wang and Ren [2009] found out that people can aim for targets as small as 11mm diameter in a lab setting. If we take a closer look at finger orientation (pitch, roll) and individual aiming behavior, we can get as low as 5mm diameter on small devices (Holz and Baudisch [2010]). But these studies focussed on small devices where only few parameters, such as finger orientation, change in typical use-cases. On a large-scale tabletop, however, other variables could be important, such as body posture and touch gesture. We therefore wanted to see how the location of the target (and thus implicitly the body posture) influences touch accuracy. While observing users, we also found out that the targeting behavior is different for single touches and touches that are part of a touch sequence. We will also show our results on how to exploit this in order to increase the overall input accuracy in the second part of this chapter and thus improve touch on tabletops.

Find out how contextual information of a touch relates to targeting and use it to improve touch accuracy.

## 3.1 Effect of Body Posture on Touch Position

We will first reflect on existing research, then present our study hardware and design, and closeup with our results that show how the body posture induces targeting errors.

### 3.1.1 Related Work

A lot of work has been done to understand and predict human targeting performance. Performance can be measured in error rate, accuracy, or speed, but these are all related as pointed out, e.g., by Zhai et al. [2004]: More error means less accurate input, and vice versa. Also, if a person tries

Targeting error and times have been extensively studied in the psychologies.

to aim very accurately, he needs to take more time on the task, so there is a trade-off between the two. This relation between the difficulty of the targeting task and its completion time was made prominent by Fitts [1954] who created a prediction model for the targeting time given distance to the target and its size. His study design was limited to a 1D-tapping task, and he used real pen and paper. Numerous papers followed that analyzed it in different contexts such as by MacKenzie and Buxton [1992] for a 2D tapping task, the work from Murata and Iwase [2001] on a 3D tapping task, and also for different hardware, e.g., the PC mouse by Murata [1996].

Insight about effects of the body posture on targeting are missing.

Interestingly, after about 60 years this is is still a very active research topic. We think this is due to the fact that the results help us to get a better understanding of human behavior, which in turn can be used to inform our interaction design. Another reason is that it is quite a complex topic as numerous factors are likely to have an impact on the targeting behavior and thus speed and accuracy. Examples include orientation and position of various body parts and how they change over the course of the interaction.

### 3.1.2 Study Setup

Our research goal challenges us to find a good mapping between touch inaccuracies and input context, which we considered to be the upper body posture and relative location of a person to the touch target.

Users touch random targets on a display.

The general study design was to display randomly ordered targets on the display, let the user touch them, and look for systematic touch behavior w.r.t. the targets. More specifically we wanted to find out whether different targets have an effect on the direction and margin of touch offset.

Additionally, we wanted to find out how people would turn their limbs to touch the targets. We were especially interested in the finger pitch because previous research indicates that it can be used for correcting touch offset (Holz and Baudisch [2010]).

Body posture was captured with marker-based tracking.

To infer the body posture, we made use of the VICON tracking system (Figure 3.1). We placed eight cameras around the table so that it could track a large volume and users were not restricted in their movement. Marker pairs were placed on several joints: both shoulders, the waist, the elbow, the wrist, the back of the hand, the knuckle of the index finger, and the tip of the index finger itself.

### 3.1.3 Task & Procedure

Touch targets were located on a uniform grid.

The task was rather simple: We showed touch targets in a random order on a uniformly spaced grid (5x9) one target at a time, and participants had to touch this target. After the touch, they had to move their hand to a resting position. To make sure that the hand was moved back, the investigator

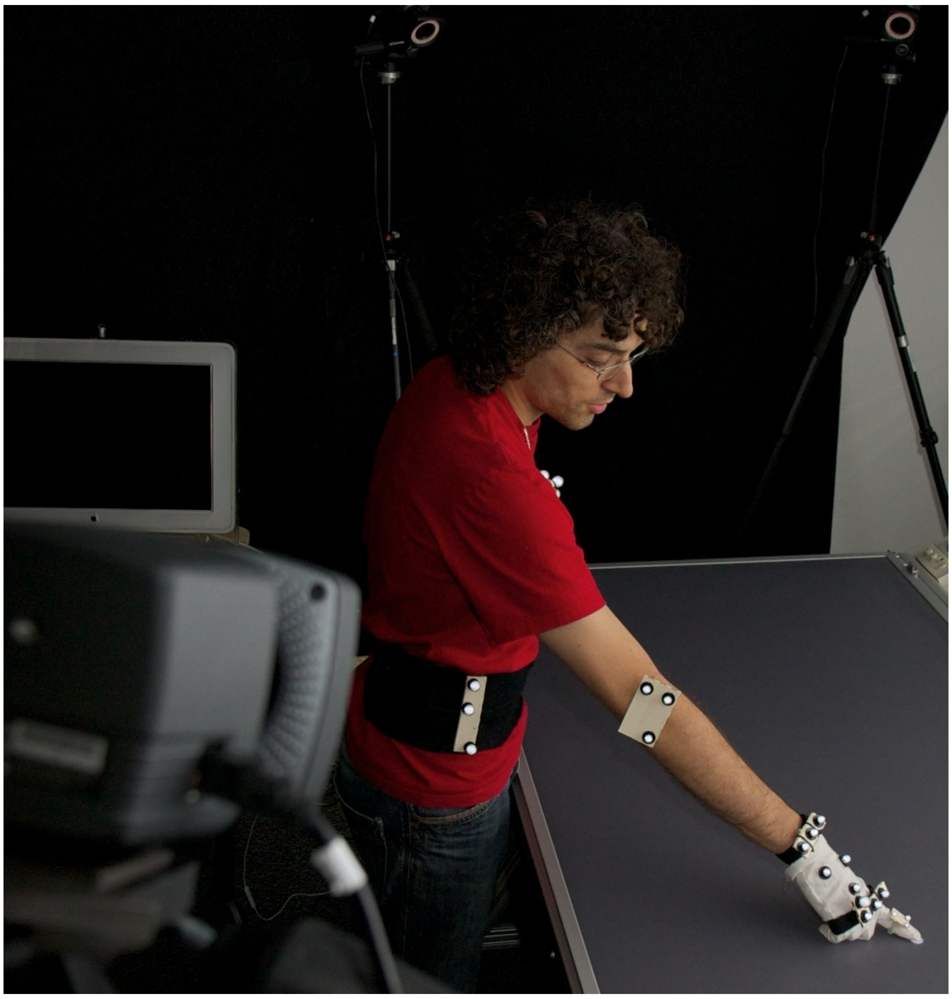


**Figure 3.1:** This is the full construction including the table and the VICON setup.

waited until the participant had done so and only then activated the next touch target. This was done for every of the 45 touch targets, and repeated a total of 10 times, taking about 20 minutes per participant. We finished with a survey regarding their prior touch experience using 5-point Likert scales.

After each user, we recalibrated the system to compensate any unexpected flaw in the camera mount. In addition to the touch offset, we also stored the body posture of the user as 3D coordinates of the marker positions.

### 3.1.4 Analysis

Measured position, orientation, and curvature of the right arm.

Based on the VICON marker positions, we calculated the curvatures of the following joints as the inner angle of a triangle spanned by the two adjacent VICON markers and the joint itself: shoulder joint (waist, shoulder, elbow), elbow joint (shoulder, elbow, wrist), wrist joint (elbow, wrist, knuckle), and knuckle joint (wrist, knuckle, finger tip), and lastly the pitch angle of the

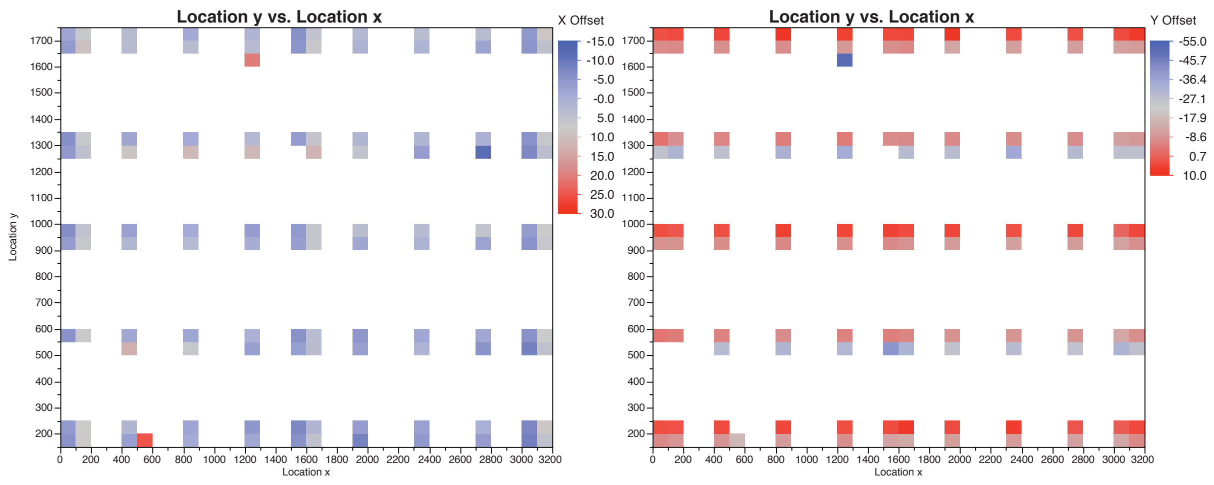


**Figure 3.2:** Top view on the tabletop. The squares represent the means of the offset in x (left) and y (right) direction for each target.

finger (knuckle, index finger, projection of knuckle onto the surface), i.e., how steep the finger is touching the surface.

### 3.1.5 Participants & Results

We performed our study with 10 participants (1 female), age 21–32 ($M = 26.6, SD = 3.4$), height 159–185 cm ($M = 178, SD = 7.9$). They were all students from computer science or related fields and had all experienced touch on smartphones or other handhelds. Six of them also had prior experience with table-sized touch displays.

Height of the participants was not an issue.

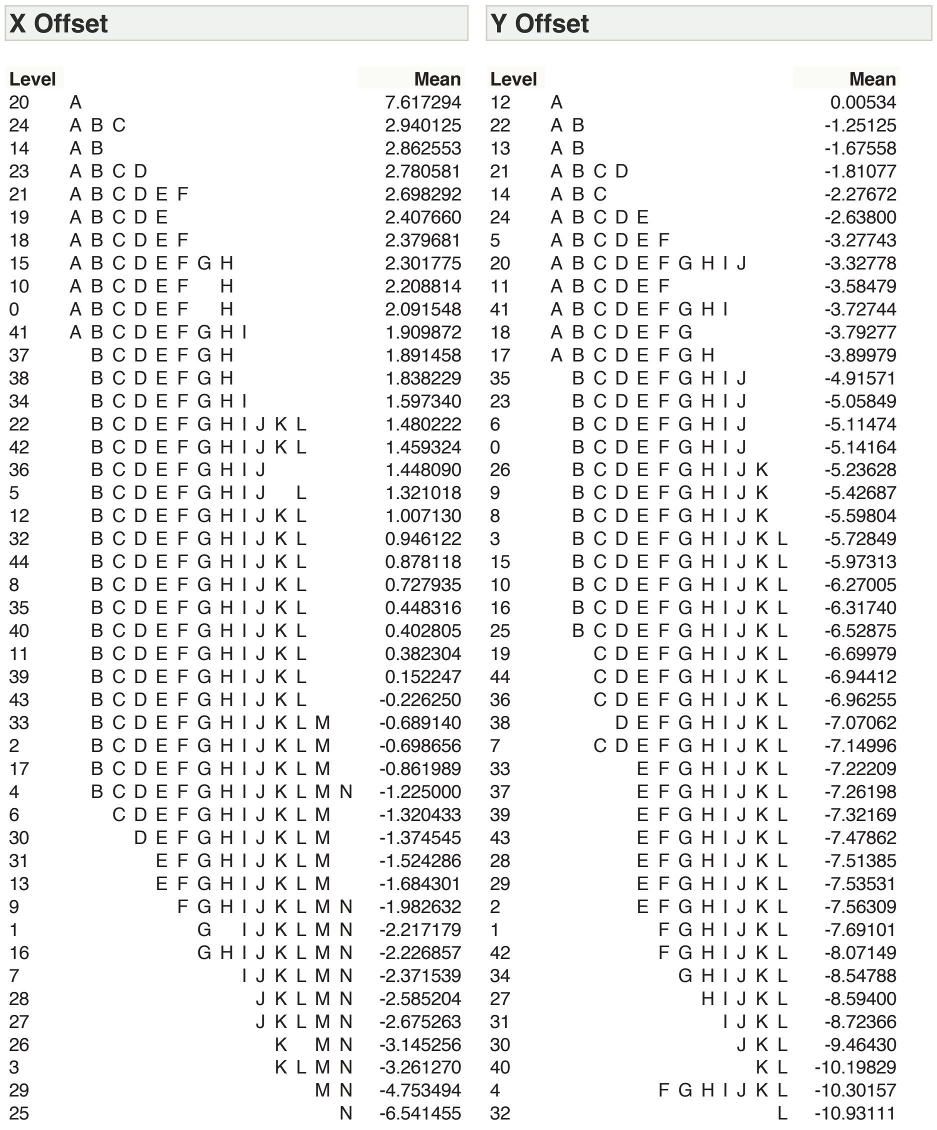
We had concerns how the participant's height would impact the body posture which might lead to different touch angles and thus might impact touch offsets. However, arm length scales with height and seemed to counterbalance this effect: we could not see any effect of height in the data.

The target has a significant impact on the offset.

To see the impact of different targets, we ran a repeated measures ANOVA and saw a significant main effect of the choice of the target on x-offset ($F_{(1,44)} = 12.36$, $p < 0.0001$) and on y-offset ($F_{(1,44)} = 9.61$, $p < 0.0001$). Figure 3.2 provides a visualization of the offset means. The first suggests that in the (top) left area people are slightly drifting to the middle (red), i.e, they are undershooting. In the second (y-offset), we can see that in the second and fourth row of targets people often undershoot.

Top left and top middle targets are touched significantly different than the bottom targets.

For a target-to-target comparison, we used Tukey's HSD. We were able to identify 14 power levels for the x-offset and 12 for the y-offset (Figure 3.3). Targets on the same power level have a similar mean, targets who are not on the same level are significantly different. Figures 3.4 and 3.5 validate the impressions from the previous figures: top left and top middle targets being touched significantly differently than the bottom targets (regarding x) and middle row being touched differently than the others (regarding y).



**X Offset**

| Level | | Mean |
|---|---|---|
| 20 | A | 7.617294 |
| 24 | A B C | 2.940125 |
| 14 | A B | 2.862553 |
| 23 | A B C D | 2.780581 |
| 21 | A B C D E F | 2.698292 |
| 19 | A B C D E | 2.407660 |
| 18 | A B C D E F | 2.379681 |
| 15 | A B C D E F G H | 2.301775 |
| 10 | A B C D E F H | 2.208814 |
| 0 | A B C D E F H | 2.091548 |
| 41 | A B C D E F G H I | 1.909872 |
| 37 | B C D E F G H | 1.891458 |
| 38 | B C D E F G H | 1.838229 |
| 34 | B C D E F G H I | 1.597340 |
| 22 | B C D E F G H I J K L | 1.480222 |
| 42 | B C D E F G H I J K L | 1.459324 |
| 36 | B C D E F G H I J | 1.448090 |
| 5 | B C D E F G H I J L | 1.321018 |
| 12 | B C D E F G H I J K L | 1.007130 |
| 32 | B C D E F G H I J K L | 0.946122 |
| 44 | B C D E F G H I J K L | 0.878118 |
| 8 | B C D E F G H I J K L | 0.727935 |
| 35 | B C D E F G H I J K L | 0.448316 |
| 40 | B C D E F G H I J K L | 0.402805 |
| 11 | B C D E F G H I J K L | 0.382304 |
| 39 | B C D E F G H I J K L | 0.152247 |
| 43 | B C D E F G H I J K L | -0.226250 |
| 33 | B C D E F G H I J K L M | -0.689140 |
| 2 | B C D E F G H I J K L M | -0.698656 |
| 17 | B C D E F G H I J K L M | -0.861989 |
| 4 | B C D E F G H I J K L M N | -1.225000 |
| 6 | C D E F G H I J K L M | -1.320433 |
| 30 | D E F G H I J K L M | -1.374545 |
| 31 | E F G H I J K L M | -1.524286 |
| 13 | E F G H I J K L M | -1.684301 |
| 9 | F G H I J K L M N | -1.982632 |
| 1 | G I J K L M N | -2.217179 |
| 16 | G H I J K L M N | -2.226857 |
| 7 | I J K L M N | -2.371539 |
| 28 | J K L M N | -2.585204 |
| 27 | J K L M N | -2.675263 |
| 26 | K M N | -3.145256 |
| 3 | K L M N | -3.261270 |
| 29 | M N | -4.753494 |
| 25 | N | -6.541455 |

**Y Offset**

| Level | | Mean |
|---|---|---|
| 12 | A | 0.00534 |
| 22 | A B | -1.25125 |
| 13 | A B | -1.67558 |
| 21 | A B C D | -1.81077 |
| 14 | A B C | -2.27672 |
| 24 | A B C D E | -2.63800 |
| 5 | A B C D E F | -3.27743 |
| 20 | A B C D E F G H I J | -3.32778 |
| 11 | A B C D E F | -3.58479 |
| 41 | A B C D E F G H I | -3.72744 |
| 18 | A B C D E F G | -3.79277 |
| 17 | A B C D E F G H | -3.89979 |
| 35 | B C D E F G H I J | -4.91571 |
| 23 | B C D E F G H I J | -5.05849 |
| 6 | B C D E F G H I J | -5.11474 |
| 0 | B C D E F G H I J | -5.14164 |
| 26 | B C D E F G H I J K | -5.23628 |
| 9 | B C D E F G H I J K | -5.42687 |
| 8 | B C D E F G H I J K | -5.59804 |
| 3 | B C D E F G H I J K L | -5.72849 |
| 15 | B C D E F G H I J K L | -5.97313 |
| 10 | B C D E F G H I J K L | -6.27005 |
| 16 | B C D E F G H I J K L | -6.31740 |
| 25 | B C D E F G H I J K L | -6.52875 |
| 19 | C D E F G H I J K L | -6.69979 |
| 44 | C D E F G H I J K L | -6.94412 |
| 36 | C D E F G H I J K L | -6.96255 |
| 38 | D E F G H I J K L | -7.07062 |
| 7 | C D E F G H I J K L | -7.14996 |
| 33 | E F G H I J K L | -7.22209 |
| 37 | E F G H I J K L | -7.26198 |
| 39 | E F G H I J K L | -7.32169 |
| 43 | E F G H I J K L | -7.47862 |
| 28 | E F G H I J K L | -7.51385 |
| 29 | E F G H I J K L | -7.53531 |
| 2 | E F G H I J K L | -7.56309 |
| 1 | F G H I J K L | -7.69101 |
| 42 | F G H I J K L | -8.07149 |
| 34 | G H I J K L | -8.54788 |
| 27 | H I J K L | -8.59400 |
| 31 | I J K L | -8.72366 |
| 30 | J K L | -9.46430 |
| 40 | K L | -10.19829 |
| 4 | F G H I J K L | -10.30157 |
| 32 | L | -10.93111 |

**Figure 3.3:** These are the power levels of the offsets in x and y direction. Levels not connected by the same letter are significantly different.

### Joint Curvature

To better understand why people are over- and under-shooting, we took a look at the recorded body postures.

We can see that people aimed at far away targets as expected: a small elbow curvature and a rather closed shoulder joint (Figure 3.6). When we took a look at closer targets, however, we were surprised: people were opening their shoulder joint wide, and bent their elbow in order to reach spots directly in front of them. This led to quite unnatural wrist curvatures, especially when touching target 29 at $(1900, 550)$ (Figure 3.7). As expected, this is resembled in its rather large offset (Figure 3.3).

Some targets were touched in surprising postures.

The knuckle joint was more dependent on the participant ($\overline{R^2}$ =

Knuckle joint curvature depends on the user rather than the target.

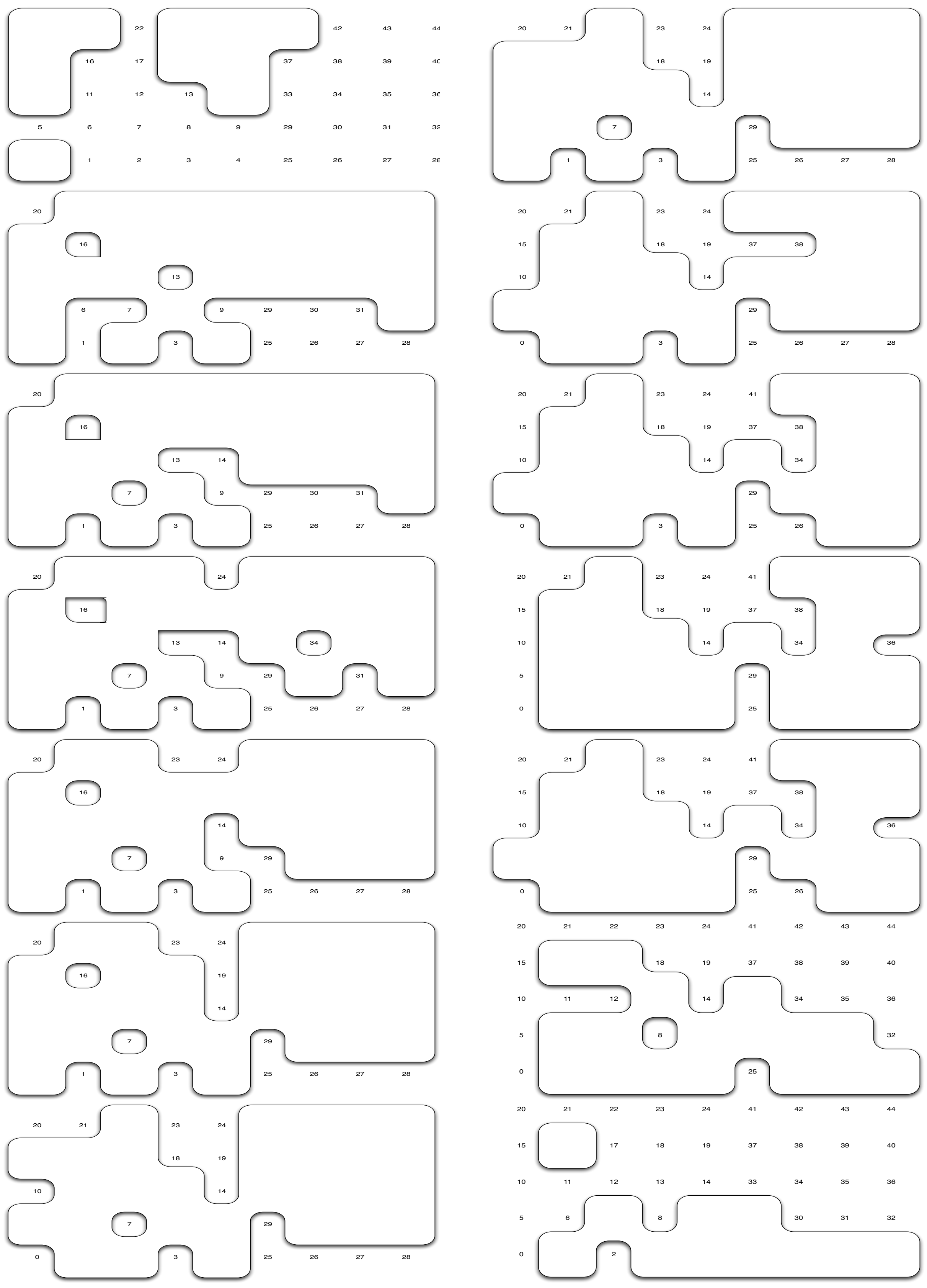


**Figure 3.4:** These are the power levels of the offsets in x direction sorted descending. Each rectangle represents the view from the top of the table. The left column shows power levels A through G, the right column shows H through N. If two targets are encircled on an image, they have a similar mean. If two targets are never connected on any of the images they are significantly different. The top left and top middle area has a big x offset. The bottom area has a rather low x offset.

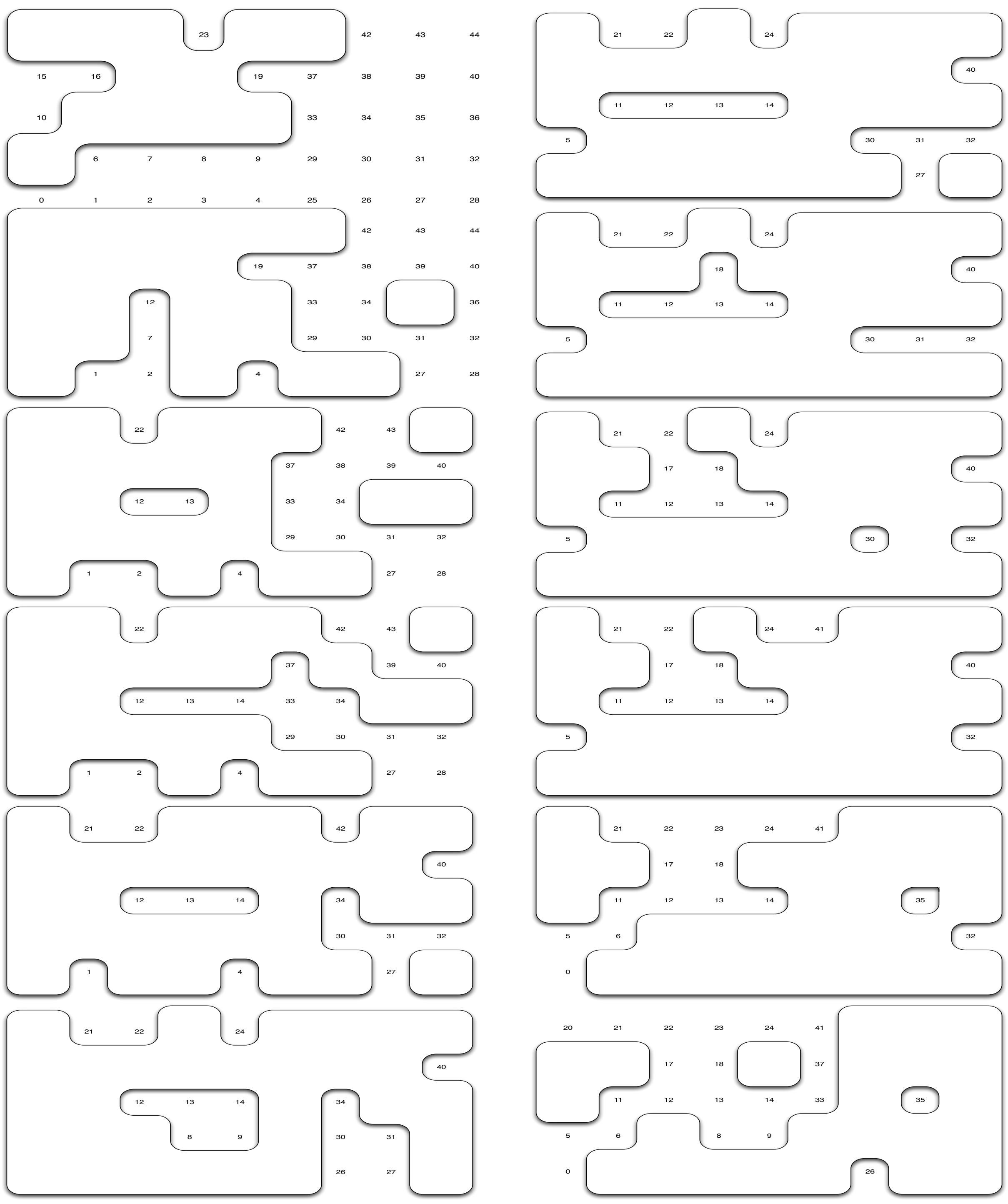


**Figure 3.5:** These are the power levels of the offsets in y direction sorted descending. Each rectangle represents the view from the top of the table. The left column shows power levels A through F, the right column shows G through L. If two targets are encircled on an image, they have a similar mean. If two targets are never connected on any of the images they are significantly different. The top left and middle area has a big y offset. The bottom right area has a rather low y offset.

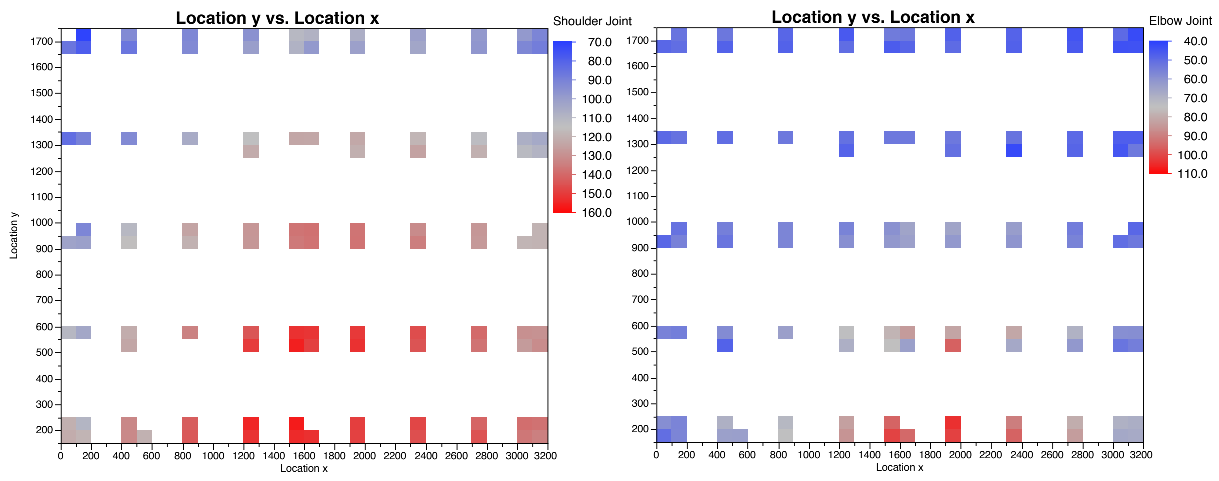


**Figure 3.6:** Top view on the tabletop. **Left:** Shoulder joint curvature. **Right:** Elbow joint curvature.

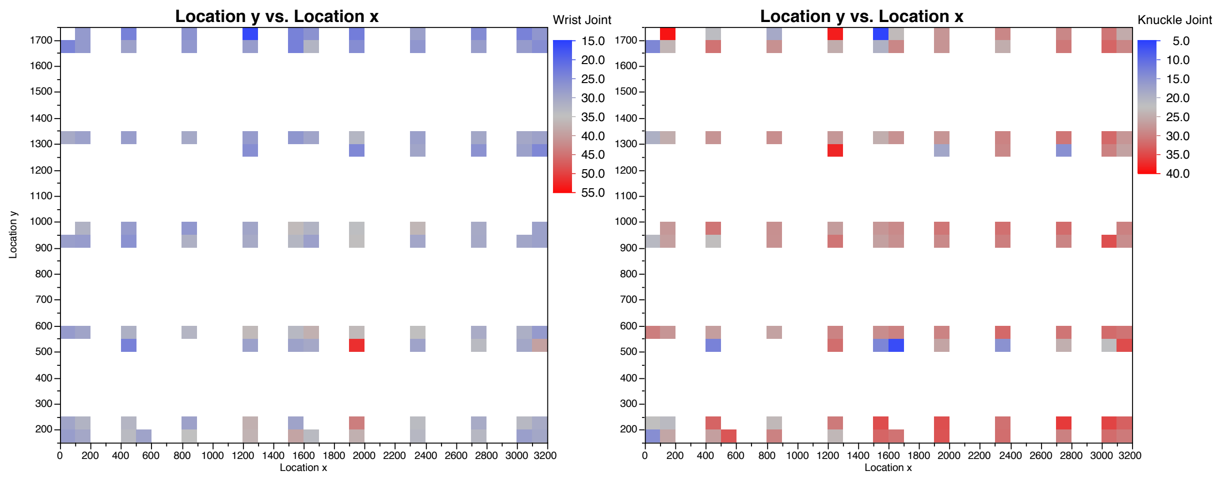


**Figure 3.7:** Top view on the tabletop. **Left:** Wrist joint curvature. **Right:** Knuckle joint curvature.

$.775, F(1,9) = 793.7327, p < .0001$) than the target ($\overline{R^2} = 0.043, F(1,44) = 2.08, p < .0001$) and thus seems to be more of a personal preference. The latter can also be seen in Figure 3.7.

People touch farther away targets with a more steep finger pitch.

Lastly, we did take a look at the finger's pitch, i.e., how steep people were holding their hands (Figure 3.8). We did expect the asymmetric distribution, but we were quite surprised to see people *increase* the pitch angle when touching targets that were farther away. Also, the range of the pitch angle ($M = 59.862, SD = 17.06$) was surprising.

Finger posture cannot be used to correct touch offset.

Previous work by Holz and Baudisch [2010] suggested that we can use pitch, yaw, and roll to correct for systemic touch errors. However, in our data, finger pitch does not correlate with either $x$ or $y$ offset, $r_x(2078) = -0.00611, p = .7808$ and $r_y(2078) = -0.0336, p = 0.1255$. Hence, we think that their findings are restricted to their setup which consisted of a single target on a small surface that drastically limited the likely body postures.

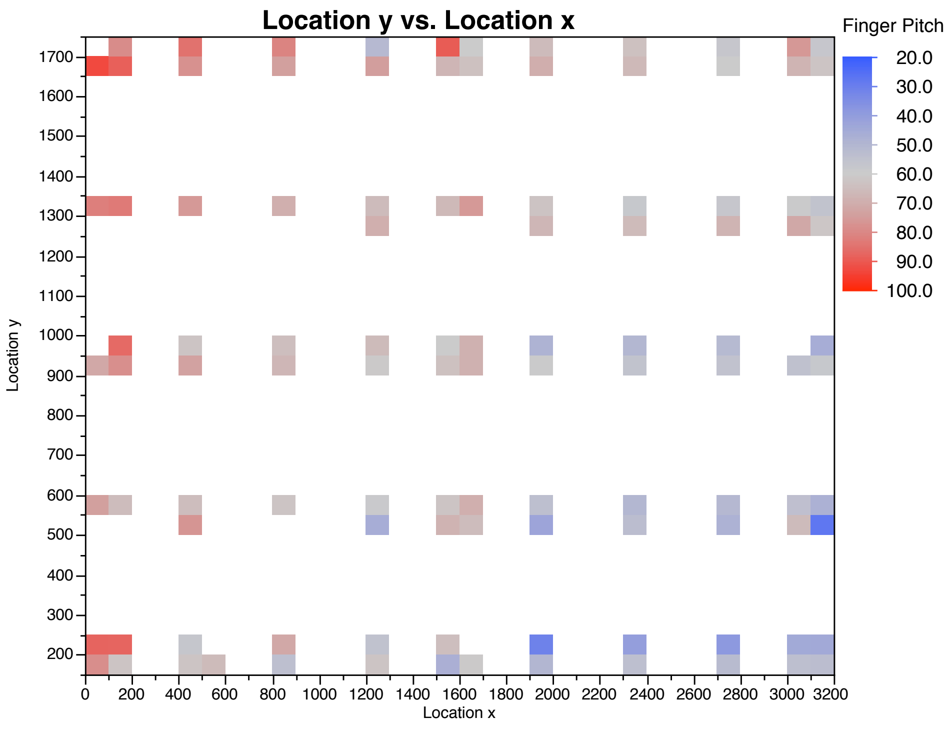


**Figure 3.8:** Finger pitch.

| Predictors | $\overline{R^2}$ |
|---|---|
| target, participant, body posture | 0.326 |
| target, participant | 0.297 |
| participant, body posture | 0.186 |
| target, body posture | 0.1611 |
| body posture | 0.118 |
| participant | 0.1129 |
| target | 0.0961 |

**Table 3.1:** Adjusted r-square values for different models based on the listed predictors. Higher value means that a larger ratio of the variance in the data can be explained.

Conclusively, we do not think that the finger posture is sufficient to correct touch offsets on larger surfaces.

Target and participant have the strongest impact.

To find out which variable has the biggest impact on the offset, we tried different linear regression models to fit our data to the predictors. We used a stepwise forward addition and stopped when reaching the minimum Bayesian Information Criterion to avoid over fitting as suggested by Schwarz [1978]. The results (Table 3.1) show that the target and participant need to be known and that the body posture (as it depends on the target) only adds limited information. Further simplification of the target by, e.g., only taking a look at relative distance or direction from the person to the target misses out a lot of information:$\overline{R^2} = 0.026$ for distance, $\overline{R^2} = 0.005$ for angle, and $\overline{R^2} = 0.049$ for both together.

As future work, we would like to use the impact of target, user, and body posture to the touch offset in order to correct those systematic errors and improve touch accuracy.

### 3.1.6 Summary

Touch offset on large tabletops depend on users and target, not on finger posture.

By analyzing touch offset and body posture of people aiming at random targets on a large-scale surface we did not only learn how people aim at targets, we also found out that restricting posture analysis to the finger's pitch (as recommended by previous work) is not sufficient for predicting touch error on large surfaces. However, we saw that the error significantly depends on the target (and the user). We interpret this connection to be partly caused by the different body postures resulting from the different targets and also likely to be influenced by different occlusion of the touch target by the user's hand.

Now take a look at predecessor of the touch.

To rephrase our findings: People touch with systematic error when they have a variable target and a fixed origin, i.e., hand resting position. The next step is to find out whether a fixed target would be touched differently depending on different origins. It could be that, e.g., movement inertia results in systematic over- or undershooting of the target. It could also be that succeeding touches impact the offset because people tend to plan movements ahead. Thus we could expect systematic errors throughout touch sequences.

## 3.2 Touches in Touch Sequences

Existing research and our previous work tell us about accuracy for a single touch. However, while observing actual user behavior, we saw people rather perform short sequences of touches with a mix of accuracy and speed as goals. We therefore wanted to look how people perform touch in these scenarios and whether, e.g., the location of the previous touch has an impact on the orientation of the finger at the next touch. The following section will present work that has been carried out with the Master student Norbert Dumont. Parts of it have been published at CHI 2013 (Möllers et al. [2013]), and more detail can be found in the master thesis of Norbert Dumont [2012].

Bad touch recognition often leaves users with frustration because long distances and small targets make touching hard (Fitts [1954]). One can increase the target size to compensate for this effect. Another option which even preserves the scarce interaction area, is to correct the user's input for errors. Existing research already tells us that we can do this for single touches on small surfaces (Holz and Baudisch [2010]), yet the correction of touch (sequences) on large-scale surfaces has not been investigated.

In this section, we will first take a look at the related work, present our

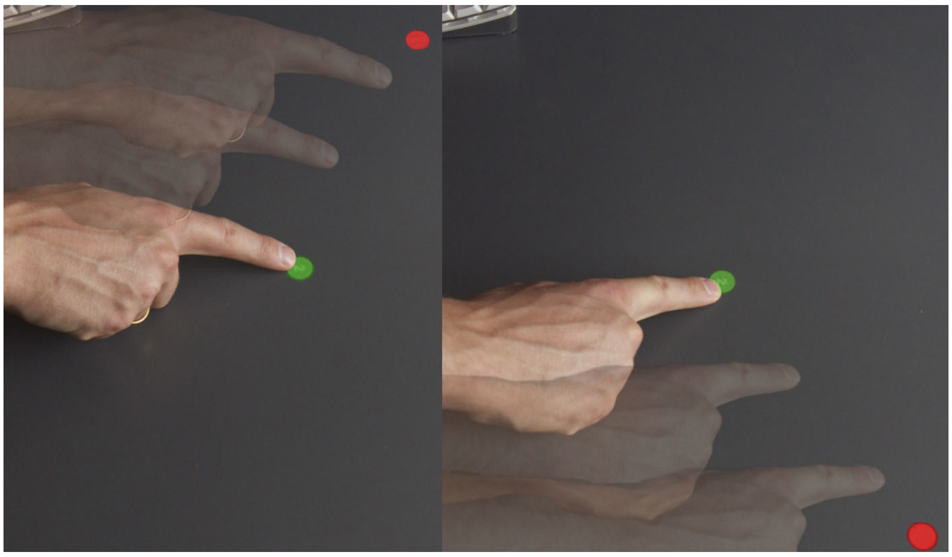


**Figure 3.9:** A person hitting the same (green) button, coming from different directions. Although the same target, the person hits it differently. Taken from Möllers et al. [2013].

study design for our own tabletop hardware, and show that the predecessor has an impact on the touch offset. We will then move to an off-the shelf tabletop and show that the effect can be exploited to improve accuracy of touch input.

### 3.2.1 Related Work

Indirect manipulation to reduce problems due to occlusion.

A lot of work has been done to improve the accuracy of input and deal with occlusions by the finger. One option is to use indirect manipulation by having an input area act as a proxy for the area which is manipulated. This proxy can be located close to the actual input area, c.f. *cross-lever*, *precision handle* (Albinsson and Zhai [2003]), and *shift* (Vogel and Baudisch [2007]) or farther away, e.g., *back-of-device interactions* (Baudisch and Chu [2009]).

Use finger posture as contextual information.

Another option is to use contextual information of the touch besides the center of the touch: Wang and Ren [2009] also use the contact ellipsis, i.e., its orientation and axis size. TapSense by Harrison et al. [2011] used the sound of the impact of an object to identify the touch. Marquardt et al. [2011] used gloves with fiducial markers on the finger tips, knuckles, etc. to thoroughly identify fingers and their posture in his TouchID toolkit. The *Ridge Pad* by Holz and Baudisch [2010] is able to track yaw, roll, and pitch from the fingerprint and uses this information about the finger posture to increase accuracy.

No research about touch sequences.

Surprisingly, we did not find systems that take touch sequences into account. This makes sense for the more dominate small touch surfaces: We can expect only minimal movement inertia, and the limb postures are very similar. On larger surfaces, however, limb postures can vary a lot, and movement inertia comes into play. This means, different predecessors (of the current touch) could have an impact on the targeting behavior. Similarly, Rosenbaum et al. [1990] analyzed target acquisitions in action sequences, and his results suggested that subjects plan their actions beyond the first grip, anticipating future states. This means, different successors could also have an impact on the targeting behavior.

We will analyze these two effects and show how they can be used to increase touch accuracy.

### 3.2.2 Study Design

People who work frequently with the same application know which input is required for achieving a goal. They turn this into a sequence of actions, which on a touch device usually consists of a sequence of touches. We assume that individuals plan these sequences of touches by predicting comfortable final limb postures similar to the participants in the study from Rosenbaum et al. [1990].

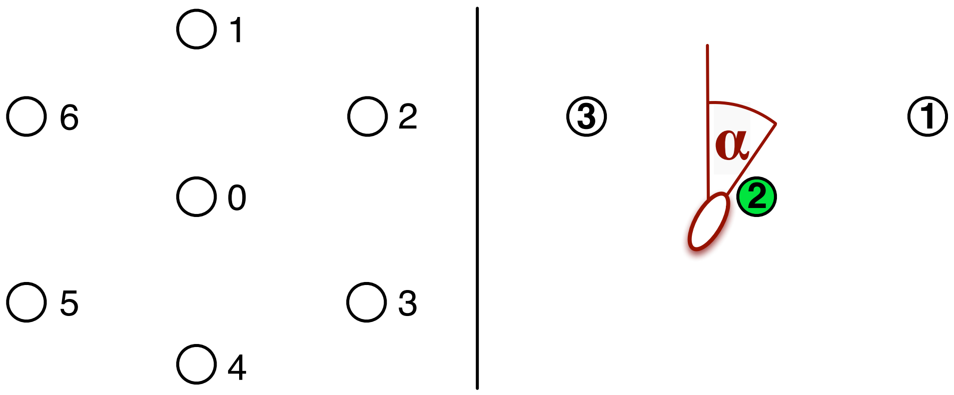


**Figure 3.10: Left:** Layout of the buttons in our study. Most touch sequences started on the ring, went to its center, and back to the ring. **Right:** A participant needs to touch the buttons labeled with "1" to "3": After touching the outer button, he aims at the center button but overshoots (red ellipsis). We record the location and orientation $\alpha$ w.r.t. the y-axis. Taken from Möllers et al. [2013].

Thus, given a touch sequence $[\dots, t_{i-1}, t, t_{i+1}, \dots]$, our hypotheses are:

**H1** Coming from different touch points $t_{i-1}$ changes the angle and offset of the touch at $t_i$. Predecessor Effect (PE).

**H2** Going to different touch points $t_{i+1}$ changes the angle and offset of the touch at $t_i$. Successor Effect (SE).

Consider angle and offset of touch.

In these two hypotheses, we do not only consider the touch offset but also the angle of the touch ellipses (Figure 3.10, right) because we were interested to see how those relate due to the fact that related work showed a significant effect of this angle on the touch offset.

To evaluate our hypotheses, we let people perform touch sequences on the tabletop in a controlled setting and analyzed our data regarding the PE and the SE. We used the tabletop that we introduced in Section 2.2.

We used a 6-ring plus middle button layout for the touch targets (Figure 3.10, left), and participants had to press up to three of them in a sequence. More details about the study can be found in the paper (Möllers et al. [2013]).

The experiment used the following four conditions:

- A first, middle, and last button are displayed simultaneously, i.e., PE and SE should occur.
- B first and second are displayed directly at the beginning, then third after the second has been touched, i.e., PE should occur, but SE not because the participant does not know the successor when touching.
- C first is not displayed at all, middle and last are displayed directly at the beginning, i.e., PE does not occur because there are no different predecessors (the resting spot is always the same), but SE should happen.
- D first is not displayed at all, middle is displayed directly at the beginning, the last after the middle has been touched, i.e., PE and SE should not happen.

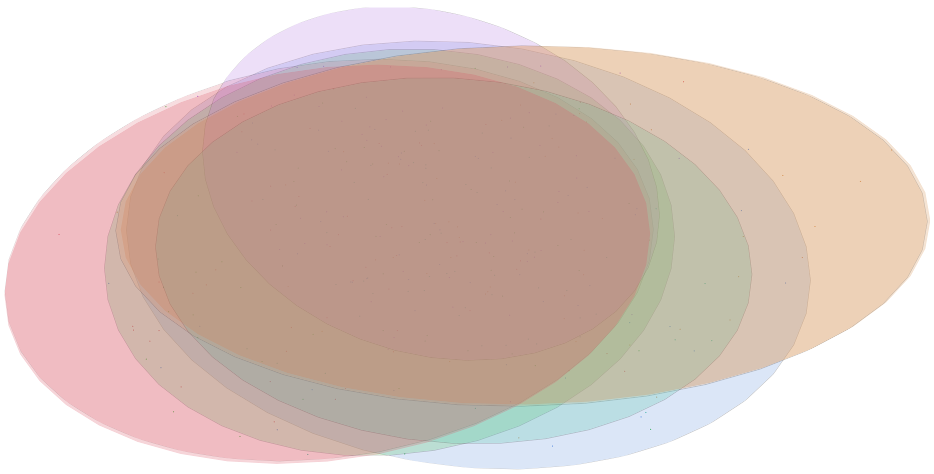


**Figure 3.11:** Offset of the touches. Different clusters represent different predecessors. Although the clusters overlap, there is a significant difference between their center.

### 3.2.3 Results & Discussion

Predecessor effect exists, successor probably not.

As can be seen in Figure 3.11, there exists a predecessor effect, i.e., the location of a previous touch in a sequence has an impact on future touches. But we did not see an effect of succeeding touches (c.f. Figure 3.12). This might be due to the fact that this effect is very small or does not exist. In both cases, we actually welcome this result: We do not need to predict user behavior to get very accurate touches.

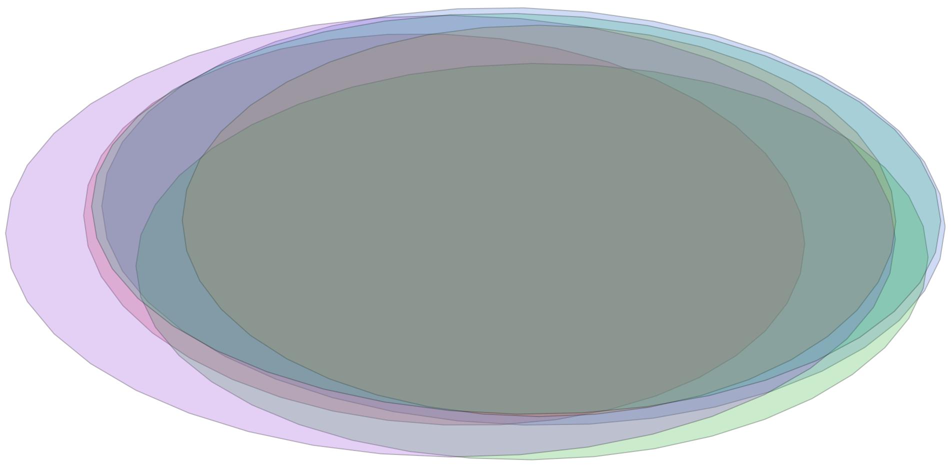


**Figure 3.12:** Offset of the touches. Different clusters represent different successors. The clusters overlap and do not have significant differences.

We cannot use yaw angle of the finger.

We also measured the angle of the offset. Contrary to Holz's work, we saw a very low correlation between angle and touch offset ($|r| < 0.1$). Thus, we cannot use the yaw angle of the finger to correct offset, but instead need to use the predecessor in this setting.

### 3.2.4 Application of the Predecessor Effect

We showed that the predecessor effect exists. We now apply it to improve the touch accuracy using a simple machine learning approach.

Used an off-the-shelf interactive surface.

As additional validation of the predecessor effect and proof of its applicability to hardware setups besides our own, we ran the next study on a different interactive surface. We used a 27" Perceptive Pixel horizontal display with sub pixel accurate tracking on a 110dpi screen.

Users had to touch the center button coming from different predecessors.

Similar to the previous study, we used a 6-ring plus middle button layout, and users touched a button on the ring first and then touched the middle button. After this, they needed to move their hand to a resting position in front of them in the air. We measured the touch offset. This data was split randomly and used in two ways: Two thirds were used to generate a model, the other third was used for evaluation, similar to other machine learning approaches.

The error prediction model works as follows. We correct touches according to which direction they come from because we know that the predecessor has an effect on the touch offset: We take all the touches $\vec{t}$ and put them into 6 buckets according to their predecessor $j$ (cf. Algorithm 1). For each bucket $BUCK_j$, we then take the median $x$ and median $y$ value of the offsets of the touches, resulting in touch correction vectors $\vec{t}_{BUCK_j}$. They indicate how people typically over- or under-shoot.

Touch correction model subtracts the offset typical for the predecessor direction.

**Algorithm 1** Correcting for the current predecessor

```
for all predecessor locations j do                          ▷ Training
    $\vec{t}_{BUCK_j} \leftarrow median(\vec{t_i}), \forall \vec{t_i} \in BUCK_j$
end for
for new touch $\vec{t}$ coming from pred $j$ do             ▷ Application
    $\vec{t'} = \vec{t} - \vec{t}_{BUCK_j}$
end for
```

While using the system, we subtract the correction term according to the model and get $\vec{t'}$. The correction term can also be based only on per participant data to account for individual behavior.

We had several people perform numerous touch sequences resulting in two datasets: corrected for the predecessor as well as corrected for predecessor and user. To compare how well the correction work, we calculate the size of a minimal rectangular button that covers at least 95% of the touches, i.e., its width and height is equal to about 1.96 standard deviations of the mean of the $x$ and $y$ data.

Compare size of minimal rectangular button.

### Results

Only using the predecessor has no effect on the button size (Figure 3.13). Our explanation is that individual differences in touch behavior overshadow the predecessor effect. However, taking individual behavior into account, we can shrink the button by 7%— and we only need to capture five minutes of user interaction. We are quite confident that capturing more interaction data will increase this value significantly.

Predecessor model per user can predict error offset.

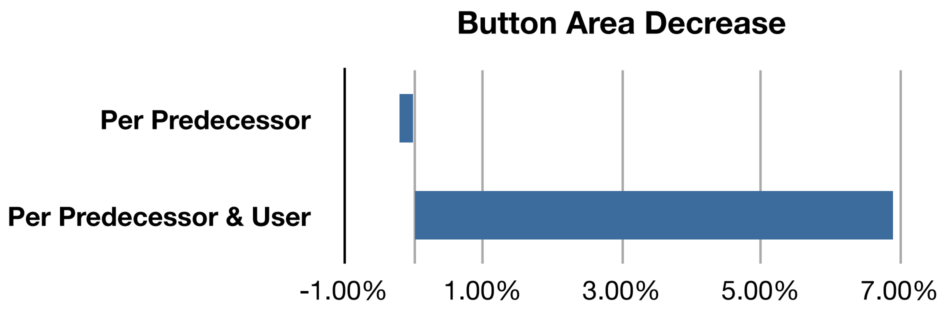


**Figure 3.13:** Decrease of minimum button size to allow for 95% touch accuracy. The baseline was the size of a button using raw touch data. Taken from Möllers et al. [2013].

Real model would learn during usage.

In a real application, we would have to extend the 6-ring to a model that stores offset per relative direction (in degrees) and relative distance to every predecessor. This model would require significant data to be collected. We would like to explore ways to learn this model during the system usage. This is similar to the way current smartphones, which are very personal devices, improve their touch accuracy.

## 3.3 Chapter Summary

Contextual information can improve touch.

In the first part, we saw how people perform touches on large tabletops and how the body posture affects it and that, contrary to the small surfaces, the finger posture is not significantly affecting the targeting behavior. We then evaluated how touch sequences have an effect on the accuracy of their single touches. We saw that the location of the previous touch affects the location and orientation of the following touch. We then applied our knowledge to improve targeting accuracy by taking into account the predecessor of a touch.

Can we extend it?

We thus saw we can use contextual information of a touch to *improve* touch. This means, using more than the traditional touch data, i.e., its center has a benefit. In the next chapter, we will create a new interaction technique by taking into account the head position and thus *extending* touch. The difference here is that the additional information is not only contextual, but adds its own input channel that even can conflict with the touch information. We will show how to handle those conflicts.

# Chapter 4

# Extending Touch on Tabletops

Use head position as additional input channel.

The last chapter told us about how people perform touches and how we can use this systematic targeting behavior to improve touch. After these low-level improvements, this chapter will focus on higher level touch interpretation on the application layer and how we can use the additional input we observed in the previous chapter: We will incorporate more than just the touch point into an interface by using the body posture, more specifically the head position, as an additional input channel. This way we can enhance our interactive 2D tabletop to a 3D display based on a perspective-correct rendering. This gives us a more powerful interface, but also rises some design challenges.

## 4.1 Using Head Position as Additional Input Channel

Motion parallax adds a third dimension.

Until now, our tabletop was used as a horizontal 2D display. If we track the position of the user's head and add motion parallax, people will get the impression that they are looking at a real 3D scene. Moving their head will then update the scene, keeping its perspective sound. Depending on orientation and shape of the surface, metaphors such as a window (vertical display) or a fish tank (horizontal display) come to mind.

Conflicting input.

Interestingly, mimicking physical behavior when manipulating digital objects—the direct manipulation metaphor—does not work too well in such a 3D scene: Direct manipulation means that dragging an object makes the object stick to the finger: However, moving the head in front of such a 3D display results in a change of the object's projection on the display to account for changes in head position. But this means that the projection of the object moves although the finger does not move, and, e.g., might make the object slip off the finger, contradicting the direct manipulation

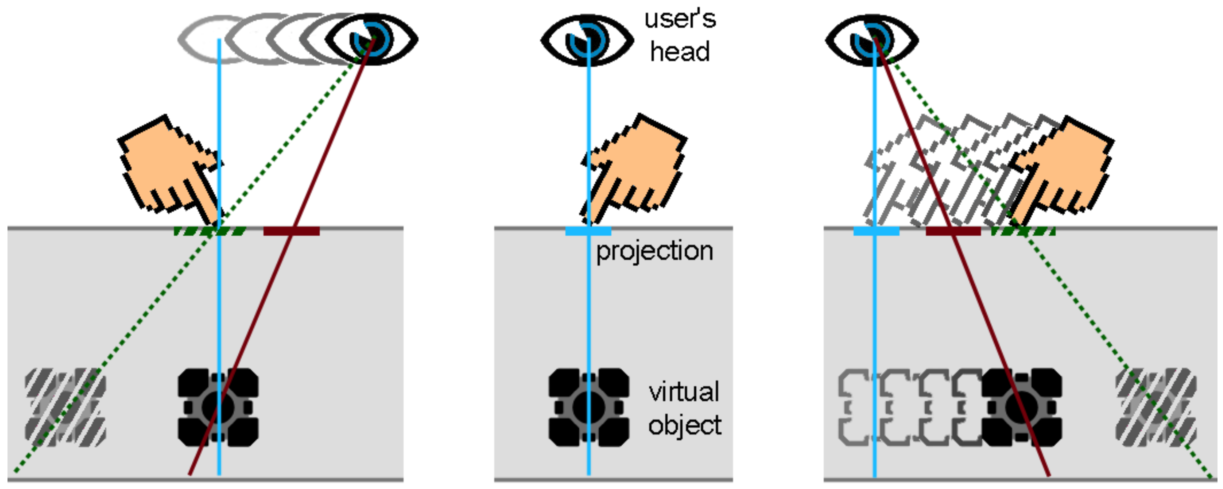


**Figure 4.1:** Co-planar dragging in a naïve fish tank VR implementation. The colored lines show the gaze direction of the viewer, and the bars are the projections of the boxes. Bright blue is before, and dark red after the movement; dotted green shows where the object would look to be “below” the finger from the user's point of view. **Left:** Moving the head does not change the 3D position of the object. **Right:** Moving the finger to the side moves the object equally. In both cases, the projections behave differently than the objects. Thus, the object is no longer below the finger, breaking the direct-manipulation metaphor. Taken from Möllers et al. [2012].

paradigm (Figure 4.1). This means by adding the head position as additional input channel, we now have conflicting input and need to find ways to deal with those problems.

Span design space and analze several of the methods within.

To find solutions for this problem, we analyzed it, created a design space for possible methods, picked several interaction methods from the design space, and compared them regarding accuracy, number of regrabs during interactions, and speed. The following section will present this work and has been carried out with the Diploma student Patrick Zimmer. Parts of it were published at ITS 2012 (Möllers et al. [2012]).

Existing work often comes from VR.

Although previous work has spent considerable time analyzing how the human hand can control 3D objects, it had a different focus. Most of this work comes from the domain of virtual reality where people did not restrict themselves to a 2D input plane (in our case a table or a wall) but allowed the full scale of 6 DOF input: 3 translatory and 3 rotational degrees of freedom for hand input. For example, the Responsive Workbench Agrawala et al. [1997] is a tabletop enhanced with head-tracked stereo rendering using shutter glasses for display and cyber gloves for the interaction. Generally speaking, those 3D enhanced tabletops are called Fish Tank VR systems (Ware et al. [1993]) and usually employ stereoscopy in addition to motion parallax. Others, such as Valkov et al. analyzed how people interact with stereoscopically rendered content Valkov et al. [2011]. Instead, we focus on a viewer-centered projection and omit stereoscopy to isolate us from the effects they found in their studies. More related work can be found in Möllers et al. [2012].

### 4.1.1 Interactions on a Fish Tank

Touch input and head position both change the projections of the objects on the display.

Direct manipulation has proven to be an intuitive metaphor for manipulating content on a touch capable surface displaying in 2D, even if we only take the success of tablet PCs as evidence. However, if we have a system that incorporates the user's head position to render the image on the 2D plane in such a way that the image is perceived to be 3D, we run into problems. This is mainly due to the fact that in this setting, we are not interacting with the object itself any longer but only with its projection on the display. However, the projection can change its size and position although the object itself remains fixed. This means that interactions can create an offset between a finger holding on to the projection and the projection itself.

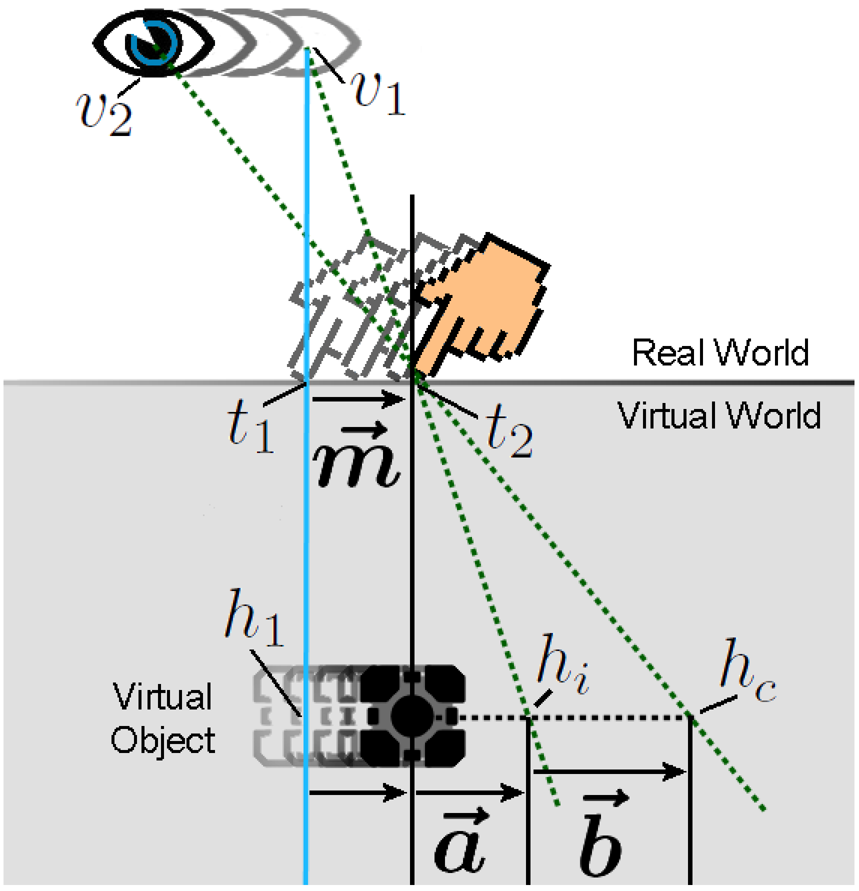


**Figure 4.2:** Offset created by dragging an object in a head-tracked environment. Total offset can be split into a touch-induced offset $\vec{a}$ and a head-induced offset $\vec{b}$. Taken from Möllers et al. [2012].

If we take a closer look, this offset can occur due to dragging or head movement (Figure 4.2). The viewer is looking from a specific point $v_1$, touches the surface at $t_1$, and the virtual object resides at $h_1$. Dragging the object by $\vec{m}$ should move the object by $\vec{m}$ as well. In a 2D setting, we would now be done. However, due the depth of the object, we need to additionally move the object by $\vec{a}$ to keep the object beneath the user's finger. If the head is then moved to $v_2$, we need to move the object again, this time by $\vec{b}$.

Unfortunately, it also means that looking around an object will move the object, and it is questionable whether this is always the expected behavior. We need other methods for those cases.

Nine methods from the design space.

As a more structured approach, we created a design space that differentiates interaction methods by how to deal with offset created by dragging, and offset created by head movement (Figure 4.3). To compensate for offset, we can either move the object, or we can change the viewport by shearing the scene, or do nothing. Theoretically, one could also do both compensations, but we decided against it to not confuse the user. This leaves us with $3 \times 3 = 9$ different spots in the design space, and we will now present these methods shortly. The full implementation details can be found in Zimmer [2011].

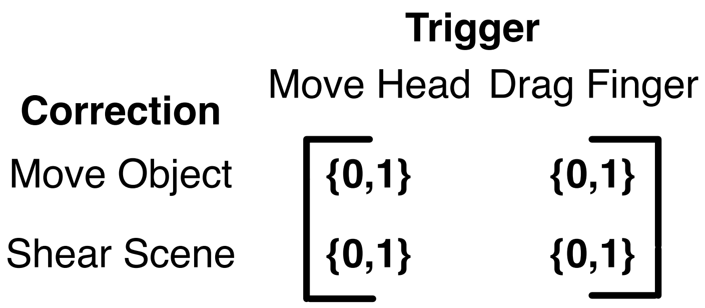


**Figure 4.3:** Design space for perspectively-adjusted direct manipulation. Taken from Möllers et al. [2012].

Their naming is loosely based on the following scheme:

- *Correction* if they move the object to compensate for offset,
- *Shift* if they shear the scene to compensate for offset,
- *Team* if they do both,
- *Adaptive* if offset is mitigated over time,
- *Cross* if the compensations are switched, e.g., head-induced offset is corrected by object movement.

$\left[\begin{smallmatrix}0 & 0\\ 0 & 0\end{smallmatrix}\right]$ Uncorrected

No correction is applied, and the resulting offset is tolerated.

$\left[\begin{smallmatrix}1 & 1\\ 0 & 0\end{smallmatrix}\right]$ ObjectCorrection

This method is also known as Sticky Finger, coined by Bowman [2005]. The object position is determined by the head and the touch position, thus, the object stays exactly under the touch.

$\left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$ AdaptiveCorrection

With AdaptiveCorrection, head movement is ignored and thus does not interfere with object positioning. Dragging the box keeps it precisely beneath the finger. While dragging, we also slowly remove any offset caused by previous head movement.

$\left[\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right]$ SceneShift

This is the first method to shear the scene to maintain alignment, i.e., we move the virtual viewpoint of the scene to avoid the touch slipping off the object. Once an object is released, the scene snaps back. When holding an object in one place, looking around it is impossible because the viewpoint stays fixed in an aligned state.

$\left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$ AdaptiveShift

In an adaptive version of SceneShift, head movement locks the view and aligns the touch. Moving the box detaches the alignment and permits offset keeping the scene perspective sound. The upside is when releasing an object the scene is unlikely to snap back. When the head is moved, existing offset is slowly diminished into a state of alignment.

$\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$ TagTeam

In TagTeam dragging, we combine two different correction methods. The offset caused by moving the finger is corrected by object repositioning, which makes the box stick to the finger. Head movement shears the scene, thus looking around a still object is not possible as the viewpoint is locked to alignment.

$\left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right]$ CrossTeam

In the three cross modes, we switch assignments between trigger and correction method. Here, in CrossTeam dragging, head movement is compensated by repositioning the box, and dragging offset is avoided by shearing the scene.

$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$ CrossCorrection

The CrossCorrection method allows offset when an object is dragged but reduces offset whenever the head is moved. When the user moves her head, it repositions the object based on the head and touch position.

$\left[\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right]$ CrossShift

The CrossShift method shears the scene automatically to maintain touch alignment while the object is dragged. When the object is held still, the user can look around.

### 4.1.2 Comparison of Methods

To analyze our methods, we created a test application and ran a study. The test application consisted of a maze where the user had to drag a box on several floors.

Sorted out the "cross methods" in a pre-study.

During the pre-study, the three cross methods consistently received bad feedback. The users were unhappy with "head movement causing collisions", "need to release the object to safely look around", and "unexpected jumping of the box after releasing it". This led to confusion and frustration. Consequently, we ran the study with the remaining six methods: UNCORRECTED, OBJECTCORRECTION, ADAPTIVECORRECTION, SCENESHIFT, ADAPTIVESHIFT, and TAGTEAM.

As our test apparatus we used the touch capable tabletop known from the previous chapter and added a head tracking solution (Figure 4.4). Head position is converted to the coordinate system of the table and the projection updated accordingly. This way, we can create the impression of the table being a 3D fish tank.

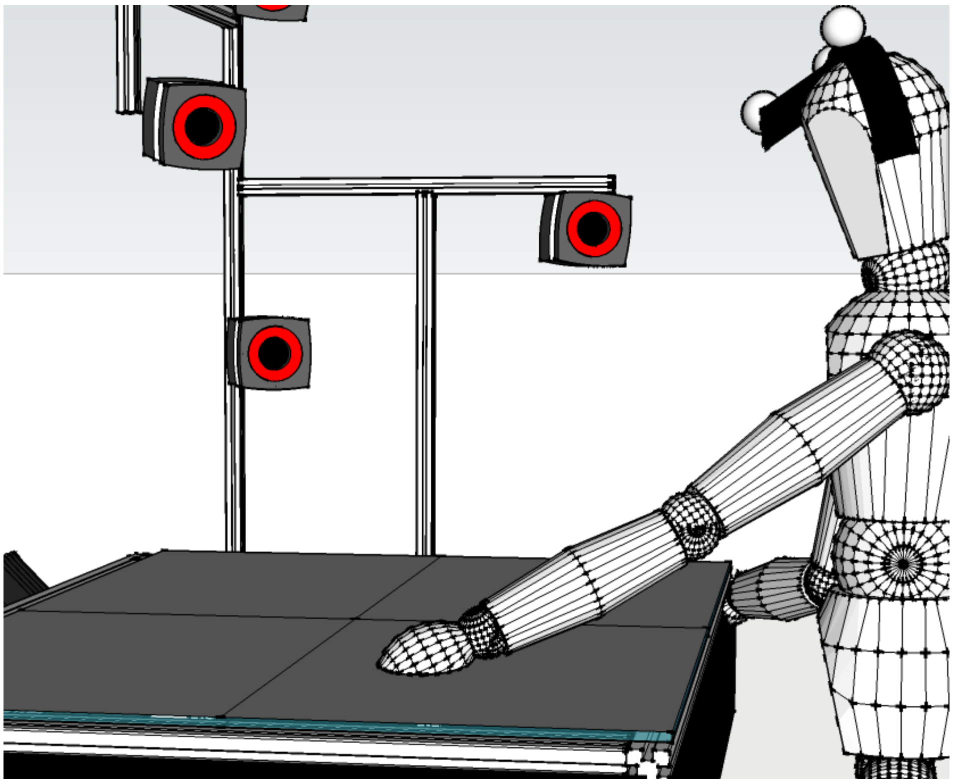


**Figure 4.4:** Infrared cameras track the four markers on the circlet on the user's head. Taken from Möllers et al. [2012].

#### Task

Move a box through a maze.

Before the study, participants had to drag a box around for two minutes in a 2D dragging setting. Then, a method was activated, and the participants could test it in a single 3D level as long as they liked. For the test itself, the participants needed to drag a cuboid through a three floor maze (Figure 4.5). This box can be controlled via two handles at the front and the back (Figure 4.5), using single touch interaction. The movement of the box is subject to a physical engine, taking into account occlusions with the wall. This allowed the user to pivot the box around the corners. Still, we did not add inertia to no make the task (unnecessarily) hard, so the box stopped as soon as the user lifted his finger.

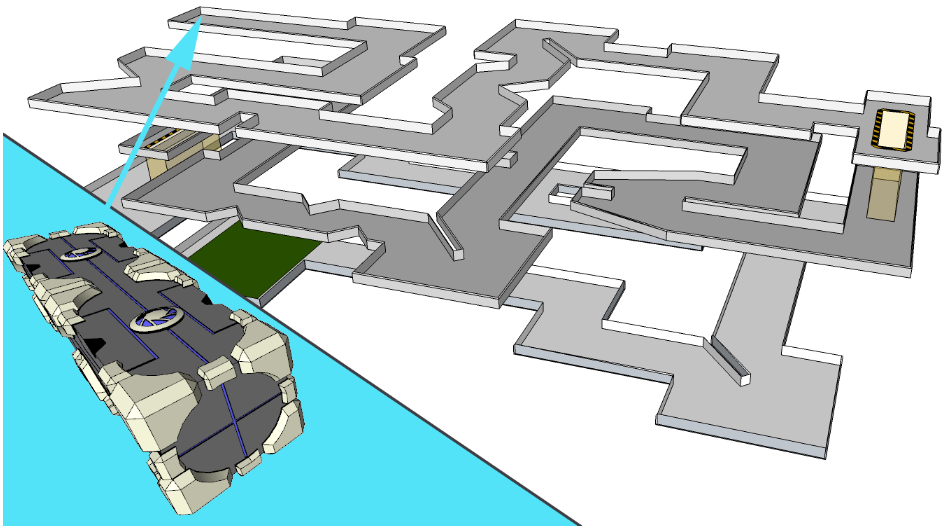


**Figure 4.5: Upper Right**: Participants had to drag the box through the maze, starting from the far left corner on top level to the green field on the bottom level. **Lower Left**: The box had two handles to control it. Taken from Möllers et al. [2012].

Participants started on the top level and moved to a marked area, thereby proceeding to the next floor. The upper levels did not disappear, and thus participants had to additionally deal with occlusion.

Between-subject design

We used a between-subject design with three repetitions and measured completion time, number of occlusions, and number of re-grabs. We also collected qualitative data based on our observations and a post-test questionnaire. More details about the setup, task, and procedure can be found in our ITS paper Möllers et al. [2012].

### Results

The results of our study, i.e., means and confidence intervals of our three metrics, can be seen in Figures 4.6, 4.7, and 4.8. All the results we present here are significant (**$p < 0.05$**), and most of them even highly significant (**$p < 0.0001$**). The full statistical details can be found in Möllers et al. [2012].

Time and number of re-grabs correlate.

First of all, we can see that completion time and the number of re-grabs

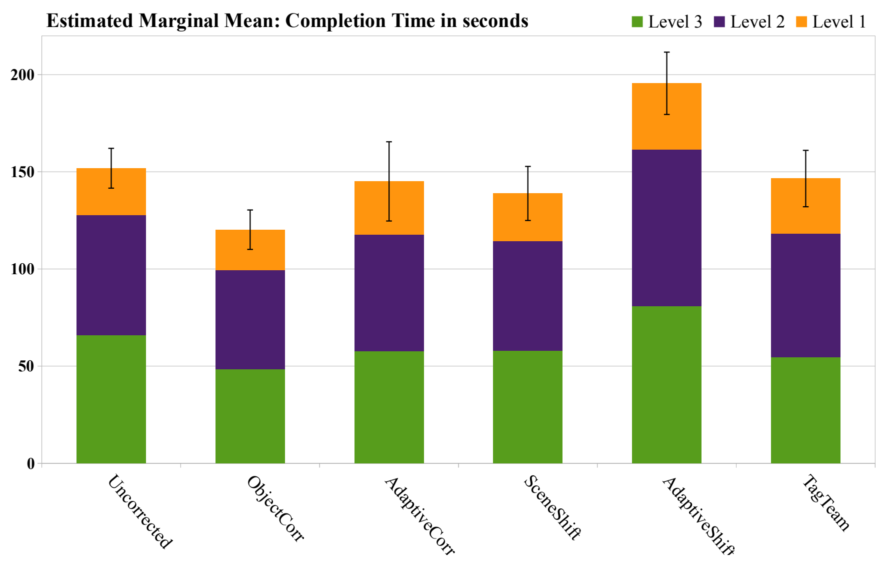


**Figure 4.6:** Mean completion time in seconds. Mean bars are subdivided into single levels. 95% Confidence intervals are shown for the sum over all levels. Taken from Möllers et al. [2012].

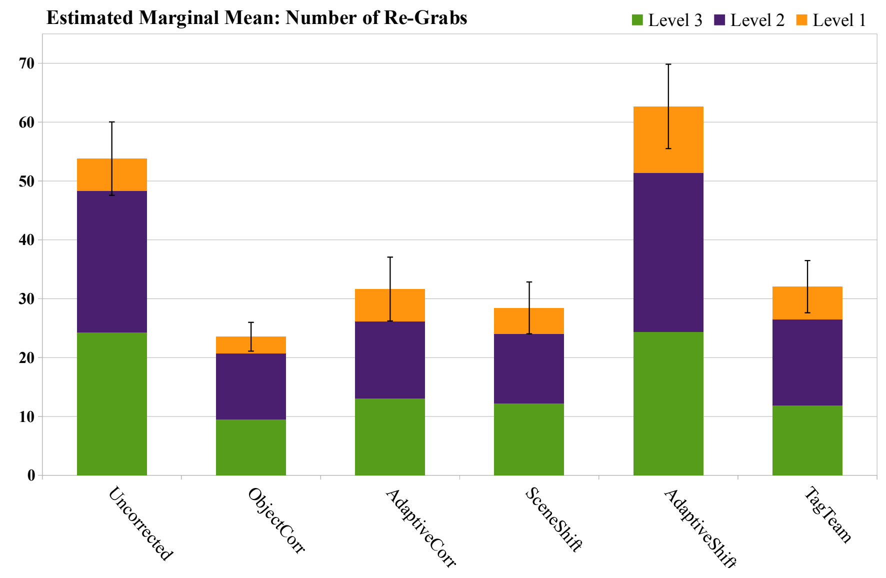


**Figure 4.7:** Number of re-grabs. Mean bars are subdivided into single levels. 95% Confidence intervals are shown for the sum over all levels. Taken from Möllers et al. [2012].

are connected. Clearly, a re-grab takes time. Still, after a re-grab occurred, the offset is removed, thus users can control the box more precisely. This might speed up the time. As we saw a strong correlation between re-grabs and time ($\boldsymbol{r(89) = .79}$), we think the former effect is the stronger, yet one should not deem any interaction method that requires re-grabs useless.

For the remainder of this chapter, we will refer to UNCORRECTED as our baseline because it is the simplest of all the methods, and others build upon it. Also, no clear "standard" interaction method currently exists in the field.

AdaptiveShift was the slowest method.

Compared to the baseline, OBJECTCORRECTION was 27% faster and led to 56% fewer re-grabs (Figure 4.6,4.7). However, many users stated confusion and frustration with OBJECTCORRECTION due to (a) their head movement causing collisions, (b) the need to release the object to safely look around, and (c) the unexpected jumping of the box shortly after releasing it. ADAPTIVESHIFT performed worst and thus should be avoided. Using the baseline method resulted in many re-grabs, but users did complete with average times. It might be that the re-grabs that occurred in this method did not take too long to perform.

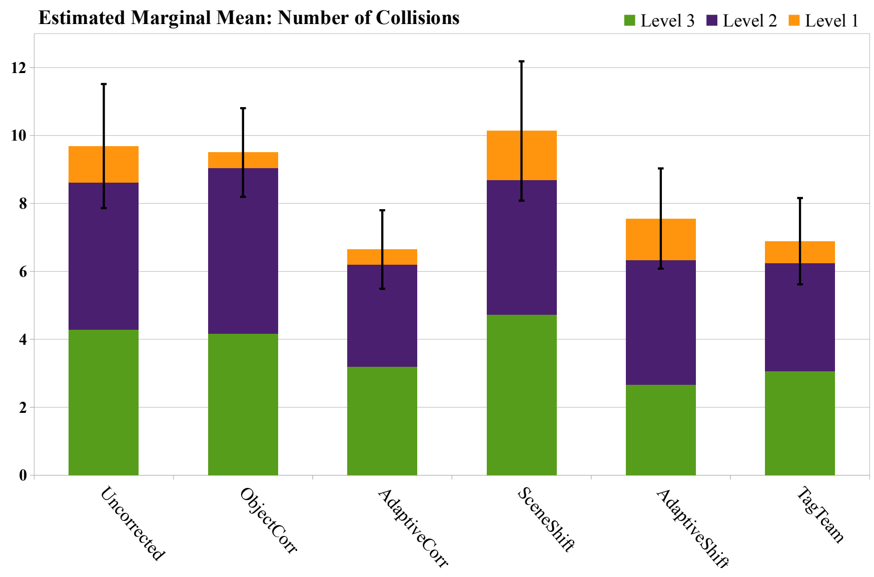


**Figure 4.8:** Number of collisions. Mean bars are subdivided into single levels. 95% Confidence intervals are shown for the sum over all levels. Taken from Möllers et al. [2012].

Number of collisions increased with the depth and vary greatly with the level.

The number of collisions increased with the depth and vary greatly with the level due to the amount of occlusion and interaction-depth. It also primarily occurred at bottlenecks and curves (Figure 4.9). SCENESHIFT and UNCORRECTED performed badly overall (Figure 4.8). OBJECTCORRECTION had few collisions on the first level but did have problems with lower levels due to the aforementioned implicit box movement. Interestingly, ADAPTIVESHIFT method rose from the worst on the first level to best on the last level. Yet, participants criticized the view snapping back, classifying the method as precise but weird. ADAPTIVECORRECTION performed great among all levels (32% fewer collisions), closely followed by TAGTEAM. Both methods offer direct control. While TAGTEAM is restrictive about object-finger alignment, ADAPTIVECORRECTION allows to break it in case the user just wants to look around a bit.

We suggest to use AdaptiveCorrection.

We suggest to use ADAPTIVECORRECTION: Users were rarely confused or

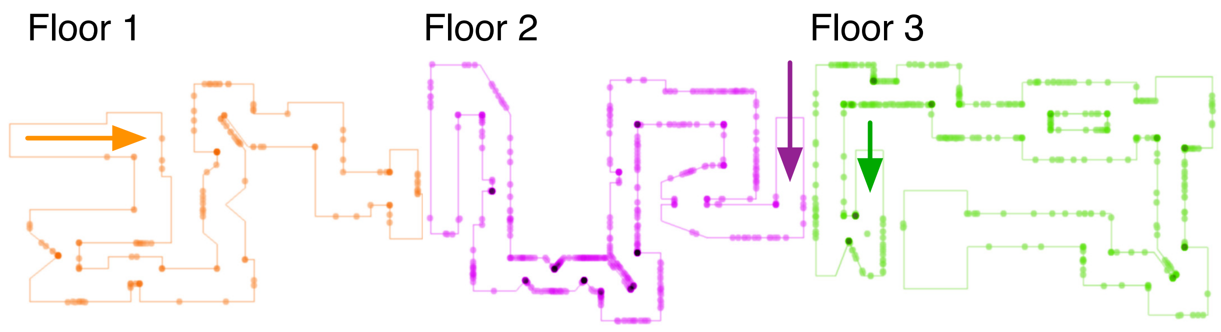


**Figure 4.9:** Collisions for each level across all methods. They primarily occurred at bottlenecks and in curves.

surprised, and it was accurate in the studies. Only one method was faster, OBJECTCORRECTION, but not even with a significant difference. Additionally, ADAPTIVECORRECTION requires few re-grabs, which might be helpful in certain domains, e.g., for handicapped users. ADAPTIVECORRECTION works well because it takes care of two main problems we identified: the perceived offset between touch and object projection, and head movement inferring object translation.

### 4.1.3 Future Work

Take a look at multiple fingers or stereoscopy.

This work led to some first insight into the design space of interaction methods for dragging in 3D environments but could be extended by using multiple fingers for single object control, using other tasks that shine more light on the impact of occlusion, or add stereoscopy to our system setup. Another option would be to switch methods based on the depth or occlusion levels. This would raise the question how one would easily move from one to the other method: implicitly or explicitly.

## 4.2 Chapter Summary

Extended touch provides a more natural input.

In this chapter, we used the head position in addition to the touch data and were able to create a 3D display, but ran into design challenges as direct manipulation and head-coupled projection both interact with the projection of the 3D content at the same time, and we needed to find good ways to prioritize one over the other. In other words, extending the touch so that the touch itself is only one part of the input creates new interaction metaphors. These new interaction metaphors feel more natural to the user, but require a little afterthought to handle this concurrent input so that the user gets the best benefit out of it.

With this example applications for a more holistic use of touch data and its context, we now move from interactive tabletops to interactive objects that can have any shape or form, thereby leaving the desktop behind even more. Similar to the first three chapters, we will split the following three in a “detect, improved, and extend” manner.

# Chapter 5

# Detecting Touch on Arbitrary Objects

To understand how people touch arbitrary objects and how we can build new interaction techniques, we first need to have a system to detect touches on an arbitrary object. This chapter will present our solution to this problem. This work has been carried out with the Master student Ignacio Avellino. More implementation details can be found in his thesis (Avellino [2013]). Part of this work has also been presented at ITS 2013 (Corsten et al. [2013]).

## 5.1 Requirements & Related Work

Focus on non-planar surfaces.

Since we want to detect touches on arbitrary objects, we need to have sensor hardware that can be used for any shape of object and then interpret the data to find the touches. Smartphones, tablets, and interactive tabletops of varying sizes already cover most of the design space for planar surfaces, so we will focus on non-planar objects for our system. As another simplification we are only considering objects that are portable by a human being. Based on this subset, we now define requirements for the process of turning an object into an interactive surface:

1. It should not take too long to prepare one object to allow for studying several objects.
2. It should be affordable.
3. If it is vision-based, it should not require some kind of markers because markers can be occluded by the user interacting with the object. Markers can also change how people interact with an object, thus results about targeting behavior would be questionable.

Current solutions are not flexible enough.

A desired solution would be to have some kind of “magic” tape that you can glue onto any object and that is receptive to touch. In May 2013, Wu et al. [2013] presented very promising research of a flexible foil that can detected touches, but this technology is still in an early stage. Most mobile phones nowadays use capacitive sensing to track touches, but commercially available solutions are not flexible and cannot be cut to size. Instead, Wimmer and Baudisch [2011] used capacitive coupling to detect where a single wire is touched. This wire can be glued to an object and laid out in patterns to cover the whole surface. Yet it only supports single touch and, more importantly, must be calibrated in a rather lengthy process for each object and user (because the capacitive coupling differs from person to person).

Use depth-cameras for remote sensing.

Instead of modifying the object to be touch receptive, we can sense the touches remotely. Remote sensing can be done using different approaches: In our review, we will focus on vision-based systems because they provide superior spatial resolution compared with other, e.g., sound-based systems. All the latest research in the field of remote sensing uses depth-capable video cameras, typically Microsoft Kinect, because they provide depth information per pixel in addition to color.

Use depth thresholding to detect occluded touches.

This additional information makes it easier to separate objects from the background and recognize them. More importantly, it also enables touch detection on surfaces that are occluded by the object by depth thresholding. This technique was first introduced by Wilson [2010]: If we know that the surface of the object is $\boldsymbol{x}$ cm away from the camera, and we see a finger occluding the surface at distance $\boldsymbol{x+2}$ cm, we can (assuming that the finger is about two centimeters thick) detect the touch without actually seeing it.

Point cloud representation limits interactions to object-agnostic metaphors.

One way to work with the incoming data is to turn the depth plus color images into a 3D point clouds. This means we do not try to understand which point belongs to which object, but just have a large amount of ”particles” in 3D space. This is already enough to mimic physical behavior, and we can make the real objects interact with virtual objects as, e.g., KinectFusion by Izadi et al. [2011] does. However, without understanding which points comprise an object, we are limited to those object-agnostic metaphors. This means, even a simple UI, such as a shirt with interactive buttons that make the system react differently depending on where it is touched, is not possible. For those object-specific interpretations of user input, we need to recognize an object and thus turn the point cloud representation into an object representation. This will allow us to react differently to touches depending on which object is touched.

Having collected our requirements and shown that current related work does not fulfill them, we will now describe our approach. It consists of three steps. Object recognition, object tracking, and touch detection on the object.

## 5.2 Object Recognition, Tracking, and Touch Detection

Single camera leads to occlusion problems.

Before going into detail, we want to mention one design decision: Our system gets data from a single camera. This has two inevitable implications which could be removed in future work by supporting multiple cameras: (i) We will be unable to detect the orientation of objects that appear rotation-invariant from the current point of view, e.g., if the handle of the cup is behind the object, we cannot figure out its exact orientation. (ii) We will be unable to detect touches behind the object. Apart from those two limitations, the system works reliable, and we will now describe how it achieves it.

Mesh as object representation.

The first task when creating an object recognition algorithm is to decide on the digital representation of the object. We decided on a 3D mesh: It can be created digitally—most physical objects that surround us start out as a CAD model—, it can come from laser scans of an actual object, or we can use a depth-camera to create the model similar to the work by Liu et al. [2012]. We chose to laser scan the physical objects because it gives the highest accuracy while being applicable to any object.

For setup, define background plane and scan object.

For the remainder of this section, we will focus on the concept of the algorithm and less on its software implementation. The reader may refer to Avellino [2013] for those details. As a preparation, we create meshes for the objects that we want to track, setup the camera, and then select three points on the background. This defines a plane, and we can remove points in the vicinity to the plane. Less points means significantly faster processing times.

Before going into the specifics, we first sketch the overall algorithm (c.f. Figure 5.1):

1. Raw data is acquired from the depth-camera.
2. We created a point cloud from the depth data by unprojecting the pixel-based information into 3D space.
3. We (optionally) remove the background plane to reduce the number of points to process.
4. We use our object recognition algorithm to get an initial pose estimation.
5. The object is tracked from one frame to the other and its position and orientation updated.
6. If a finger is in front of the object, we perform a way of depth-thresholding to detect a touch on the object.

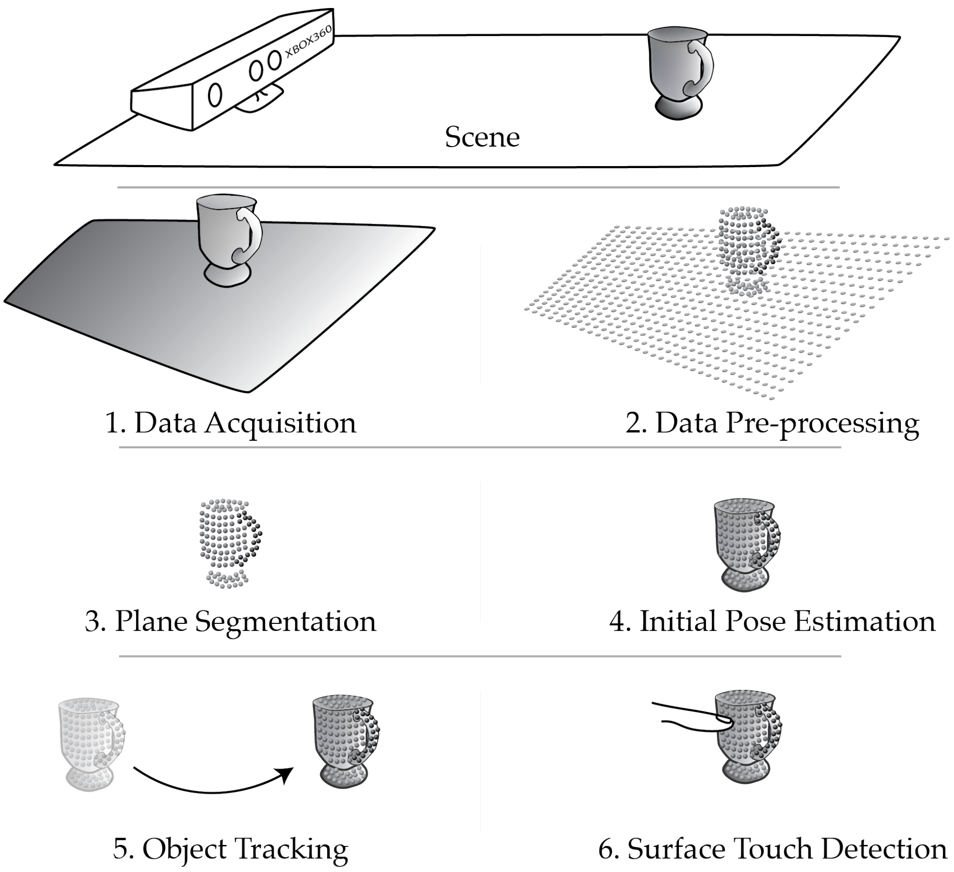


**Figure 5.1:** The whole object recognition, tracking, and touch detection algorithm. Step 4 is done only once at startup. Image adapted from Avellino [2013].

### 5.2.1 Initial Pose Estimation

Use local features to counter occlusion problems.

In order to track an object, we first need to recognize it from the images that our camera delivers. As we already mentioned, we only have one camera, so the object will only be partially visible. It can also be occluded by the user. Therefore, the object recognition needs to be robust against occlusions. This means, our object recognition should not rely on global features such as size of the object and instead should focus on local features such as curvature along its surface.

Fortunately, object recognition is widely studied, and we can modify an existing method to our application domain. We chose an object recognition algorithm that is tailored to noisy single view scan data, similar to our problem domain. It is capable of finding existing 3D-models in an unsegmented point cloud. The standard recognition algorithm works roughly as follows, more details can be found in Papazov and Burschka [2010]. In an *offline* phase, for each model, it takes every pair of oriented points that have a specific distance $\boldsymbol{d} \pm \boldsymbol{\delta}$ to each other and calculates the following features: angle between the orientation of the points, angle between one orientation and the connecting line, and the angle between the other point's orientation and the connecting line. For future lookup, the points, their feature descriptor, and the model they belong to are stored in a three dimensional hash table. In the *online* phase, the point cloud from the depth camera is converted to an octree for faster neighborhood search in future steps. To create a hypotheses, i.e., a guess which object could be at which point with which orientation, we first take one sample point from the point cloud. Then we take some neighboring points, and calculate the features from these pairs. Using these features, we can take a look into the hash table to find point pairs from the models that have similar features, i.e., look similar. We then compute a rigid transform that best aligns the models to the oriented point pair and add this to our list of hypotheses. After a sufficient amount of hypotheses has been created, we start filtering based on how well the hypothesized model locations fit the rest of the point cloud.

The initial pose estimation is quite robust, yet slow. On our MacPro with 2.26 GHz and 6GB ram it takes up to 10 seconds to find the object. Based on this robust recognition, we perform the tracking with a more light-weight algorithm that is capable of running in real-time.

### 5.2.2 Object Tracking

Use ICP to track object from to frame.

In our application scenario, it is very unlikely for objects to disappear or to be moved very far from one frame to the other. This gives two advantages: (i) We can limit the search volume for the object in the next frame by creating a bounding box around the last know position, and (ii) instead of running the full object recognition every frame, we just track the object using Iterative Closest Point (ICP) by Besl and McKay [1992]. Iterative Closest Point (c.f. Figure 5.2, Step 5) is an algorithm that maps two point clouds (a model and a template) to each other while minimizing point distances in each step until a convergence criteria (here distance) is reached. It then returns the mapping that moves and rotates the template on to the model.

As the ICP works best for similar point clouds, we need to prepare our data before applying ICP (c.f. Figure 5.2):

1. We start out with knowledge where the object was in the last frame, i.e., a rotation and translation that maps our stored model into the 3D world coordinates of the camera system. For the first frame, we use the object recognition mentioned beforehand.

2. The object moves.

3. We now get new input from the camera and transform it into a point cloud and create a bounding box around the previous location and assume that the object did not move too far from it.

4. (a) We remove the background plane, use color filtering to remove hands that might occlude the object, and crop the points outside

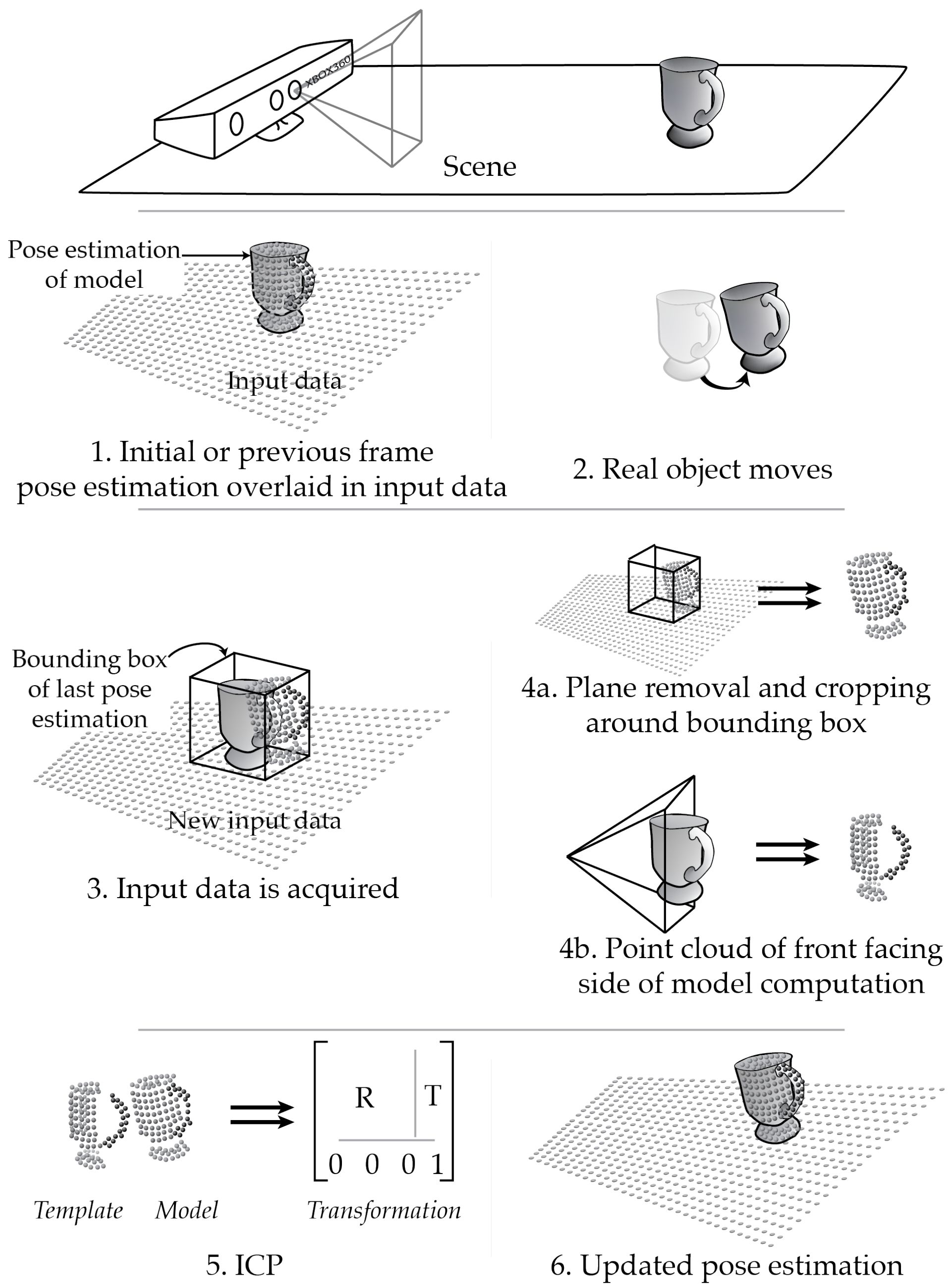

XBOX360
Scene
Pose estimation of model
Input data
1. Initial or previous frame pose estimation overlaid in input data
2. Real object moves
Bounding box of last pose estimation
New input data
3. Input data is acquired
4a. Plane removal and cropping around bounding box
4b. Point cloud of front facing side of model computation
R
T
0 0 0 1
Template
Model
Transformation
5. ICP
6. Updated pose estimation

**Figure 5.2:** Preparation of the input data for the ICP. Image adapted from Avellino [2013].

of the bounding box. We now have one of the two point clouds for the ICP: the model.

(b) As we know where the object was in the last frame, we can place our model in the 3D space and take the points that face towards the camera. This point cloud is our expected input: the template. Without movement of the object (and noise in the data), the template would be exactly the same as the model. We only use the front-facing points for the template because the model is also only seen from one side and the ICP will not give good results if we ask him to map the front side of an object to a full object.

5. We now apply the ICP, and it returns a mapping from the template to the model.
6. We can now store this information and update our pose estimation.

Now we know where the object is and how it is oriented. This enables us to look for fingers and see whether they are touching the object.

### 5.2.3 Touch Input & Touch Occlusion

Intersection detection takes long.

The problem of detecting touch for us is the same as detecting an intersection of object model with the finger. Currently, the finger is only known as a point cloud, more specifically, the points facing the camera. If we had a mesh representation of the fingers and would map it on to the point cloud, we could check for intersections. Unfortunately, the finger cannot easily be modeled by a (solid) model, and especially the computations required to check for intersections would take considerable time: We would need to intersect the triangles of the two meshes.

Depth thresholding requires expensive ray-casting.

Another problem is that the finger occludes the object. Although our object recognition gives us the pose of the object, allowing us to restore which points of the model are occluded by the finger, we do not know whether the finger is touching the object or is just in front of it. However, if we assume a static finger thickness, we can check for the touch by comparing depth of the front-side of the finger plus finger thickness to the depth value of the object's front side. To be more specific, we need to check this for every point of the finger with the point lying "behind" it. Calculating the point behind another, however, requires us to cast a ray (parallel to the camera axis) through that point and intersect it with every triangle of the model, which would also require a lot of computation.

Map object to 2D and compare there.

Instead, we simplify the 3D calculations to 2D image comparisons. This works well in our use case and is very fast because it exploits the capabilities of modern graphics cards: Instead of casting rays, we render the object as it would be seen from the perspective of the camera. Theoretically, this is also quite an expensive computation, but modern graphics are highly optimized for this task so that they can deliver this in only a few ms, making it a very feasible approach. We also project the points of the finger into a 2D image with the same viewport as the other image. This means that we can now compare pixel by pixel whether the finger is in front of the object. As we store the depth data of the model and finger as well, we can thus use depth thresholding to do our hit detection.

The resulting touch detection process is explained in Figure 5.3:

1. We start out with the point cloud data and the current pose estimation of the object.

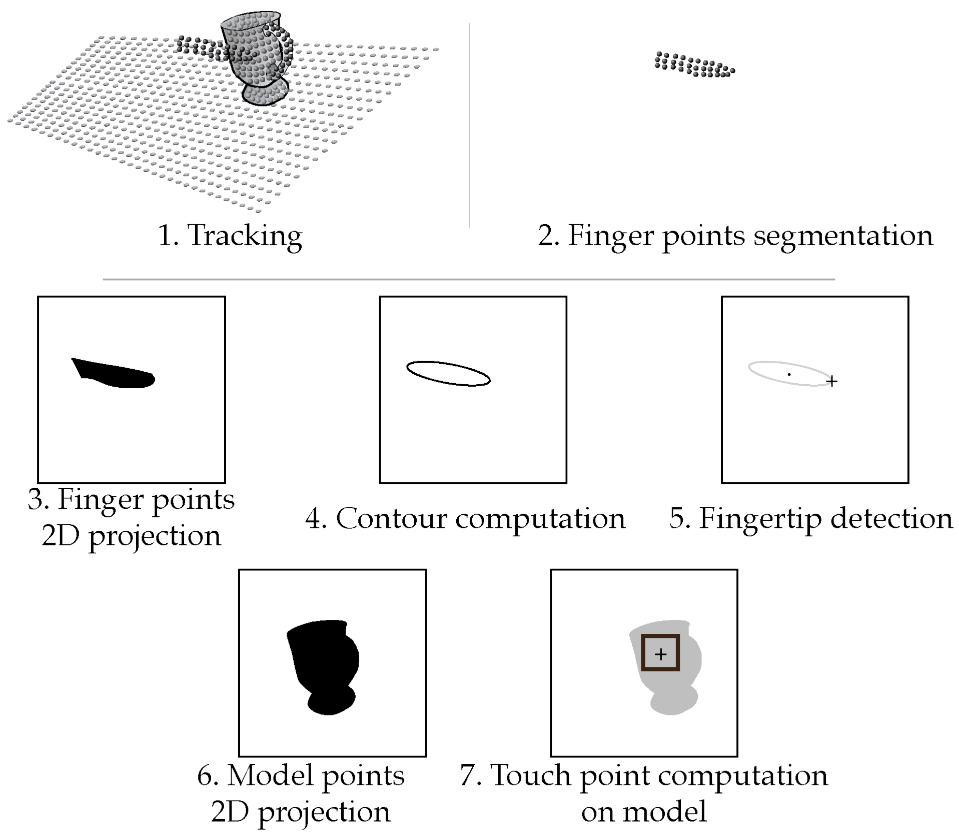

1. Tracking
2. Finger points segmentation
3. Finger points 2D projection
4. Contour computation
5. Fingertip detection
6. Model points 2D projection
7. Touch point computation on model

**Figure 5.3:** Touch detection algorithm. We reduce the 3D problem to a 2D image problem. Image adapted from Avellino [2013].

2. We segment the points of the finger by color filtering.
3. We now treat the data as a 2D image as seen from the camera.
4. We compute the contour by calculating the convex hull of the finger points.
5. We now take the point farthest away from the center of the hull as the fingertip.
6. We render the model and project the fingerprints from the same viewport.
7. We now look up fingertip coordinate in the model image, take the points from the vanity (3 pixel neighborhood), and perform depth thresholding to decide whether there is a touch or a hover.

Color touch zones on the object.

Now we know whether the finger intersects with model, i.e., whether we hit an object, and we can create a touch event and send it to an application. However, we do not yet know where we hit the object. For this, we need to extend the last step to react differently depending on where it hits the object. We color the object differently for each touch zone. The color can then be checked in the last step of Figure 5.3 and will tell us which touch-zone we hit.

### 5.2.4 Evaluation

Focus on accuracy and lag.

Our system has two main parts, the tracking of the object and the detection of touches on it, and we will evaluate them separately. The primary criteria for evaluating any input device or tracking system are its reliability (or accuracy) and how long it takes to react to a given input.

The whole evaluation is done on a MacPro with 2.26 GHz, 6GB ram, and a Nvidia GTX 580 graphics card. As a camera, we used a first generation Microsoft Kinect that returns VGA resolution RGB and depth data. The latter is 11bit and has a base accuracy of 2 mm at 50 cm depth and gets worse farther away. The full details on the evaluation can be found in Avellino [2013], we will just describe the general setup and the main takeaways.

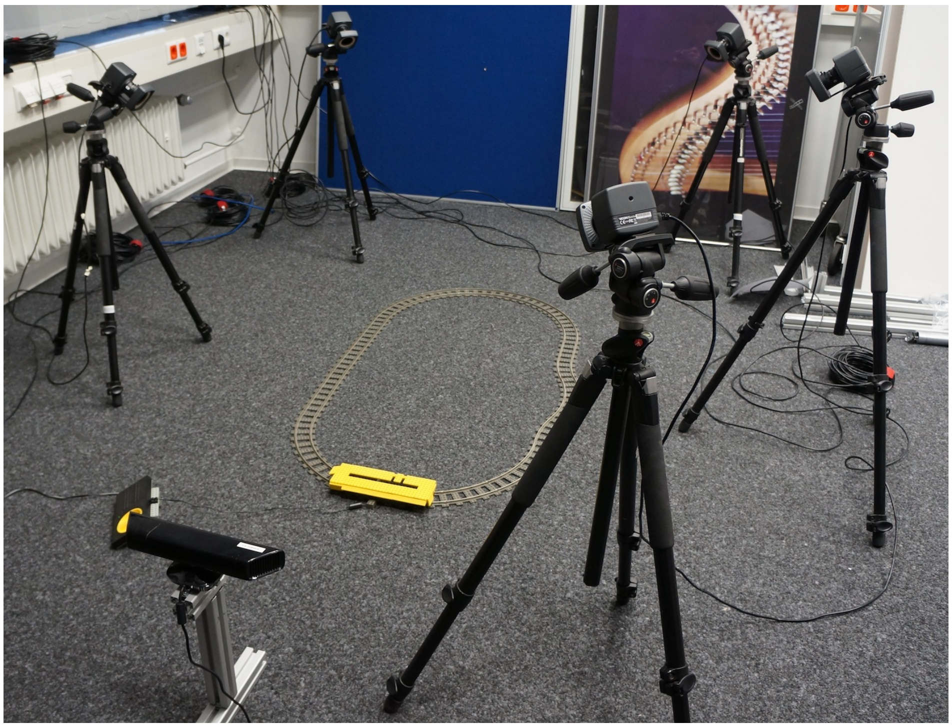


**Figure 5.4:** Our test setup with the toy train tracks and the VICON cameras. Image adapted from Avellino [2013].

#### Object Tracking

Compare with VICON.

In order to evaluate the accuracy of the system, we need to have a baseline. We decided to compare it with a sub millimeter accurate tracking system from VICON. We looked at position and orientation error, i.e., the difference between the values our system reported and what the VICON Bonita reported. For a more controlled environment, we decided to put the objects on a Lego toy train with a circular track layout (c.f. Figure 5.4). We decided to use two objects: one with few features (a cup) and one with a lot (a toy duck). We put them on the toy train in varying starting rotations

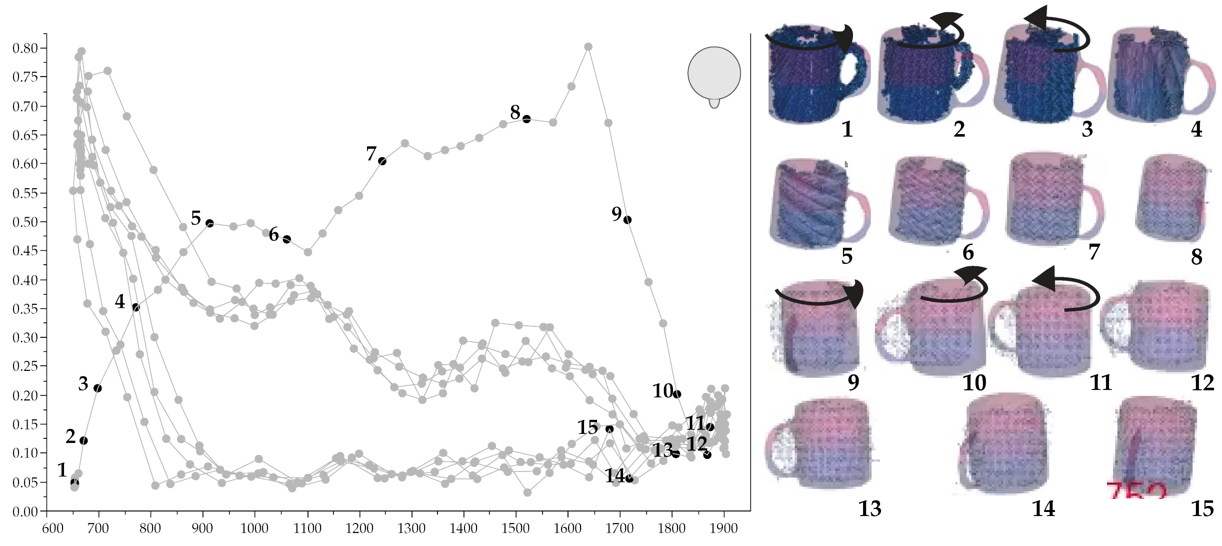


**Figure 5.5:** Rotation error (y-axis) of the cup in norm of quaternion difference. X-axis is the distance from the camera. 1-4 Asymmetric feature disappearing from sight. 5-8 incorrectly estimated rotations. 9-12 Asymmetric feature reappears. 13-15 Correctly estimated rotations. Image adapted from Avellino [2013].

and let them do several circles in both directions. Data from about 8500 timestamps was collected and analyzed.

Even depth error is acceptable.

Position error is reported in Table 5.1 and mm. As we can see, the biggest error is in the depth dimension, partly due to the limited depth resolution of the Kinect. Error in the width and height dimension are very small. Altogether, this makes the system very usable for our use case.

| Position Error in mm: | | Avg, SD |
|---|---|---|
| Cup | width | 16.27,12.51 |
| | depth | 24.04,15,65 |
| | height | 4.32,3.06 |
| Duck | width | 15.22,13.57 |
| | depth | 28.73,17.70 |
| | height | 4.11,3.8 |

**Table 5.1:** Tracking error of the position. Even the depth error is considerably small

Rotation error in norm of quaternion differences.

We measured the rotation error as the norm of the difference of rotation quaternions. The norm is in the range $[\mathbf{0}, \sqrt{(\mathbf{2})}]$, where $\mathbf{0}$ equals identity and $\sqrt{(\mathbf{2})}$ equals an rotation in the exact opposite direction. We chose to use (the norm of ) quaternions because they are a good representation for rotations and especially to compare them.

Accurate tracking requires asymmetric features.

Before taking a closer look at the data, we need to think about what we can expect from the system: For example, if the handle of the cup is not visible, any marker-less tracking system can only report that the cup is in no state where the handle would be visible, but the exact orientation cannot be calculated. We see the same results in our system: As soon as an asymmetric feature is occluded, the error gets bigger. In most cases, fortunately, the system is able to catch up as soon as the feature is visible again, i.e., we see the handle again and the system reports the correct orientation (c.f. Figure 5.5). We will suggest some options in the future work section to reduce such errors. Apart from those general limitations due to self-occlusions, the system estimated the pose well: Selecting only parts where the handle was visible, we see an average of **0.0634** with standard deviation of **0.0201**. This value is similar to a rotation of **7.23°** along a single axis.

Close to real-time performance.

On our test system, it took 100-140 ms to process the incoming frame and update the pose of the object, i.e., we had 7-10 frames per seconds in our test setup. There are several factors that influence this runtime, and they all relate to how many points need to go throughout our processing pipeline: a) the number of points of the model and b) the number of points of the template. We already did a sub-sampling, i.e., only take every 16th point after calculating the front-facing points of the model, but we did not do this for the point cloud from the Kinect in order camera to not lose accuracy. In our tests, the sub-sampled model consisted of about 600 points, and the points extracted from the Kinect image where about 3500. This also means that objects closer to the object camera generate larger point clouds than objects farther away, leading to differences in performance. Future work will take a look at ways to adapt to those factors and get a more flexible trade-off of processing time to accuracy.

### Touch Detection

Users pushed the top of a pen.

After having seen that our system provides reliable tracking, we will now see how good it can detect touches on the tracked objects. Initial testing showed us that we cannot expect a precision similar to our high-precision touch detection on tabletops. This is due to several differences between the system: the surface is non-planar, the object is not static, and the camera has a significantly worse resolution that is also spread over the whole room instead of a very small area. We therefore opted to focus on a task where a person wants to hit one small area on an object.

23% of false-positives and false-negatives.

We asked 12 people to participate in our experiment and let them hold a pen in their hand and asked them to push the upper part of the pen to forward a slide on a screen. We time-threshold the touches to filter out noise by only accepting touches that were on the object for longer than 400 $\boldsymbol{ms}$. We recorded 511 touches as well as video data of the users interacting with the pen. Comparison of video data and captured touch information gave us good results: 76% of the touches performed by the user were correctly recorded by the system. Similarly, 77% of the touches generated by the system were caused by the user (and not due to noise).

### 5.2.5 Limitations & Future Work

Restart object recognition if we lose the object.

We were very pleased with the results of this work, especially its object tracking part, and would like to continue on it. Currently, if the system loses track of an object due to full occlusions or very fast movement, it can only recover if the object is moved back into the region of its last position. Instead, we would then start the full recognition process over again to find it.

Counter occlusion problems with color data or motion prediction.

One problem in tracking the object is that the system sometimes loses track of the orientation of the object because it only sees rotation invariant parts of it, e.g., a cup without a handle. We could try to predict rotation and add constraints based on the previous object position and orientation, e.g., the object will not turn upside down from one frame to the other. Another option would be to use the color of the object in addition to the currently used depth data. There could be features only visible on some sides of the object, helping us to detect its orientation.

Extend to multiple objects.

As of now, the system supports a full database of objects for the recognition, yet the tracking part is implemented for one object at a time. However, apart from performance considerations, there is no real issue with tracking multiple objects. As the current code does not exploit parallelization, tracking of a maximum of objects similar to the number of CPU cores in the systems should be possible with similar performance.

Extend to GPU.

A current trend in a lot of computationally expensive tasks is to move the computations from the CPU to GPU. This works great in theory (e.g. nearest neighbor search can be optimized well for the GPU and is a large part of the ICP), but two reasons made us consider a CPU solution first: Every frame coming from the camera needs to be transferred to the GPU memory, processed there, and the results then transferred back to the system's memory, which still takes a significant amount of time, vastly reducing the benefits of the faster GPU computation. Also, to minimize CPU <-> GPU transfer, you need to perform large parts of the computation on the GPU. This requires reimplementing a lot of existing code in a rather restricted computing environment for optimal performance. Still, now that we know that the algorithm works, we can see which parts are worth reimplementing.

Support deformable models.

The system is based on a database of models with a static point cloud or mesh. It would be interesting to see how we could support deformable shapes. The question how to define and store the "deformability" of an object is a challenge by itself. It could lead to insight on how to recognize those shapes as the standard features used to recognize objects that are based on a fixed shape: We, e.g., use point-pair relations such as distance and curvature in the neighborhood.

## 5.3 Chapter Summary

In order to capture the expressiveness of touch on arbitrary objects, we need to first detect the touches. In this chapter, we presented a novel approach to track objects and detect touches on them. We were able to build a system that can track arbitrary objects in close to real time using low-cost hardware. We also showed an approach that is able to detect touches on those objects.

Having detected a touch, we will now take the next step and see which properties of a touch should be taken into account when interacting with arbitrary objects.

# Chapter 6

# Improving Touch on Arbitrary Objects

In the last chapter, we presented a system that can track arbitrary objects and detect touches on them. Following our "detect, improve, extend" theme, we now want to get a better understanding of how people touch and how we can use this information to create better user interfaces.

Add the hand posture as contextual information.

As a first step for a better understanding, we need to know how the user is touching or holding on to the object. For example, consider a smartphone was our object of interest. Currently, it only supports touches on its front side. We could enhance the touch detection to the whole surface by using our remote sensing approach or by adding capacitive sensing to its whole outer sides (given industrial production capabilities). But what would that enable?

Use touch info differently depending on the different region of the device.

For one, people could perform their touch input on the backside of the smartphone (c.f. Figure 6.2), removing the typical occlusion problems on small displays. Another option would be to use the sides of the smartphone as a scroll bar (Figure 6.1). As we can see, these added touch surfaces vastly increase the input capabilities of the device and support interesting interactions. However, the more of the object's surface reacts to touches, the less room we have to actually hold the device without triggering false touches. We could exploit device specific sensors such as gyroscopes to differentiate between different ways to hold the device and then only take into account specific parts of the object's surface. But this does not seem like a generalizable approach.

An algorithm to infer the hand posture from the touch data.

What would be good instead, is to know which finger is generating which touch, or even better, the full hand posture that is generating the current finger (or palm) prints on the device. This is exactly what we will be presenting in this chapter: an algorithm to infer the hand posture based on the finger prints taken from arbitrary objects. This will allow us to do an informed selection on which touch to respond to. We will also report the result of an evaluation of the algorithm on a set of five different objects

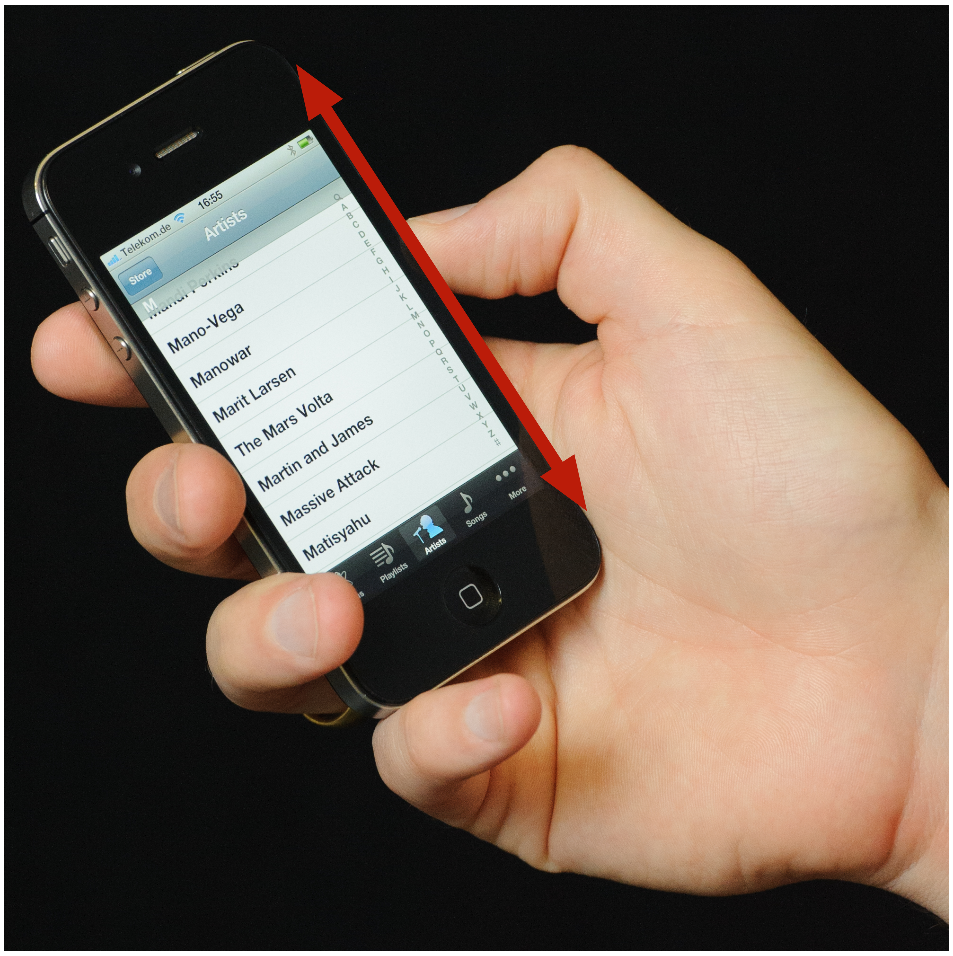


**Figure 6.1:** A person touching the sides of the smartphone with his thumb. This input could be used to navigate the scroll bar.

and several users.

Algorithm needs to work on 2D image input.

Before trying to solve it, we first need to define the problem: A person holds an object, e.g., a cube (c.f. Figure 6.3), and we want to infer the hand posture from the finger prints. The finger prints will look something like Figure 6.4, a set of surfaces with 2D information on them, independent of the way we acquire them: For this example, we used water color. If we use depth-camera-based solutions, we will not have colors but at least a binary map. Capacitive sensing would also provide per pixel information on the touch state.

How to automate hand posture calculation?

Taking a look at the finger prints, you, as a human, should be able to have a general idea of how the user held the cube and maybe come up with a solution like Figure 6.3, which is how the user actually held the cube. But how did you end up at that solution? What kind of heuristics did you apply? And how can we do this automatized on a computer?

Before showing our approach, we first take a look at how others have tried to tackle this and similar problems.

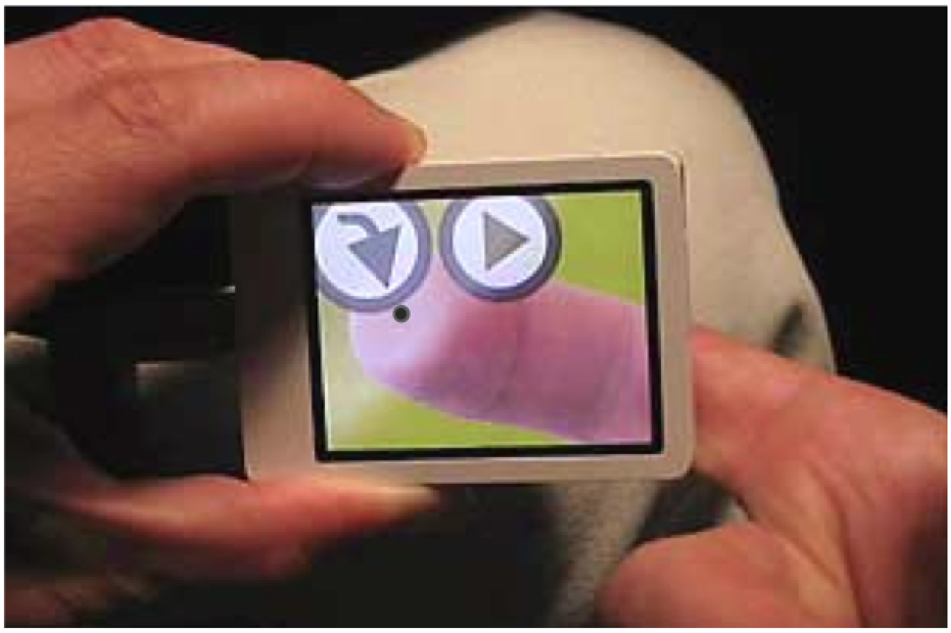


**Figure 6.2:** Using the back of the device to create touch input. Solves problems with occlusion from the fingers. Taken from Baudisch and Chu [2009].

## 6.1 Related Work

Very large solution space.

The problem of inferring the hand posture has been tackled using different approaches. The difficulty is that the human hand is an articulated object with about 27 degrees of freedom (DOF) (c.f. Lee and Kunii [1995]). This is due to the high number of joints that can be rotated each in up to 3 dimensions as well as the 6 DOF for position and orientation. To make this more clear: If we only take a look at 10 different rotation levels per joint, we have a space of 1,000,000,000,000,000,000,000,000,000 possible solutions, and the goal is to find the single correct one. Since not all joints can be rotated independently, however, a variety of models exists that reduces the DOF below 20, especially if you only take into account a subset of all postures. For example Cobos et al. [2008] reduced it to circular grasping and only needed 6 DOFs to reproduce more than 80% of the possible hand postures.

Mature algorithms could be used if we had input from several cameras.

Finding the right model is just one task, one also needs to process the input to find the right assignments of the model's variables. There are a lot of mature vision-based approaches which infer the hand pose from camera images, see Erol et al. [2007] for a recent overview. Although those approaches nowadays provide reliable recognition and support a large number of postures, they all have in common that they need input from a camera with full view on the hand and its grasp, whereas in our case we only have information about hand contacts from inside of the grasp.

Simplify it by using external sensors.

Another option, more similar to our goal, is to use different sensors on the object itself as, e.g., GripSense by Goel et al. [2012] did. They use the

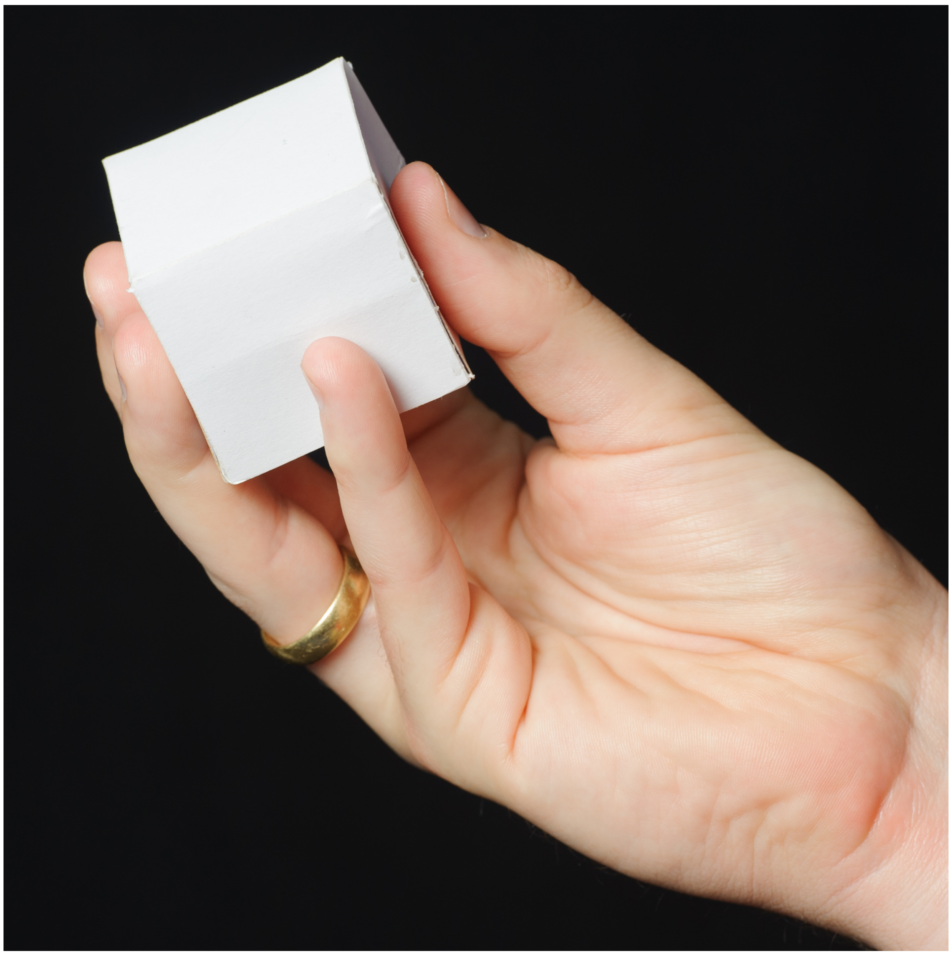


**Figure 6.3:** A person holding a cube.

gyroscope, touch, and accelerometer data and were able to differentiate between a limited amount of discrete holding postures, e.g., one-hand vs two-hand.

Lastly, we want to mention the work from Vatavu and Zaiţi [2013] who solved the inverse problem: They inferred object properties such as shape and size from the hand grasp. This could be useful for sanity checking solutions for the original problem.

Conclusively, we did not find any work that was able to infer hand posture from finger prints on arbitrary objects and will now present our approach.

## 6.2 Inferring the Hand Posture from Finger Prints

Although the basic idea behind our algorithm is rather simple, the final algorithm is not. Thus, we will first give a simple example in 2D to explain the general concept. We will then show how to apply this to our 3D objects

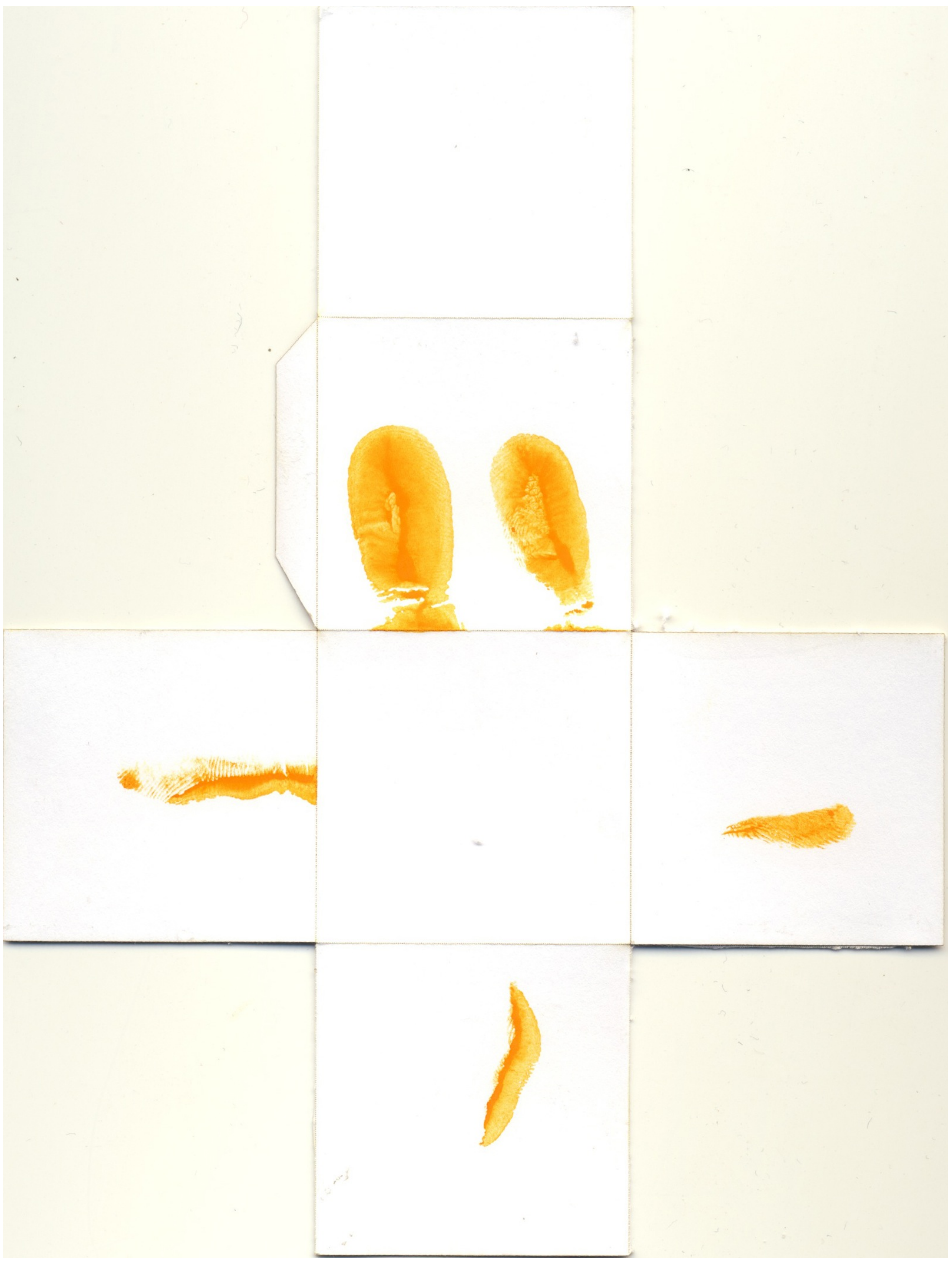


**Figure 6.4:** Fingerprints on paper cube (unfolded). In this example, the fingerprints can be identified easily, and we can infer the rough hand posture.

and give a step by step description of the algorithm on a higher level. Lastly, details of each step will be explained.

Each finger can only reach a distinct volume.

Our general idea is based on the fact that the reach of each finger describes a distinct volume. For example, take a look at the big turquoise point cloud in Figure 6.5. These are all the points that the tip of the thumb can reach in 3D space. These points have been created by our hand model, which we will explain in Subsection 6.2.1. The differently colored clusters of points resemble the reach of the other fingers (blue index, middle bright green, etc.) As we can see, they only overlap partly.

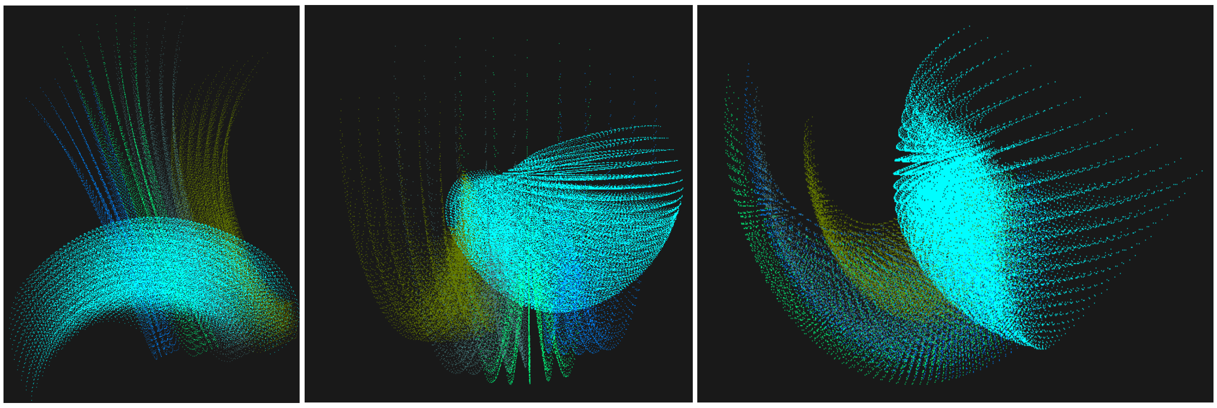


**Figure 6.5:** A visualization of the reach of the fingertips. **Left:** View from above. **Middle:** View from front side. **Right:** View from the left side.

Map fingerprints to fingers with a maximum-likelihood.

Now consider input from our touch sensors that looks similar to the left side of Figure 6.6 after preprocessing and compare it to the right point cloud. Taking a closer at the bottom left part with the “1” in the one image, we see in the other image only turquoise points from the thumb. Thus it seems reasonable that the touch on “1” came from the thumb because only those points are in the vicinity. Similarly, “2” came from the index finger, “3” came from the middle finger, etc. This means we found a solution for our input (which in this case is also correct).

How to find the coordinate transformation?

But we only got to this assignment of touch input to finger labels because the touch data and our point cloud were nicely aligned. So the challenge is to find this “alignment” , i.e., a mapping of the 2D image points into the 3D point cloud of our model and then look in the local neighborhood for likely points.

Guesstimate the transformation to hand coordinates.

Our approach finds this alignment by “guesstimating” the position and orientation of the object w.r.t. to hand origin, i.e., the hand wrist, and then return a hypothesis, i.e., finger to touch point assignment. We iterate this process to generate a lot of hypotheses and use several criteria to filter out the wrong hypotheses and come up with the right solution. This method works really well as long as we can find good (object dependent) criteria to filter.

Here is the full rundown of our algorithm (c.f. Figure 6.7), we will explain each step in detail afterwards:

1. Get touches from the sensor. Map them to single 3D points.
2. For every subset of five touches:
    (a) Guess several hand origins $\boldsymbol{o}$ close to the center of gravity $\boldsymbol{c}$ of the touches.
    (b) For each origin:
        i. Get $\overrightarrow{\boldsymbol{co}}$ (input) and $\overrightarrow{\boldsymbol{c_m o_m}}$ (model).

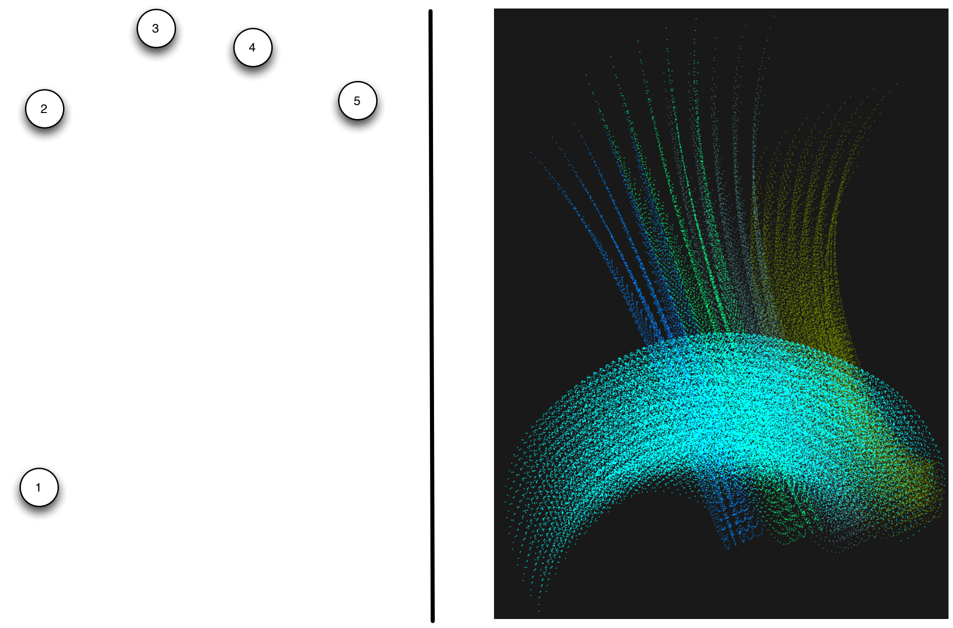


**Figure 6.6: Left:** Processed input from our touch sensors. **Right:** Point cloud of our hand model, where each color represent the volume of another finger. Select finger nearest to a touch.

   ii. Find transformation $\boldsymbol{T}$ from $\overrightarrow{\boldsymbol{co}}$ to $\overrightarrow{\boldsymbol{c_m o_m}}$.
   iii. For each rotation $\boldsymbol{\alpha} \in [\mathbf{0, 30, ..., 330}]$
      A. Map touch points into the model and rotate around $\overrightarrow{\boldsymbol{c_m o_m}}$ by $\boldsymbol{\alpha}$.
      B. Assign fingers to the touch points.
      C. Evaluate assignment.

3. Return the assignments that fulfill our selection criteria.

### 6.2.1 An Anatomical Hand Model

Before starting the actual algorithm, we need to calculate where our fingers can reach and store this for comparison with the input. For this, we created a kinematic model based on the work of Cobos et al. [2008] and took the actual bone measures and joint flexibility from the classic anatomy hand book by Schmidt and Lanz [2004]. The exact numerical values can be found in Appendix A. As can be seen in Figure 6.8, each finger has its own kinematic chain and is not influenced by the movement of the others. In an actual hand, this is slightly different because some of the fingers can be tied to the same set of tendons, limiting their movement by each other. This effect can be approximated by reducing the overall flexibility of each finger.

We create our own kinematic hand model.

There are three reasons we decided for this approximation: (i) For a first model, we do not need to model every hand posture, especially not the

Model each finger independently.

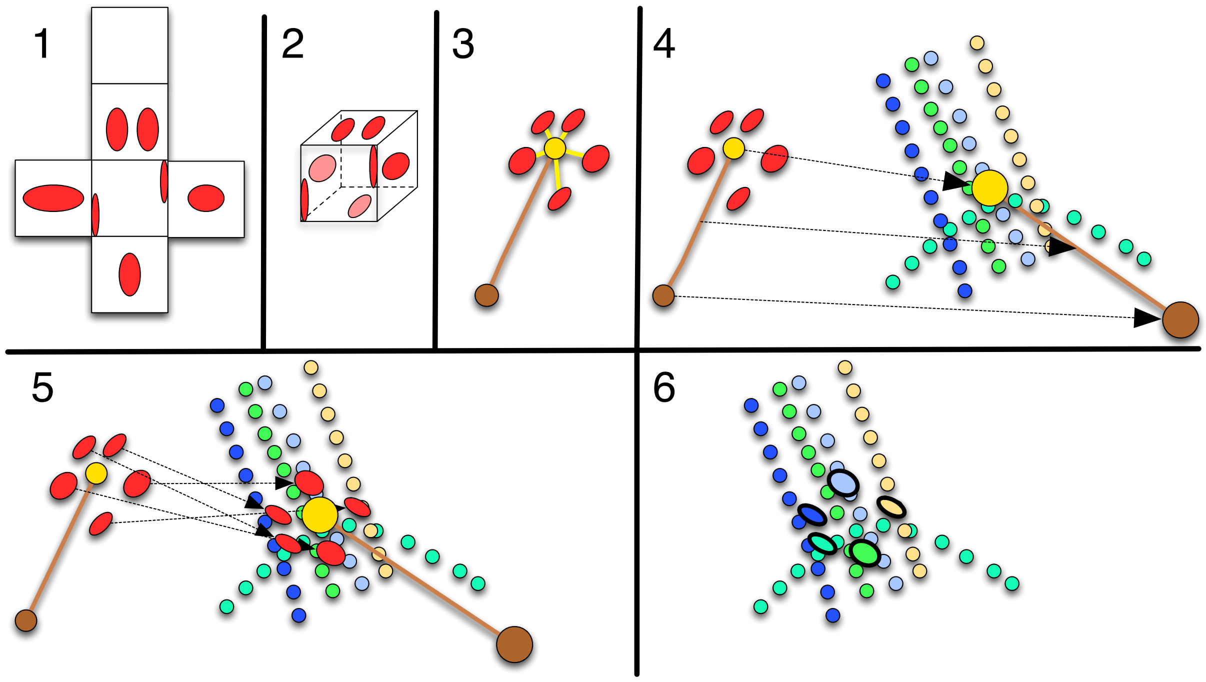


**Figure 6.7:** The algorithm to infer the hand posture. Rotation around the connection line is omitted in this sketch: **1:** Detect ellipses on touch sensor input. **2:** Select five touches and map to 3D coordinates. **3:** Calculate center of gravity, guess hand origin. **4:** Get transformation from input space to model space. **5:** Apply transformation to touch points. **6:** Assign finger labels to touch points.

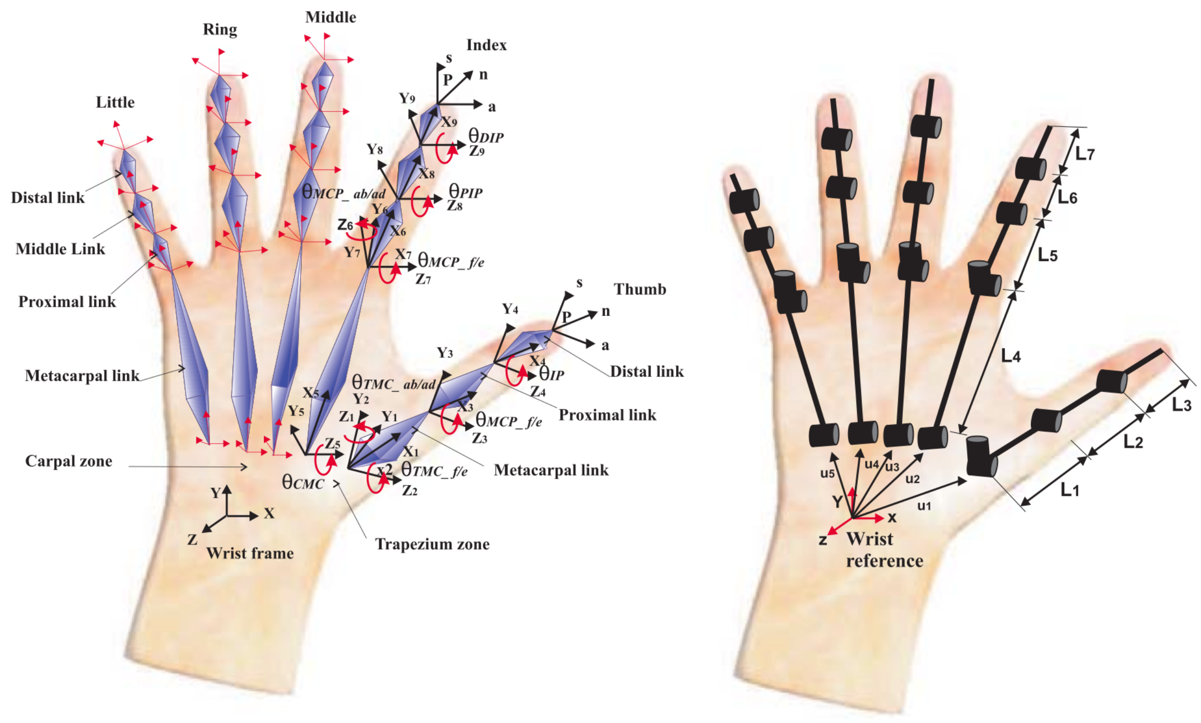


**Figure 6.8:** **Left:** Labels for the hand model and in which direction the joints can rotate. **Right:** Distances w.r.t. the wrist reference. Taken from Cobos et al. [2008].

extreme hand positions because these are typically uncomfortable and thus unlikely to be applied at all. (ii) Although well studied over decades, the interactions between the fingers are still hard to model and additionally depend strongly on each user because there is a great variation in which tendons exist in the first place for one person and also how they are tied together. (iii) Lastly, modeling of each finger separately greatly reduces the solution space because we do not need to consider cross-combinations. In other words, instead of 24 degrees of freedom for the whole hand, we have four separate finger models with 5 DOF each (abduction and adduction only on the "knuckle" joint) and one model for the thumb with 4 DOF.

Discretize rotation angles in 5°bins.

If we now sample each DOF, i.e., the rotation angle of one joint, into 10 discrete steps, we can capture the reach of the whole hand with $\mathbf{4 \times 10^5 + 10^4 = 410,000}$ points—a manageable amount. Additionally, we can exploit the fact that for active flexion, the joint close to the finger tip is always rotated two thirds to the joint next to it (Fahn and Sun [2005]), thereby removing another DOF. We also sample not into 10 discrete steps, but sample every **5°**, resulting in varying number of samples per DOF. These modifications result in a hand model consisting of **97323** points, each point representing one possible position of the finger tip as dictated by the kinematic model. At each point, we store (i) which finger belongs to it, (ii) the transformed normal of the finger tip, i.e., which direction "down" would be, and (iii) how the finger was held to create this position. This data is stored in a kd-tree (Bentley [1979]) for fast neighborhood search. Three views of this model were already shown in Figure 6.5.

### 6.2.2 From Touches to 3D Points

Get touch ellipses from sensor data.

We start out with input from our touch sensors, which are a 2D image for each face of the object (Figure 6.7 Step 1). We use standard computer vision methods to process the input: removing noise of the image using morphological operators, connected component analysis to identify different blobs on each face, and ellipse fitting on those blobs to estimate the location of the touch. This gives us the center, main-axis, and off-axis of the ellipse. These are all in 2D image coordinates.

Map to 3D coordinates and merge neighboring touches.

Since we know the shape of the object, we can transform this 2D data into the 3D object coordinate system (Figure 6.7 Step 2). Additionally, we also store the normals of each face for later use. Ellipses that are very close to each other, e.g., two that are on the same corner of a cube but on different faces, are considered the same touch and merged together. In this phase, we also filter out ellipses that are too big to come from a single finger and are thus more likely to come from the palm of the hand.

### 6.2.3 Touch Subsets, Hand Origin, and Transformations

Take any subset of five from all the touches.

The previous step can give us more than five touches because, for example,

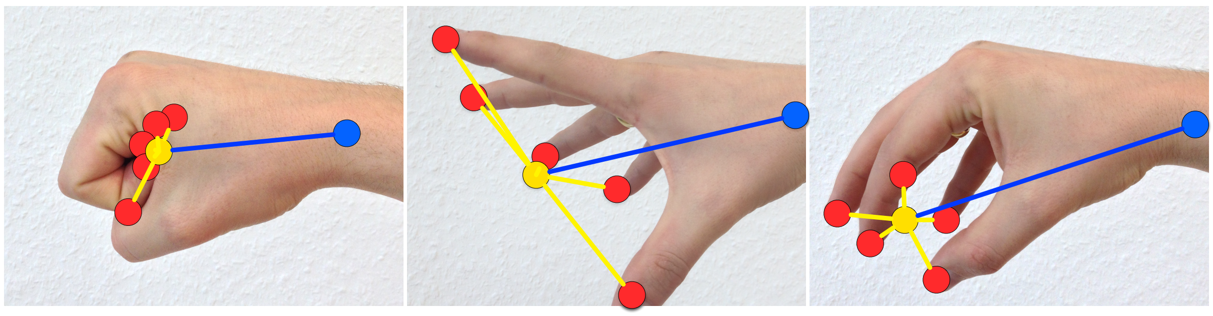


**Figure 6.9:** Calculating the center of gravity of the finger tips. Although the poses are different, the center is in a similar spot. Especially, the line between it and the hand origin is very similar.

not only the finger tip but also the rest of the finger can create touch ellipses. However, our algorithm can only map five touches to five finger tips, so we need to reduce this number. For now, we did not find good filtering heuristics that work for a variety objects, which is why we just take any subset of five touches from the set of all the touches. Although this can greatly increase the number of iterations, we will show in the evaluation section that the inner loop is efficient enough to be performed very often.

Center of gravity is quite stable.

As mentioned in the initial sketch of our algorithm, the major problem is that we do not know where the hand is relative to the object, i.e., we do not know how to map our input into the point cloud of the hand model. Instead of just guessing position and orientation of the hand (6 DOF) and then iterating until we find suitable solutions, we exploit the fact that the relation between the finger tips and the wrist (which we will refer to as hand origin from now on) is constrained to a degree. Take a look at Figure 6.9: Although the finger tips can be at quite different positions, their center of gravity is quite stable. Especially the line between it and the origin does not really change even for drastically different hand postures. We will refer to this lines as the hand orientation line because it constrains the overall hand orientation to one degree of freedom: the rotation around the axis of this line.

Additionally, the distance between the center of gravity and the origin is also constrained. In an informal study we only saw distances between 4 and 15 cm.

Map input space to model space and try different rotations along the hand orientation.

Taking this altogether, we calculate the center of gravity of our input, guess a hand origin around it in a constrained distance, and draw the hand orientation line between the two (Figure 6.7 Step 3). We then do the same for our model data (actually, we do this only once when creating the model) and map the input into our model (Figure 6.7 Step 4,5). Lastly, we rotate the points around the axis of the hand orientation line. We use a discrete sampling of all possible rotation angles to remove this degree of freedom.

### 6.2.4 Find Assignment and Evaluate

How to assign finger labels to the touches?

We finally have our input data mapped to our model and can now look which finger caused this touch. As mentioned in the first example, we are mapping the five input touches to the five fingers of our hand model by taking a look at the points in the vicinity (Figure 6.7 Step 6). These points from the model are labeled with which finger they belong to. However, contrary to the first example, the mapping between input and model is not always straight forward because fingertips from the input can have differently labeled points in their vicinity. So, which tip gets to be labeled as the thumb, which tip is the index finger, etc.? It would make sense for each finger tip to pick the "label" that occurs the most in its vicinity. However, labeling the first tip as, say "thumb", removes this choice from the other finger tips, and it could be that one of the other finger tip only had "thumb" labels in the vicinity.

Instead of trying different heuristics, let us take a more formal look at the problem. We have a set of five points (from the input), and we need to map them to five labels (the fingers of the model). This mapping is bijective because we need exactly one label for each point, and vice versa. This is exactly the definition of a (perfect) matching (West [2008]) and thus easy to compute. However, we are not only looking for any matching, but the one that makes most sense. As mentioned above, we prefer assignments where each finger gets the label that occurred the most.

Assignment problem can be reduced to a matching.

Fortunately, this can be modeled as follows: For every finger tip, we have a node on one side of our bipartite graph. For every label, we have a node on the other side of the graph. We now connect every label with every finger and assign weights to these edges as follows: An edge connecting node **left** with node **right** has the weight equal to the number of points labeled with **right** in the vicinity of finger tip **left**. Solving this graph for a Maximum-Weighted-Perfect-Matching returns the best possible assignment. We store this assignment in our solution set.

Two metrics to compare our solutions.

Additionally, we calculate two values to see how good our solutions is. A histogram fit and an angle fit. The first measures how unambiguous the matching was and is calculated as the product of the histogram fit of each finger tip, which in turn is the ratio between the number of neighboring points from the label we selected to the number of all neighboring points. A value of **1** would mean that the vicinity of each finger tip only consisted of points from one finger, i.e., labeled with the same number.

The second criteria, the angle fit, measures how aligned the touch was with the surface. If you touch a surface with your finger tip with zero pitch or roll of the finger, we consider this as a **1**. If you touch the surface with your finger turned **180°**, i.e., with the nail lying down, this is a **0**. To get to this value, we compare two direction vectors, the one of the surface (looking out of the surface) and the one of the finger pointing down. For the surface, we take the value which we stored in the beginning of the image recognition and transformed similar to the finger tip itself along the whole pipeline. For the finger tip direction, we use the tip direction that is in our model: Let's say we have touch "1", and the matching told us that this finger tip came from the "thumb". We take the average direction of all the points from the thumb that were in the vicinity of this tip and use this to compare against the surface normal. Similar to the histogram fit, we calculate the product of all the finger angle fits.

### 6.2.5 Final Filtering

Filter solutions by angle fit, histogram fit, and occurrence.

All of the above steps are iterated as described in Listing 6.2 and generate a large amount of hypotheses. We filter them based on the two criteria by a simple thresholding. Additionally, as similar touches-to-finger assignments can occur, we can also consider only solutions that occur very often, thus filtering in a maximum likelihood approach.

## 6.3 Implementation

Implemented in C,C++, and Objective-C.

Before talking about how well the whole algorithm works on real data in Section 6.4, we want to point at some implementation details. The algorithm itself was done in C/C++ and was less than 3000 lines of code, whereas for visualizing the point cloud, the GUI and importing, exporting data we used existing code in Objective-C. The connection between both parts was realized with a C bridge. Overall, the implementation of the algorithm is straightforward because each step consists of control loops, generating point sets, and basic linear algebra, but some parts warrant detailed explanations.

We recommend fast kd-tree and matrix multiplication libraries.

As mentioned before, the points of the hand model were stored in a kd-tree. As a significant amount of time is spent in performing nearest-neighbor search, we would definitely recommend to use that or a similar data structure that supports this kind of access. We used the PCL library for this. Another major part of the algorithm consists of vector and point transformations, which are done via matrix multiplications. We decided to use the highly optimized uBLAS Library for these operations. In one of the later steps, we want to find a weighted matching between input points and the hand model based on the cardinality of the neighborhoods. There are a lot of different libraries, but we used the lemon library that allowed the calculation including data type conversion to be done in less than 20 lines of code.

Some parameters need to be adapted to the input.

Throughout the algorithm, we used several constants and will now explain how those were set. The first part that turns the scans of the object surfaces into 3D points used several filters and thresholds. We first used a gaussian smoothing filter with a kernel size of 15, then dilated 1 pixel. We then filtered for orange color between $\mathbf{(80, 50, 100)}$–$\mathbf{(120, 255, 255)}$ in HSV space. After ellipse fitting, we only considered ellipses with more than 200 pixels. In the last step of creating the touches, we merged neighboring ellipses. We considered ellipses that are no farther apart than 90% of the sum of their average axis to be neighbors. All of these above values would need to be adapted for the specific sensor "hardware" in other projects.

As part of the actual algorithm, we generate a fixed amount of hand origins. Throughout our test we used 1000 origins. Here is a trade-off between accuracy and run-time speed. Similarly, we are looking in the 1 cm neighborhood of a point when comparing input and hand model. The last trade-off parameter is how dense we create the initial hand model. We opted to calculate one finger tip for every $\mathbf{5}$ degree of joint flexion or abduction.

## 6.4 Evaluation

Use water color and cardboard as touch sensors.

As our approach should be applicable to arbitrary objects, we needed to evaluate it on different objects. Although we introduced a tracking and touch detection algorithm for those objects in the previous chapter, we will pick another solution because our algorithm has one drawback here: It currently supports only one camera and thus cannot capture touch contacts on all sides of the object. Instead, we will pick a much simpler solution that makes data acquisition more tedious but allows for great accuracy: We paint the user's finger with water color, let them touch the one-way object made out of cardboard, cut and unfold the object, and scan its surface for further processing (see Figure 6.10).

Users perform one task with each object.

We selected five different objects: a cube, a laser pointer, a tissue box, a can, and an iPhone. For those objects, we created cardboard versions with the same shape and size (Figure 6.11). Throughout the study, users were instructed to solve a simple task with the object, e.g., carry the box from spot A to spot B, while using all five fingers. However, as not only the shape and size afford a specific posture, they always started with a real object and solved the task with their left hand. Then they had to repeat this very gesture with their water-colored right hand and the cardboard version for capturing the touch.

Manually labelled the scanned finger prints.

We collected ethnographic data of the users (gender, age, length and width of the hand, physical impairments) and videotaped the interaction of the users with the objects. After the study, we cut the objects open at predefined edges and unfolded the objects. We scanned the surfaces and manually labeled all fingerprints to which finger (or the palm) and which finger segment they belong. This gives us ground truth for comparison with the output of our algorithm.

Study was performed with 11 computer science people.

For the user study, we recruited eleven users, all right-handed, aged 18–36 ($\boldsymbol{M = 26.7}$, $\boldsymbol{SD = 5.9}$), with 8 males and 3 females, all with a computer science background. Their hands were similarly sized (Width: $\boldsymbol{M = 10.4}$, $\boldsymbol{SD = 1}$; Height: $\boldsymbol{M = 18.2}$, $\boldsymbol{SD = 1.5}$), and the participants had no

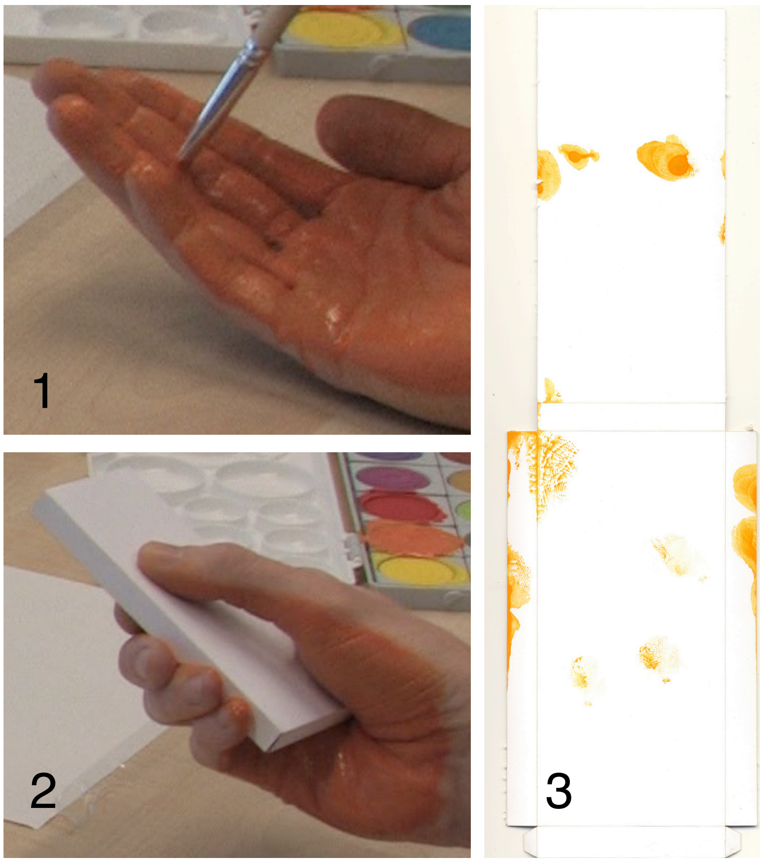


**Figure 6.10:** The process to acquire touch data for the evaluation. (1) Paint the hand, (2) Let them touch the object, (3) Unfold and cut the cardboard, then scan.

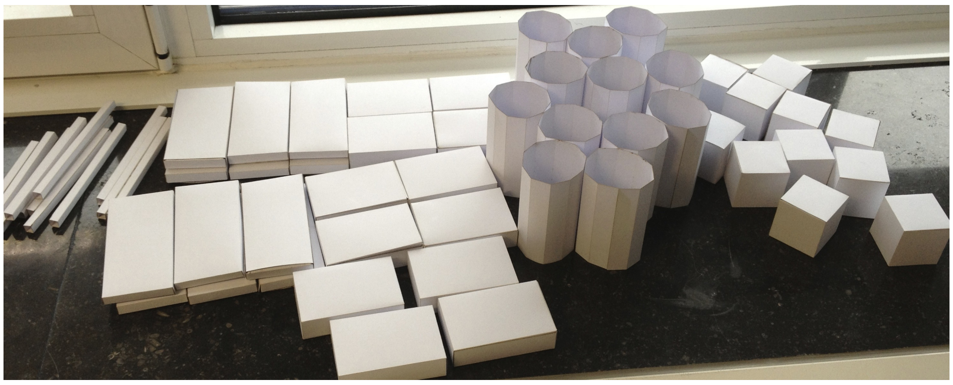


**Figure 6.11:** Five different object types used in our study: pen, smartphone, tissue box, can, cube.

physical impairments.

### Results

Removed some cases due to missing fingerprints.

The study gave us **11** × **5** = **55** different test cases, of which ten cases needed to be removed from further analysis because the posture could not

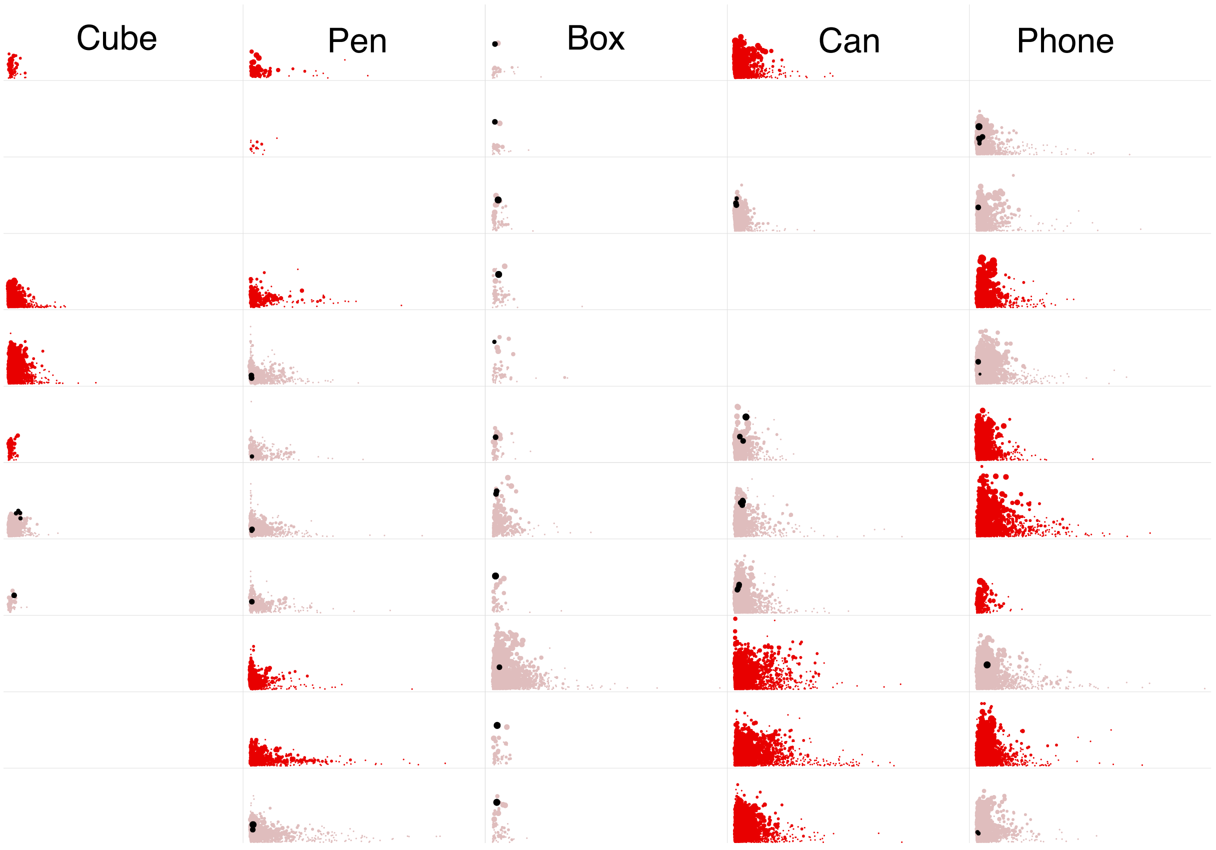


**Figure 6.12:** Results of the posture recognition algorithm. The columns are the different objects, the rows are the 11 different users. Red (light or saturated) represents incorrect solutions, black represents correct solutions. Inside the rectangles, the X-axis is the histogram fit, y-axis is the angle fit. Circle size represents number of occurrences of this solution. In most cases, the algorithm finds at least one correct solution (among the wrong ones). We can also see that the objects perform differently.

be reconstructed manually, e.g., due to missing fingerprints on the object. This leaves us with 45 cases to evaluate.

Performing the algorithm on the data from the user study with the aforementioned parameters results in about 7 million suggested solutions. Merging duplicate solutions, we have **60, 833** suggested unique solutions, which can be seen in Figure 6.12. The reason for those many suggestions is the high number of finger prints that some objects had. The can, especially, sometimes had 20 distinct contacts with the palm and different finger segments.

Taking any subset of touches only increases the amount of solutions by a factor of 5.

The merging reduces this number, but we still need to test any five subset of all those touches. However, this is way less problematic than we feared because selecting the wrong subset does rarely lead to a solution: Final data shows that of all the 60 thousand solution sets, 12 thousand are based on a set of fingerprints that belong to different fingers (fingertip or the finger itself) and only four times more are using parts of the palm or one finger multiple times.

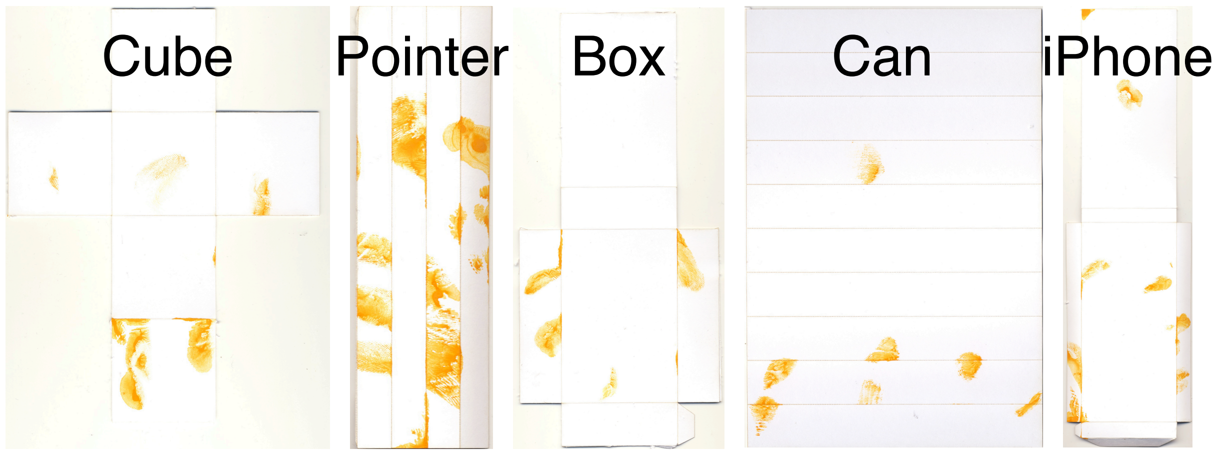


**Figure 6.13:** Scans of the objects from one user. The algorithm found correct solutions for all objects but the phone.

Better ellipse detection should lead to correct solutions for the failed cases.

In 27 of the 45 case (60%), the algorithm suggested a correct solution among other wrong solutions. After taking a look at the individual scans of the objects (c.f. Figure 6.13), we think that a more sophisticated ellipse detection would have allowed us to generate correct solutions for a lot more test cases.

Criteria thresholding often returns good results.

For the cases where the algorithm did give a correct solution, the question is how good can it differentiate the correct from realistic, yet wrong solutions. We suggest a filtering based on the three thresholding criteria as mentioned in the algorithm description: the histogram fit, the angle fit, and how often the single solution occurred. Although there are about a thousand times more wrong than correct solutions, the filtering can remove 87% of those wrong ones while removing less than 10% of the correct solutions. We can tailor a filter to one specific object. This works best for the box because 72% of the filtered solutions are correct, the other objects go as low as 5% accuracy. Our interpretation is that it works so good for this object because a) it has very few fingerprints (and especially only few palm contacts), so we do not base our calculations on the wrong selection of fingerprints and b) it has a rather natural holding gesture, making our histogram fit and angle fit work very well.

With a real-time constraint, detection accuracy is only halved.

As we envision our algorithm to become part of a standard touch detection pipeline, not only accuracy but also run-time is of interest. For objects such as the can that can have a large number of touch contacts, the algorithm takes hours to complete, making it not usable. However, if we have an object wit typically few touches, e.g., the box with less than ten touches, the algorithm is really fast: It takes only 75 seconds to analyze the touches of the 11 users. If we now reduce the number of guessed origins from 1000 to 50, it only takes 3.5 seconds for 11 users, i.e., 310ms for the touch detection of a single set of finger prints. However, this change halves the detection accuracy to 35%.

Having seen that the algorithm is able to produce valid solutions for a variety of objects, we will now take a look at how we can improve it further to give reliable detection for any kind of object given real-time settings.

## 6.5 Limitations & Future Work

We will sort our future steps by following along the current processing pipeline of the algorithm.

When generating the model, we used a sampling stride of $5°$ along each joint. This was based on initial testing, but other values could give a better trade-off of detection vs. run-time. The current model itself only takes into account the reach of each finger, but does not check whether a bone of one finger would cross a bone of another finger to achieve a specific gesture. We could add this as part of a physics simulation that would also consider the object as solid and rigid. This would mean we could check whether the fingers were able to hold this posture without "intersecting" the model.

Physics-based model could detect collisions.

During the initial recognition of ellipses, we could use more sophisticated methods, such as 2D template matching, to ensure that we are not considering prints of the palm as prints of the finger. This would reduce the number of touches to be tested, giving us a lot less invalid solutions and especially reduce the run-time significantly: We take any subset of five touches from all touches, which means that we have $\binom{n}{5}$ iterations, a term that grows (or shrinks) exponentially with the number of touches $n$.

Better ellipse detection could reduce run-time drastically and increase accuracy.

In the evaluation, we guessed 1000 origins for each set of touches and searched in the $1$ cm neighborhood of the model. This worked well but were set arbitrarily. A smarter solution that maybe adapts to the model could lead to better results. Lastly, we could explore different filtering criteria in addition to our angle and histogram fit. They worked well but were not sharp enough to always distinguish correct solutions from bad ones.

Explore other filtering criteria

## 6.6 Chapter Summary

In this chapter, we showed that interaction with arbitrary objects requires a generalized touch detection pipeline. We suggested an approach to infer the hand posture from the finger prints on the surface of the object that was based on an anatomical hand model describing the reach of each finger. Our evaluation showed that the algorithm always returns the correct finger assignments, yet it depends on the object whether the algorithm can distinguish between correct and invalid assignments.

We now can rely on the framework from Chapter 5 and know that we could infer the hand posture from the touches. This contextual information helps us to improve the touch: Depending on the hand pose, we only react to specific fingers and ignore the others. In the final chapter, we can now add another piece of information: the object itself. This creates a new interaction metaphor—Instant UIs—and truly extends the touch far beyond its current usage.

# Chapter 7

# Extending Touch on Arbitrary Objects

Following our overall theme of "detect, improve, extend", we managed to detect touches on arbitrary objects in Chapter 5 and got a better understanding in Chapter 6 on how different people touch objects and how those hand postures could be inferred. We will now apply this knowledge to enhance interactions with arbitrary objects by taking into account the object themselves and not only considering them as a multi-faceted interactive surface. Part of this work has been done in collaboration with the master student Christian Corsten and presented at ITS 2013 (Corsten et al. [2013]).

Take into account the object itself.

Although our framework is capable of detecting touches on arbitrary objects, we wanted to focus the set of objects a little bit to have a more manageable variety of sizes, shapes, and object materials. Thus, we take a closer look at objects that exist in our vicinity: everyday objects such as pen, paper, staplers, and rulers. This has the added benefit that we can rely on the design knowledge that was used to create those objects for similar or different tasks.

First, we will take a look at this design knowledge and how it shapes objects around us. Then, we will give a more profound definition of Instant UIs, present a diary study that shows which objects actually are in our vicinity, present results from two initial studies that showed us how people envisioned to use such a system, and finally present our final study based on our framework in Chapter 5 that shows how well people could repurpose everyday objects as input devices.

## 7.1 History, Current State, and Future of Interaction Design

Norman wrote in 1988 that *"..the term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used. [...] Affordances provide strong clues to the operations of things. Plates are for pushing. Knobs are for turning. Slots are for inserting things into. Balls are for throwing or bouncing. When affordances are taken advantage of, the user knows what to do just by looking: no picture, label, or instruction needed."* Norman [1988].

Affordances have been exploited for product design.

These affordances have been exploited for product design for decades. In HCI, we closely try to resemble the look and feel of real world objects to ease the learning of a UI: Buttons are plate-shaped, and when clicked, they are pushed in, rotary controls look and work like knobs, etc. Although this has been proven to ease the learning of a UI, the question arises why the user needs to deal with "bad copies" of real world objects instead of just using the original. We think that from an interaction point of view, there is no reason.

A design contract was established over time.

Taking a closer look a the hot topics of HCI research over the last 50 years, we can see: command line input, GUI (using mouse keyboard), GUI using touch, and gestural input. All have in common that HCI constrains itself to one single I/O metaphor or even device, making it easy for the designer to expect input and prepare the application logic accordingly. The user, however, had a rough start, the command line expected him to know the commands, i.e., the input, beforehand and also what they did. This improves with GUIs because they provide visual hints about application state and how you can manipulate it, e.g., press a button. Over time, a design contract was established between user and designer, so that the former knew what will happen, e.g., when he clicks on a button with a disk on it. Then, to support mobile and smaller form-factors, it was necessary to mix input and output space: the touch era began. Here, the application could display content or provide buttons. Additionally, physical manipulation of the content was possible: pan, rotate, and zoom. This time, the input was constrained to touch, but most of the input options were known from the GUI world or relied on input metaphors similar to the physical behavior of a sheet of (stretchable) paper. This ends the history of the single "device" input history (while considering mouse+keyboard as one device).

Gesture interfaces do not afford anything.

Nowadays, (3D) camera-based in-air gesturing removes the device at all. The designer, yet again, defines a "good" set of input gestures, but the user is back in the area of the command line and does not even have a keyboard. He does not know what commands he can enter and not even how he can enter it. Visual feedback can give hints on how the user should perform the input, yet still the mapping between a "Z" gesture and the resulting command might feel arbitrary.

Use everyday objects as input controllers.

Instead, we suggest to take a different route—not a single device, and not no device—pick any device! Even a two year old knows how to interact with a cube, so instead of making it easy for the designer to know the input, make it easy for the user to create the input; let the designer take care of input interpretation for the trivial cases and give easy means for the user to program a device. Our existing framework already supports object tracking and touch detection, and so we only need to think of how to interpret the input.

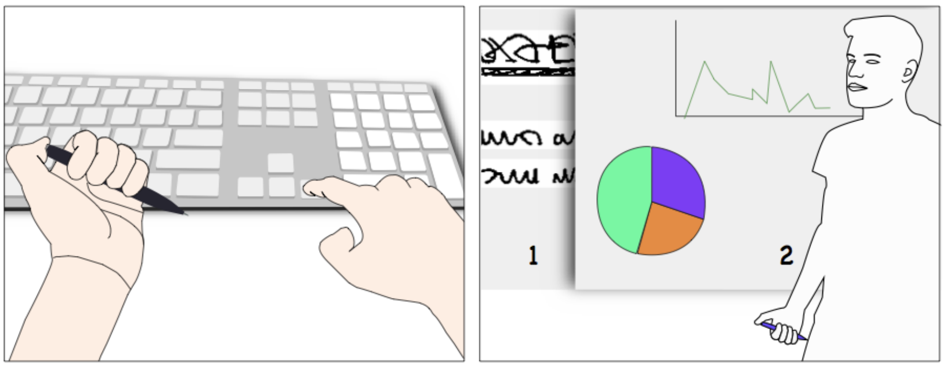


**Figure 7.1:** A presenter forgot to bring the presentation remote. **Left:** She repurposes a pen as clicker and pairs the keyboard shortcut for advancing the slides with the pen by pushing both key and pen button simultaneously. **Right:** The presenter advances the slides by pushing the pen — an almost identical substitute for the presentation remote. Taken from Corsten et al. [2013].

Example: Use pen tip as a remote control.

Let us give an example of how this repurposing of everyday objects as substitute for an existing dedicated controller could look like: Imagine a presenter who forgot to bring her presentation remote. Without the remote, she would be rooted to the computer to control the presentation, thereby limiting her freedom from using expressive body language. To improvise, she grabs a pen, pushes its button while hitting the “next”-key on the keyboard to pair the control. From now on she can press the pen button to advance the slides remotely (Figure 7.1). This “clicker” is unobtrusive and can be used eyes-free. We call such improvised and ubiquitous controllers *Instant User Interfaces.*

## 7.2 Instant User Interfaces

We define an *Instant UI* as: “a user interface that lets a user select a physical object within reach to control a technical system by establishing mappings in an ad hoc manner from object to system based on end-user programming. Instant UIs map affordances of physical objects to their digital counterparts such as buttons, knobs, or sliders.” (Corsten et al. [2013], Figure 7.2).

UbiComp, tangible and organic user interfaces are closely related.

The general idea of combining real world and computing has been explored for a long time in HCI:

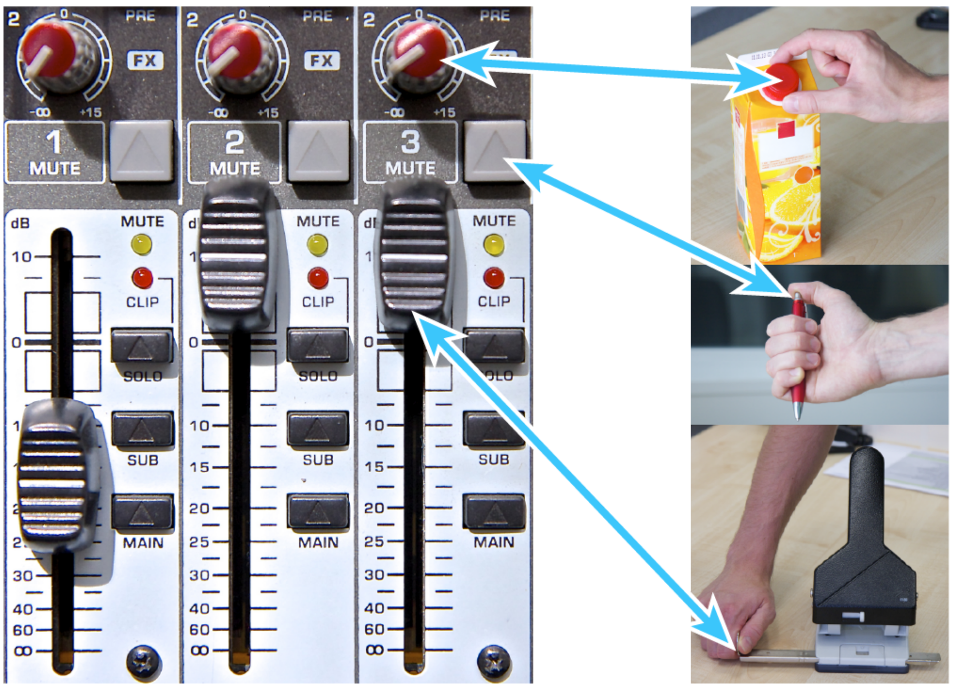


**Figure 7.2:** Everyday objects, such as a juice carton, a pen, or a hole puncher, provide controls with physical affordances (rotary knob, push button, slider) as used in dedicated input devices, such as a mixer. Taken from Corsten et al. [2013].

1. Ubiquitous Computing suggests that computing devices should "weave themselves into the fabric of everyday life" (Weiser [1995]). Similarly, Instant UIs use everyday objects but do not need fabricated objects.

2. Tangible User Interfaces (TUIs) "couple digital information to everyday physical objects" (Ishii and Ullmer [1997]). TUIs are similar in the fact that they rely on real world objects rather than digital artifacts, but they also require specially fabricated objects.

3. Organic User Interfaces, as suggested by Holman and Vertegaal [2008], consider objects with non-planar displays and inputs as do Instant UIs, but focus on a built-to-purpose approach.

### 7.2.1 Scenarios

Several scenarios can benefit from InstantUIs.

Our initial example with the pen during the presentation is just one scenario where the ad hoc nature of Instant UIs can be beneficial:

**Improvisation.** When a controller is missing, the user can pick up a nearby object and repurpose it like the missing control.

**Convenience.** Even if the controller exists, it might be out of close reach. Instant UIs can replace them for a more mobile or remote use and can be placed anywhere. Imagine lying in your bed at home, but you forgot to turn off the lights, and the switch is at the door. You could repurpose a stapler once and put it next to the bed, such that you can operate the light whenever desired even in the dark by pushing the stapler lever.

**Simplification.** Controllers such as a TV remote often provide a plethora of input options, although only a small subset of controls might be required for most users. Instant UIs can be used to select this subset and only copy those controls to one or multiple objects. This allows us to use objects with better ergonomics and clearer affordances.

**Duplication.** Instant UIs can also be used to create physical copies of existing controllers, such as creating additional joysticks for a multiplayer game console.

### 7.2.2 Alternatives to Instant UIs

All of the above scenarios could be supported by alternative technologies, and we will present some of them now.

Other options can only partly support our scenarios.

Touchscreens, e.g., used in smartphones, combine direct manipulation with a dynamic UI, allowing us to flexibly adapt to different use cases. Since smartphones are carried around by the user, the controller substitute is in direct reach. However, due to their flat screen, smartphones lack physical affordances and tactile feedback.

Speech interfaces are an obvious option for an ad hoc interaction; however, using spoken commands may not always be appropriate in some situations due to background noises or social awkwardness. Another option is to use hand gesturing in mid-air. However, this faces similar social awkwardness and, more importantly, problems of accidental triggering of events.

## 7.3 Related Work

Become independent of a dedicated physical controller.

Instant UIs want to make the user independent of a dedicated physical controller and there are two ways to achieve this. You can simply eliminate the physical controller and perform input via other modes, or you can repurpose any object as a physical controller.

Perform touch input on arbitrary surfaces without considering affordances.

As a hybrid of the two options, people can perform interactions on their body, such as touches in the systems Skinput by Harrison and Tan [2010] and Imaginary Phone by Gustafson et al. [2011]. In WorldKit by Xiao et al. [2013], the user also uses his hands to perform input, but is not restricted to its own body but can define any surface in the vicinity as an interactive touch area. Another object is to perform those gestural input in the air, as, e.g., Gustafson et al. [2010] did. The general problem here is that the interactive areas are not built on physical affordances of the objects and lack tactile feedback (especially with mid-air gesturing). Skinput and Imaginary Phone are independent of the user's visual attention but require her to recall the spatial layout of the invisible interface from memory during interaction and may therefore increase cognitive load.

Marker-based tracking cannot deal with occlusion.

Opportunistic Controls by Henderson and Feiner [2008], Smarter Objects by Heun et al. [2013], and iCon by Cheng et al. [2010] exploit existing physical controls or objects in the vicinity that are repurposed as input devices. For example, Opportunistic Controls exploit unused affordances of physical items in a mechanic's work space, such as ripped pipes or screws. However, they strongly rely on the use of markers which cannot be recognized when they are occluded. More related work can be found in our paper (Corsten et al. [2013]).

Although the idea of repurposing objects has been explored by a few projects in the past, the projects relied on markers which significantly hindered its actual use and the evaluation of systems.

## 7.4 Everyday Objects

Diary study to find out about everyday objects.

As a first step in exploring appropriation of everyday objects, one needs to know what common objects people will carry around or find in their surroundings. Therefore, we conducted a diary study.

### 7.4.1 Procedure

Participants were asked to take pictures of everyday objects.

In this study, participants were asked to take pictures of everyday objects in their reach during a regular work day and during a day while not being at work. They were asked to take multiple photos at different times of the day, which could result in different locations or events. Each participant was offered a text message notification service that reminded them to take pictures six times a day.

### 7.4.2 Analysis

Photos were analyzed and categorized.

For each participant, the photos were analyzed and categorized regarding objects that were in reach in the user's work environment in contrast to objects that were in reach while not being at work (e.g., while being in the bathroom, doing sports, having a drink, etc.). The photos were stitched into related scenes, and all object shapes were traced. Subsequently, we created a list of identified objects for each participant regarding photos taken during work hours vs. pictures taken during leisure hours. Multiple occurrences of the same object, e.g., stack of hand towels, pile of paper, etc. were counted as one occurrence for a user.

### 7.4.3 Participants

19 Participants with different background.

A total of 19 users, two females, took part in the diary study (aged 20-62, $M = 29.26$, $SD = 8.93$). The users had various professional backgrounds represented by a mayor, an architect, a project manager, a student of arts, a student of business administration, participants involved in media, and one person from the automobile industry. All other participants were computer science students or research assistants. Seven participants made use of the notification service.

### 7.4.4 Results

All but four participants took photos on both days. A total of 360 pictures were analyzed, and 144 different objects were identified, which were merged into a set of 98 distinct object categories. Hazardous objects, such as a razor, were removed from the result set.

Of all 19 participants, 16 (18) participants took photos in the working (leisure) environment. This led to 67 (88) different object categories. An overview of the most frequent objects can be found in Table 7.1.

Results confirm our initial expectations.

Considering all counted objects, we have 57 matches of occurrence (of at least once) between working environment and leisure environment items. Objects in the work environment were mostly desktop items typical for office jobs. Objects in the leisure environment were typically found in the living room, bathroom, or kitchen. In addition, among these objects, seven items were things participants usually carry with them: clothes, phone, pen, key, wallet, bag, watch. Interestingly, participants tended to photograph carried items more often while being in their leisure environment compared to working environment (seven vs. three items on average) although these objects are completely independent of the environment. Our guess is that participants were more busy during work times and that they felt that their personal (and mobile) objects did not belong to the work environment. The low number of phones was probably due to the fact that most participants (16) used a mobile phone with an included camera for taking the pictures, which consequently cannot appear in the photos.

## 7.5 Appropriations

While developing the tracking system in Chapter 5, we ran a Wizard-of-Oz study to see how people would use objects to control devices. This could

| Object Occurrences and Scenario | | | |
|---|---|---|---|
| Object Name | Leisure | Work | Overall |
| Table | 16 | 12 | 28 |
| Container ($\iota, \delta$) | 15 | 10 | 25 |
| Paper ($\iota, \delta$) | 10 | 13 | 23 |
| Bottle ($\iota, \delta$) | 12 | 10 | 22 |
| Drinking Vessel ($\iota, \delta$) | 11 | 9 | 20 |
| Pen ($\iota, \mu, \delta$) | 8 | 10 | 18 |
| Underlay ($\iota, \delta$) | 11 | 7 | 18 |
| Cable ($\iota, \delta$) | 7 | 10 | 17 |
| Book ($\iota, \delta$) | 7 | 9 | 16 |
| Laptop ($\delta$) | 5 | 10 | 15 |
| Phone ($\iota, \mu, \delta$) | 9 | 6 | 15 |
| Cloth ($\iota$) | 12 | 2 | 14 |
| Cutlery ($\iota$) | 11 | 3 | 14 |
| Clothes ($\iota, \delta$) | 10 | 3 | 13 |
| Plate ($\iota$) | 11 | 1 | 12 |
| Chair | 5 | 6 | 11 |
| Keyboard ($\delta$) | 2 | 9 | 11 |
| Bag ($\iota, \mu, \delta$) | 6 | 4 | 10 |
| Desk Lamp ($\iota$) | 7 | 3 | 10 |
| Wallet ($\iota, \delta$) | 7 | 3 | 10 |
| Key ($\iota, \mu$) | 8 | 1 | 9 |
| Mouse ($\delta$) | 2 | 7 | 9 |
| Spray Can ($\iota$) | 7 | 2 | 9 |
| Cushion ($\iota$) | 7 | 1 | 8 |
| Display ($\delta$) | 3 | 5 | 8 |
| Door | 6 | 2 | 8 |
| Puncher ($\delta$) | 2 | 6 | 8 |
| Remote Control ($\iota$) | 8 | 0 | 8 |
| Stapler ($\delta$) | 0 | 8 | 8 |
| Watch ($\iota, \mu$) | 6 | 1 | 7 |
| Stick ($\delta$) | 0 | 6 | 6 |
| White Board ($\delta$) | 0 | 5 | 5 |

**Table 7.1:** This table shows how often an object category occurred throughout the first study and is restricted to occurrences greater than 4. The greek characters show in which scenario of the second study the objects were used: $\delta$ for standard desktop GUIs, $\mu$ for mobile, and $\iota$ for indoor scenarios.

allow us to find out how everyday objects are used for appropriation and which gestures people apply to them. We will present its setup and the main results, full details can be found in Corsten [2012].

### 7.5.1 Setup

This user study covered qualitative aspects only. It was a combination of think-aloud, recorded observations, and a post-study questionnaire. In order to avoid blind angles, cameras were set up from two perspectives.

The study dealt with scenarios consisting of a sequence of simple tasks, such as "Go to next slide" or "Go back one slide", the participant was asked to solve using everyday objects. The scenarios were sorted in groups: Desktop for standard desktop GUIs, Mobile, and Indoor.

Objects used as identified previously.

The objects that were used in this study were identified beforehand from a set of 360 pictures taken by people in two locations: at work and at home (Fig. 7.3). Due to variation in physical affordances, we included different objects variants in this study, e.g., for a drinking vessel we included a drinking glass, a porcelain cup, and a paper cup.

The scenarios were presented using sketched slide sets. Although our Wizard-of-Oz system did not feature realtime feedback, the slides were updated after the user performed a task.

### 7.5.2 Scenarios

Scenarios were presented on slides.

Desktop scenarios consisted of simple GUIs with: (i) three buttons, (ii) three radio buttons, (iii) two check boxes, (iv) two rotary buttons, (v) three horizontal color sliders, and (vi) a spreadsheet with two columns, ten rows, and a horizontal and a vertical scroll bar. Users had to switch focus, activate buttons, change states, and navigate. The single mobile scenario asked the user to manipulate a 3D rendering of glasses. Finally, the indoor scenarios asked the user to (i) navigate presentation slides, (ii) control a TV, (iii) play a 2D jump'n'run game, and (iv) operate lights.

### 7.5.3 Procedure

Initial warm-up phase.

First, the participant was given the chance to inspect all everyday objects used in this study. All objects were placed on tables and on a window sill in a random layout. The participants had a good overview of all objects from a sitting or standing position. Having explained that our Wizard-of-Oz system is capable of recognizing any interaction with the objects, the first block of scenarios was presented to the participant. Tasks were faded in one after another such that the participant was unable to see which

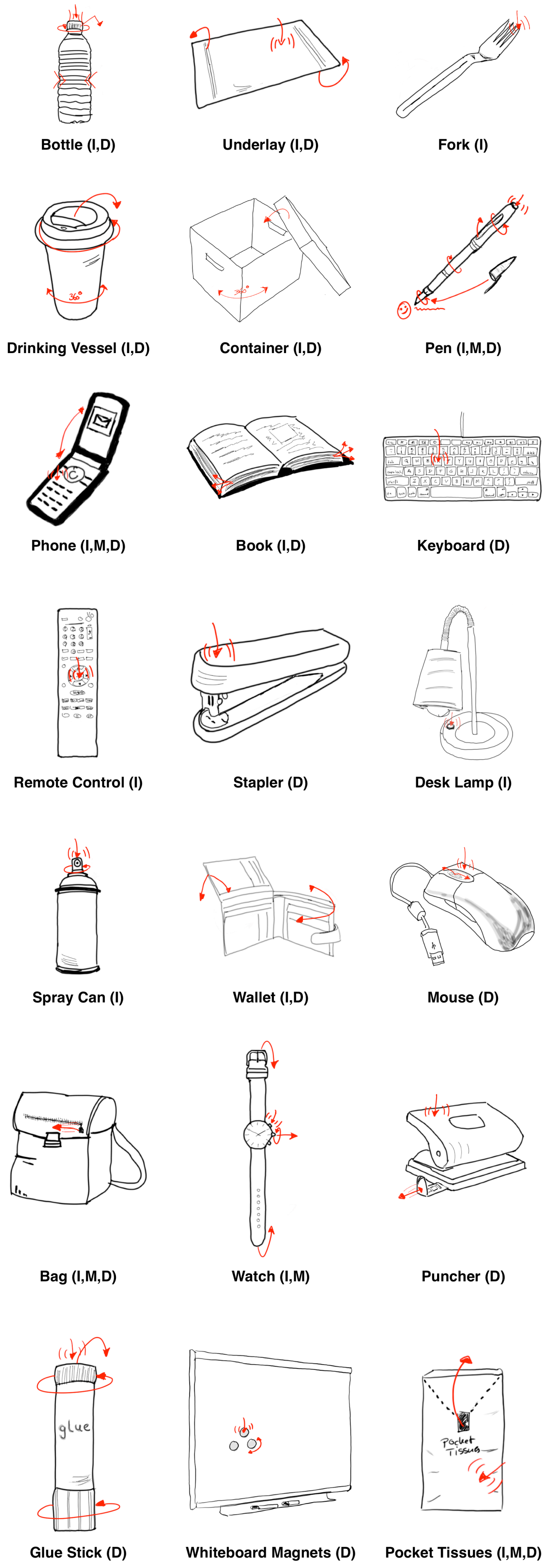


**Figure 7.3:** Everyday objects used in the affordance study for the (D)esktop-, (I)ndoor-, or (M)obile scenario. Red arrows show the actual manipulation behavior: The object was used at least once in this way.

tasks were to follow. The user was asked to pick one or more objects and to demonstrate which gesture she would apply to the items. New objects could be picked at any time.

Counter-balanced scenario order.

After each scenario, all objects were put back in order to refresh the setting. The order of the scenario blocks was counter-balanced. Also, the sequence of scenarios within a block was randomized. All participants evaluated all scenarios and filled out a post study questionnaire. The whole process took about 40 minutes per person.

### 7.5.4 Analysis

Only consider first solutions.

For reasons of spontaneity, only the first solution presented by a user was regarded. Incomplete solutions were filtered out in case the user also presented a complete set of mappings for the scenario. For each scenario, we identified the most frequent object-gesture-action mappings, also referred to as patterns: Whenever the same task solution was observed in three or more users, it was denoted by a pattern.

*Pattern Example:* The user is about to hold a presentation but forgot to bring the remote control. Tasks: 1. Go to the next slide, 2. Go one slide back. Patterns: (a) *ball pen*: 1. Push the pen button once, 2. Push the pen button twice (b) *mobile phone*: 1. Push the right button on the directional pad, 2. Push the left button on the directional pad, (c) *hardcover book*: 1. Flip one page forward, 2. Flip one page backward.

### 7.5.5 Results

18 participants, aged 20–28 ($M$ = 24.39, $SD$ = 2.15), 16 male, participated in this study. Besides a teacher, only students participated in the study, 15 from computer science.

Most users were able to find task solutions.

Most users were able to find task solutions without experimenting with several objects. Ten out of 18 users handled the tasks by using just one object per task. Using multiple objects simultaneously was rare.

They often used physical instantiation.

In the desktop scenarios, participants principally tried to imitate the missing input device through button-equipped everyday objects (Fig. 7.4). However, one third of the testers tried to rebuild a GUI with objects by looking for similar shapes and by adopting the layout of the UI widgets. For instance, in the presentation scenario, three testers appropriated the pages of a book as physical instances of the virtual slides. We define this as **Physical Instantiation**: *It is the act of using a physical object as representative substitute of either a physical or virtual target for remotely directly controlling it. Manipulation of the substitute is supposed to identically effect on the remote original.*

Continuous controls were often instantiated with one object.

This behavior was most prominent for the rotary buttons and sliders, the only continuous controls. As regards the rotary buttons, 15 users needed two objects to solve the tasks; for the slides, seven used three objects for controlling the widgets. Thus, these two scenarios forced most participants to physically instantiate each UI widget with a single object. Moreover, selecting the right widget for manipulation felt much easier by applying Physical Instantiation: *"Selection [of each rotary button] is difficult. I can solve that by picking two distinct items and turn each of them individually."* said one of the participants.

In case Physical Instantiation was not applied, participants used either a special object or a special gesture to sequentially switch the focus from one UI widget to another.

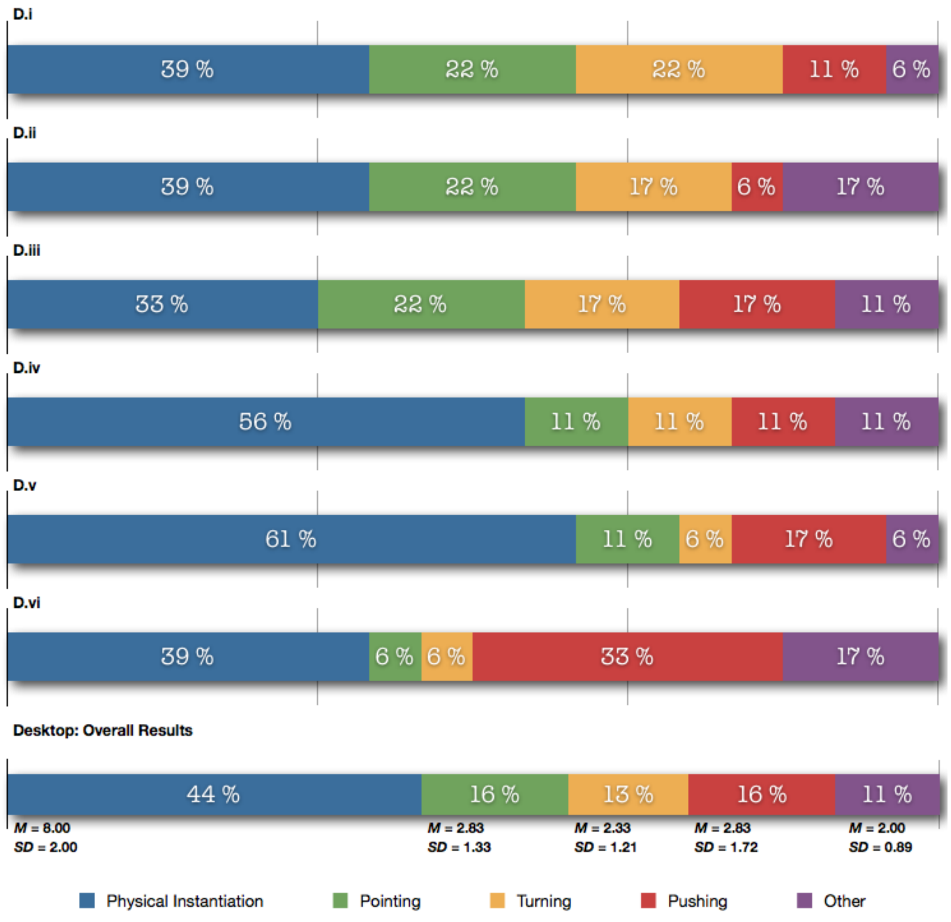


**Figure 7.4:** Focus change usage patterns for desktop scenarios. Many participants rebuilt the UI by applying Physical Instantiation. Others pointed or turned to change the focus. Rotary button and slider were perceived as continuous controls. This resulted in a higher tendency to rebuild the UI.

Shape was an important object property.

In the desktop and indoor scenario, shape was probably the most important object property for deciding which object to pick. Participants imitated either the shape of the input controller or the target that was to be controlled. Besides, some participants looked for colors on objects that could be associated with a certain action: some looked specifically for the red button to shoot flames in the jump'n'run game.

Pointing was a natural way of activating an object. It was used to indicate that object manipulation should have an effect on the system that was being pointed at. Physical Instantiation was more likely to be applied in case continuous input was expected, such as slider navigation or rotation.

After the study, we asked participants to assess the ease of (a) finding objects for a given scenario and (b) subsequently finding gestures to be applied to them for solving the tasks. Users easily found object-gesture-action mappings for all but one task: manipulating a spreadsheet, which seems to be the ceiling for this kind of interface. We do not see Instant UIs applicable to even more complex tasks.

## 7.6 Live Interactions

Instant UIs span a vast design space.

We performed another pre-study using a very limited depth-tracking approach that did not allow object movement. Despite the limitations, we got very interesting results. The patterns which objects people used and how they used them varied a lot between application context, task, and each user. This means Instant UIs span a vast design space, and a single exploratory study will not result in facts that can be applied to any set of objects or context.

End-user programming for mapping input to the interface.

Still the system must know how to interpret the input of the user. Artificial intelligence approaches using context information might be possible, but we think that end-user programming is a good solution here. Thus, we performed a comparison of three different ways of end-user programming for everyday objects, i.e., using speech, by demonstration, and with a GUI. This will also show us how users experience the new interaction metaphor of Instant UIs. The full details of the study can be found in our paper (Corsten et al. [2013]).

### 7.6.1 Setup

Speech, Demonstration, or dedicated GUI.

As a first step in the study, the user had to program the system by telling the system how to establish mappings from object to target interface, or by performing gesture on the object while simultaneously performing input on the dedicated control, or by using a dedicated GUI (Figure 7.5).

Navigate slides and operate a light.

During the study, users had to participate in two scenarios and perform a task: (A) navigate a slide-based presentation using a clickable pen, (b) operate a dimmable light using a mug (Figure 7.6). For the pen, the user could choose from a push (touch object for 400-1200ms) and a push-and-hold (touch longer than 1200ms) gesture and map those to the commands “next slide” and “previous slide”. For the mug, the user could choose between the handle facing the user and it facing away from the user, and map these positions to “light is at maximum brightness” and “light is turned off”.

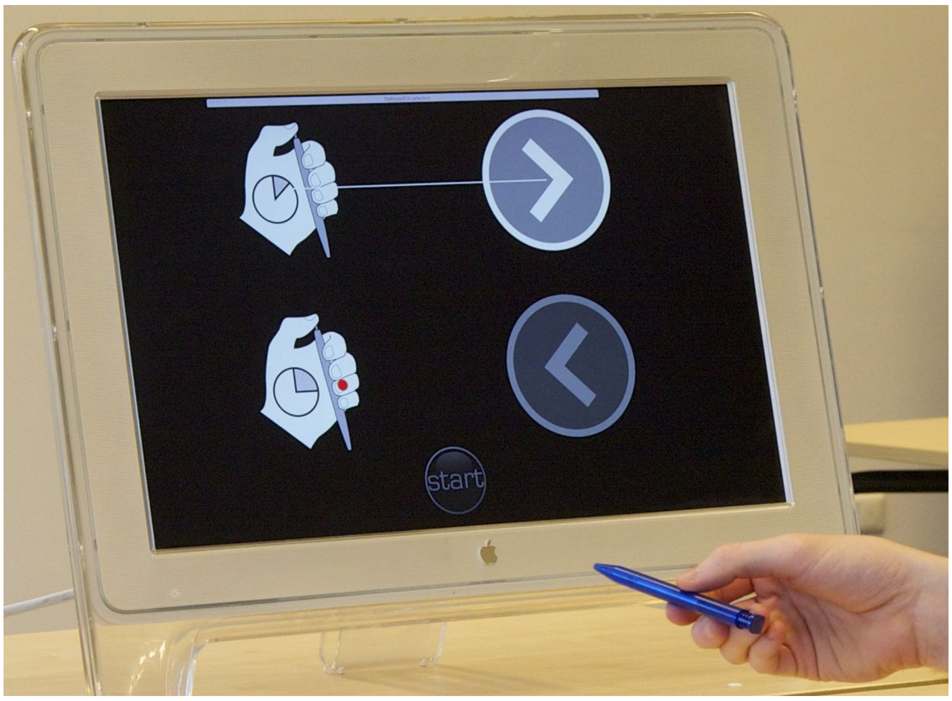


**Figure 7.5:** GUI for programming the pen as remote. Pointing with the object at the GUI controls a red cursor with which the user selects a pen gesture visualized on the left side of the GUI (push, push-and-hold) and maps it to an action from the target interface (right side: next slide, previous slide). Established mappings are indicated with a line. Programming was finished by selecting the "start" button at the bottom. Taken from Corsten et al. [2013].

## 7.6.2 Results

All participants could program the objects and finish the task.

All participants were able to complete the task in the expected time and were intrigued by the Instant UI concept. The post-study questionnaires also revealed that participants had diverse preferences for the different programming methods. Programming the pen in all three conditions was considered a simple approach by participants: "*It was simple—no complications.*". Programming the mug, however, was considered more difficult for demonstration and the GUI because the participants had to rotate the mug on a traverse plane while rotating the rotary knob on a frontal plane. Synchronized pushes (pen and keyboard), however, did not cause any difficulties for the users.

Discrete controls were easier to program than continuous controls.

Hence, programming by demonstration for continuous controls is more difficult than for discrete controls. Another difference became obvious during the testing: the mug was heavier, and thus it felt more tedious to control the demonstration GUI with it by pointing.

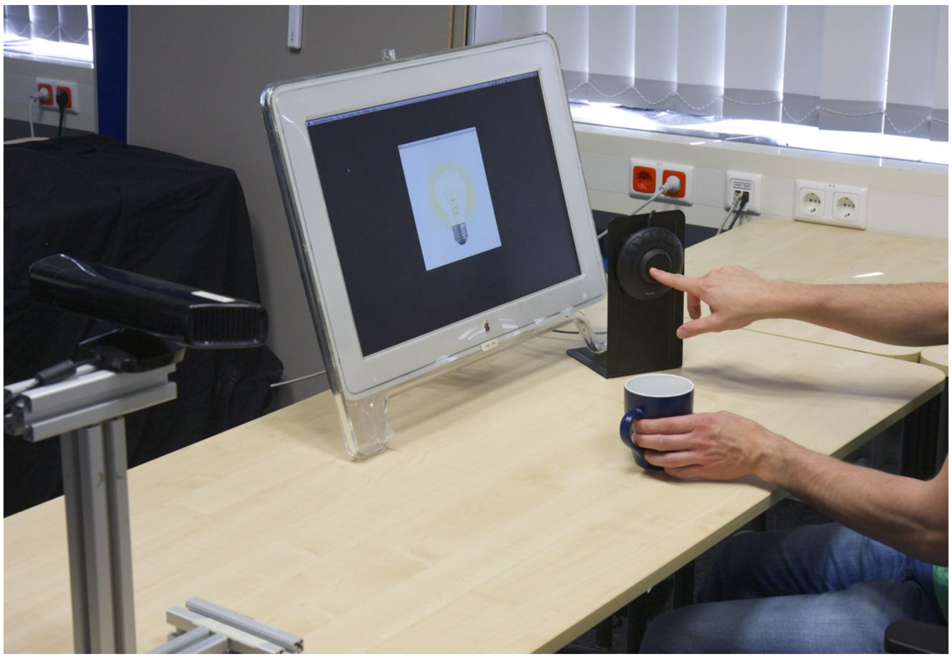


**Figure 7.6:** Setup of the system for the light scenario (left to right): Kinect, display, mug, rotary knob. The user is programming the mug by demonstration: mug and rotary knob used to control the light are rotated at the same time. From now on, the user can rotate the mug to dim the light visualized on the screen. Taken from Corsten et al. [2013].

## 7.7 Future Vision and Challenges of Instant UIs

Add projectors to provide feedback on the repurposed object.

Our system shows one way to implement Instant UIs: a depth-based camera installed in a room. We could easily extend this by adding directed projectors, microphones, and speakers similar to the work by Zerroug et al. [2011]. This system could assist users with visual and audible feedback during object programming mode, e.g., to confirm that a mapping has been established. Interaction with the remote system using the repurposed object does not need additional feedback, as long as the remote system (e.g., a light) already provides feedback (light is on/off).

Object descriptor baked into the object.

Currently, the end-user programming “framework” is strongly tailored to our objects and devices, and the question is how we should support the plethora of objects and devices that exists. Sterling [2005], e.g., envisions that within the next 30 years, each physical object, be it a bottle, a light switch, or a TV, will have a digital object descriptor or “aura”, that can be stored on an RFID chip attached to the object. We could store a 3D model of the object shape as well as afforded controls, such as knobs, buttons etc. If the object is a technical device, we could add an API descriptor for functions that can be accessed or called from outside.

Differentiate between repurposed and regular use.

Considering the current limitations, the main challenge is to differentiate between repurposed and regular use of an object. In general, this can be solved by setting the device into a mode by using a specific gesture. However, this is not always necessary because there can be significant differences how people use the object. Consider a stapler: For stapling paper, you will smash the stapler with your full fist. Instead, for appropriated use, we only saw people touching the top part gently in our studies. Here, the system can easily distinguish the two.

"Beautify" end-programming input.

In the study we saw that particularly demonstrating the continuous input was challenging for some users. Smoothing this input could be one option to ease the task for the user. Another question is how the user can enter the *enter* programming mode? Similarly to repurposing the object, this could be done either explicitly by using a dedicated gesture or implicitly when the system detects that the user manipulates the object in an untypical way.

## 7.8 Chapter Summary

In this chapter, we wanted to capture the expressiveness of touch on arbitrary objects. We did so by introducing a new interaction metaphor—Instant UIs—that makes use of the object specific affordances to more easily align the user's expectations with the object's reactions to touch input. We found out that in this context of mobile objects (in contrast to the immobile tabletops of the earlier chapters), the position and orientation of the object can and should also be taken into account because this was actually expected and preferred by the users. Additionally, we saw that in this very general interaction space, we can make use of end-user programming, e.g., by demonstration, to allow for quick setup of the objects.

Having seen how far we can and should extend the concept of a touch, we will now combine the results from the previous chapters and answer the question "What is a Touch?".

# Chapter 8

# The Expressiveness of Touch: A Conclusion

Widen the communication channel.

The overall goal of this thesis was to widen the communication channel between humans and computers in the context of touch interfaces in order to reduce erroneous touch detection and allow for more natural input metaphors. We achieved this by improving current knowledge and touch systems on three levels—Detect, Improve, and Extend—while continuously reevaluating what should be taken into account for touch interactions.

Capture expressiveness on tabletops.

We started out in the domain of interactive tabletops and realized that we have a rather poor understanding of how people perform touches and that we do not know whether we can apply sophisticated touch models from smaller form factors. After building a high-precision touch detection system, we were able to find out how the body posture affects the touch behavior and how touch sequences are different from single touches. Using our interactive tabletop as a 3D display shows another problem with the current use of touch because the direct manipulation metaphor conflicts with a perspective correct rendering. We did a thorough selection of alternatives and evaluated them.

Capture expressiveness on everyday objects.

In the second half of the thesis, we no longer restricted ourselves to touch on interactive tabletops but considered arbitrary objects as touch surfaces. For this, we developed a more general touch detection approach and suggested an algorithm to infer the hand posture from the raw touch data. As closure of the thesis and outlook of what the future could hold for interactive surfaces, we introduced the concept of repurposing everyday objects as input controllers.

## 8.1 What is a Touch?

In the introduction, we posed the overarching question of this thesis: "What is a touch?". We came to conclusion that current applications interpret this input as a $(x, y)$ coordinate. However, throughout the thesis, it became clear that this simple model is not able to capture the expressiveness of the touch and that we should extend it.

Contextual information can improve touch accuracy.

In "Improving Touch on Tabletops", we also took into account the previous touch location. This allowed us to improve touch accuracy. Similarly, we saw that different body postures result in systematic errors which could be used to predict errors, i.e., also increase touch accuracy.

Use head position as input channel.

We can not only use contextual information to improve the input but also as an additional input channel. We did this in "Extending Touch on Tabletops" by tying the display rendering to the head position, resulting in a 3D rendering. However, there we saw that both touch input and head position have an impact on the position of the object rendering. This concurrent input led to interaction errors. We solved this concurrency problem by allowing parallel input, yet slowly reducing the offset created by the one that is currently changing less.

Infer hand posture from touch information.

In Chapter 3, we added the finger ID as an additional property of the touch and provided means to infer the hand posture. This allows application to cater to the fact that the thumb and the other fingers have a different reach. Additionally, the hand posture combined with head position allows us to know which parts are occluded of the object and display information on the visible parts. Also, we can differentiate to which fingers to react at all because some can be just part of the holding gesture.

From a focus on 2D touch to touch as one input channel to touch as context.

So far we, only added attributes to our initial $(x, y)$ touch model: more information about the finger contact surface, the hand posture, and the full body posture. However, in Chapter 7, we are no longer interested in the exact $(x, y)$ and can replace it with the information which part of which object was touched. Obviously, we need the $(x, y)$ for calculating the touch region, but the user does not care for this: The affordances of the object influence which areas are considered as one area by the user and thus frame his mental model of the interaction.

With this new interpretation of "What is a touch?", we can divide the history of input into three stages: We came from the desktop GUI world, where mouse and keyboard provided ways to manipulate a cursor residing at a position $(x, y)$. Then, HCI moved to GUIs on touch, or pen-based surfaces, removed the cursor, and switched to a direct manipulation at a specific $(x, y)$ position. And now we are slowly leaving this coordinate-based input behind and are focussing on the touched objects and which parts of which object were manipulated.

## 8.2 Future Work

Extend detection to deformable objects.

We presented two touch detection frameworks, one for tabletops and one for arbitrary objects, and although both worked well enough for our studies, we can definitely see improvements. One (standard) option is to improve accuracy and frame rate of the detection. Better recognition of rigid objects, adding recognition of material deformation as an input, or a less static camera setup could be future extensions. Additionally, we think that more visual feedback on the objects themselves could help perceiving the application's state, help to know which parts of the objects can be used for input, or even show what the input will result in.

Extend prediction models and improve data sets.

We used the predecessor to correct touch errors on a tabletop. We also saw systematic offset w.r.t. the body posture, and thus we should be able to use that as well to increase touch accuracy. The final goal would be to have an exact mathematical model that predicts the error based on touch location, predecessor touch, target size, distance between predecessor and current touch, and the relative angle.

Mixed methods for 3D interaction.

When comparing different methods to control objects on a 3D display, we saw that some performed better at specific occlusion or depths levels. We would like to differentiate between the two effects and then find ways to move from one to the other methods, creating mixed methods.

Add physics support to hand posture recognition.

In Chapter 6, we showed an algorithm to infer the hand posture. We would like to vary the various parameters we (arbitrarily) chose for the algorithm and see how they impact run-time and accuracy. It would also be very interesting to extend our hand model to support physical simulation, e.g., check for collisions of the object with hand, or whether bones "intersect" for a given gesture.

Differentiate between repurposed and regular use.

We introduced a new interaction metaphor—Instant UIs— and got great feedback from our user studies. However, to apply this in an actual context, we need to take a closer look at how one can differentiate between the repurposed use and the original use of the everyday object. When programming the input mapping by demonstration, how much filtering should the system perform to generate the right gestures? For example, mapping the mug turning to jog wheel turning challenged the user's dexterity.

Create an information aura for objects.

As most of the objects that surround us are industrially produced, a 3D model of most everyday object already exists. But this means, that based on traditional tagging schemes, e.g., barcodes or RFID, we can imagine that this model could be looked up. Additionally, information about how this object can be used and also how it should be used, can be integrated, creating an "interaction aura". This information could either be provided by the original manufacturer as an added benefit or by a community of people interested in appropriating objects.

Apply "detect, improve, extend" to other input channels.

Throughout this thesis, we extended and replaced the attributes that are considered as a touch. It would be interesting to see whether different kinds of input can benefit from a similar treatment. For example, in mid-air gesturing, the user's input is typically mapped to a 2D or 3D cursor that then acts as an input channel for a GUI. Instead, we could take into account the user's viewport, i.e., its eye position, and directly detect which (virtual) object he is looking at and trying to interact without a need for a cursor.

## 8.3 Closing Remark

The interface is the product.

Providing the right input channels is crucial for a widespread adoption of computing devices in our society—as can be seen by the success of direct manipulation devices such as the iPhone and the iPad. We as interface designers and application developers should therefore always ask ourselves whether we are reducing the human output to the right set of information: it is intriguing to only pick aspects that are easy to track by our hardware or easy to write software for.

We showed several examples where additional, contextual information provided great benefit for the user, and we are excited to see how other input, such as gestural or voice-based, will be extended in the future.

# Appendix A

# Details of the Hand Model

The hand model was adapted from Cobos et al. [2008]. Exact measures were extracted from the anatomy handbook by Schmidt and Lanz [2004]. Measures are given in centimeters and use right-handed X-Y-Z coordinate system.

We are modeling the right hand, and the palm is lying flat on the table.

| | TMC/CMC (flex) | TMC/CMC (adduct) | MCP (flex) | MCP (adduct) | PIP (flex) |
|---|---|---|---|---|---|
| thumb | (-15,60) | (-30,30) | (0,80) | (0,0) | (-10,80) |
| index | (0,5) | (0,0) | (-30,90) | (-10,10) | (0,110) |
| middle | (0,5) | (0,0) | (-30,90) | (-10,10) | (0,110) |
| ring | (0,10) | (0,0) | (-30,90) | (-10,10) | (0,120) |
| little | (0,15) | (0,0) | (-30,90) | (-15,15) | (0,135) |

**Table A.1:** Range of possible curvatures for each joint as flex or adduction rotation angles. TMC = trapeziometacarpal, CMC = carpometacarpal, MCP = metacarpophalangeal, PIP = proximal interphalangeal joint. The distal interphalangeal joint flexion was always set to two thirds of the pip joint flexion.

| | palm | metacarpal | proximal | middle/distal | distal |
|---|---|---|---|---|---|
| thumb | $\mathbf{(2.5, 1.5, 1.5)}$ | $\mathbf{(3, 4, 0)}$ | $\mathbf{(1.5, 2.5, 0)}$ | $\mathbf{(1.5, 2, 0)}$ | $\mathbf{(0, 0, 0)}$ |
| index | $\mathbf{(2, 2, 0)}$ | $\mathbf{(1, 7, 0)}$ | $\mathbf{(0, 4.2, 0)}$ | $\mathbf{(0, 2.6, 0)}$ | $\mathbf{(0, 1.9, 0)}$ |
| middle | $\mathbf{(0.5, 2, 0)}$ | $\mathbf{(0, 6.9, 0)}$ | $\mathbf{(0, 4.7, 0)}$ | $\mathbf{(0, 3.2, 0)}$ | $\mathbf{(0, 1.9, 0)}$ |
| ring | $\mathbf{(-0.5, 2, 0)}$ | $\mathbf{(-1, 6.2, 0)}$ | $\mathbf{(0, 4.5, 0)}$ | $\mathbf{(0, 3, 0)}$ | $\mathbf{(0, 2, 0)}$ |
| little | $\mathbf{(-2, 2, 0.5)}$ | $\mathbf{(-2, 5.5, 0)}$ | $\mathbf{(0, 3.5, 0)}$ | $\mathbf{(0, 2.5, 0)}$ | $\mathbf{(0, 1.8, 0)}$ |

**Table A.2:** Bone length and directions. The Palm is considered as one bone for each finger.

# Bibliography

Maneesh Agrawala, Andrew C. Beers, Ian McDowall, Bernd Fröhlich, Mark Bolas, and Pat Hanrahan. The Two-User Responsive Workbench: Support for Collaboration Through Individual Views of a Shared Space. *Proc. SIGGRAPH '97*, pages 332–388, 1997.

Pär-Anders Albinsson and Shumin Zhai. High Precision Touch Screen Interaction. *Proc. CHI '03*, pages 105–112, 2003.

Ignacio Avellino. A Framework for Inexpensive and Unobtrusive Tracking of Everyday Objects and Single-Touch Detection. Master's thesis, RWTH Aachen University, Aachen, April 2013.

Patrick Baudisch and Gerry Chu. Back-of-Device Interaction Allows Creating Very Small Touch Devices. *Proc. CHI '09*, pages 1923–1932, 2009.

Jon Louis Bentley. Multidimensional Binary Search Trees in Database Applications. *IEEE Transactions on Software Engineering*, 5:18, 1979.

Paul J Besl and Neil D McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 586–606, 1992.

Christopher M Bishop. *Pattern Recognition and Machine Learning.* Springer, 2007.

Doug A Bowman. *3D User Interfaces.* theory and practice. Addison-Wesley Professional, 2005.

Kai-Yin Cheng, Rong-Hao Liang, Bing-Yu Chen, Rung-Huei Liang, and Sy-Yen Kuo. iCon: Utilizing Everyday Objects as Additional, Auxiliary and Instant Tabletop Controllers. *Proc. CHI '10*, pages 1155–1164, 2010.

Salvador Cobos, Manuel Ferre, M A Sanchez Uran, Javier Ortego, and Cesar Pena. Efficient Human Hand Kinematics for Manipulation Tasks. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2246–2251. IEEE, 2008.

Christian Corsten. Co-Optjects: Instant User Interfaces through Everyday Objects. Master's thesis, RWTH Aachen University, Aachen, January 2012.

Christian Corsten, Ignacio Avellino, Max Möllers, and Jan Borchers. Instant User Interfaces: Repurposing Everyday Objects as Physical Input Devices. In *Proc. ITS '13*, New York, NY, USA, 2013. ACM.

Norbert Dumont. The Impact of Body Posture on Touchtable Accuracy. Master's thesis, RWTH Aachen University, Aachen, June 2012.

Ali Erol, George Bebis, Mircea Nicolescu, Richard D Boyle, and Xander Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108:52–73, 2007.

Chin-Shyurng Fahn and Herman Sun. Development of a Data Glove With Reducing Sensors Based on Magnetic Induction . *Industrial Electronics*, 2005.

Paul M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

Mayank Goel, Jacob O Wobbrock, and Shwetak Patel. GripSense: Using Built-In Sensors to Detect Hand Posture and Pressure on Commodity Mobile Phones. In *Proc. UIST '12*, pages 545–554, 2012.

Sean Gustafson, Daniel Bierwirth, and Patrick Baudisch. Imaginary Interfaces: Spatial Interaction with Empty Hands and without Visual Feedback. In *Proc. UIST '10*, pages 3–12, 2010.

Sean Gustafson, Christian Holz, and Patrick Baudisch. Imaginary Phone: Learning Imaginary Interfaces by Transferring Spatial Memory from a Familiar Device. *Proc. UIST '11*, pages 1–10, October 2011.

M Hammerton and A H Tickner. An Investigation into the Comparative Suitability of Forearm, Hand and Thumb Controls in Aquisition Tasks. *Ergonomics*, 9(2):125–130, March 1966.

Jefferson Y Han. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection. *Proc. UIST '05*, pages 115–118, 2005.

Chris Harrison and Desney Tan. Skinput: Appropriating the Body as an Input Surface. *Proc. CHI '10*, pages 453–462, 2010.

Chris Harrison, Julia Schwarz, and Scott E Hudson. TapSense: Enhancing Finger Interaction on Touch Surfaces. *Proc. UIST '11*, pages 627–634, October 2011.

Steven J Henderson and Steven Feiner. Opportunistic Controls: Leveraging Natural Affordances as Tangible User Interfaces for Augmented Reality. In *Proc. VRST '08*, pages 211–218, New York, New York, USA, 2008. ACM Press.

Herbert Heuer and Steven W Keele. *Motor Skills*, volume 2 of *Handbook of Perception and Action.* Academic Press, May 1996.

Valentin Heun, Shunichi Kasahara, and Pattie Maes. Smarter Objects: Using AR Technology to Program Physical Objects and their Interactions. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, page 961, New York, New York, USA, 2013. ACM Press.

David Holman and Roel Vertegaal. Organic User Interfaces: Designing Computers in any Way, Shape, or Form. *Communications of the ACM*, pages 48–55, 2008.

Christian Holz and Patrick Baudisch. The Generalized Perceived Input Point Model and How to Touble Touch Accuracy by Extracting Fingerprints. *Proc. CHI '10*, pages 581–590, 2010.

Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. *Proc. CHI '97*, pages 234–241, 1997.

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proc. UIST '11*, October 2011.

Jintae Lee and Tosiyasu L Kunii. Model-based Analysis of Hand Posture. *IEEE Computer Graphics and Applications*, 15(5):77–86, 1995.

Kexi Liu, Dimosthenis Kaleas, and Roger Ruuspakka. Prototyping Interaction with Everyday Artifacts: Training and Recognizing 3D Objects via Kinects. In *Proc. TEI '12*, pages 241–244. ACM, 2012.

I Scott MacKenzie and WIlliam Buxton. Extending Fitts' Law to Two-Dimensional Tasks. In *CHI '92*, pages 219–226, New York, New York, USA, 1992. ACM Press.

Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring, and Saul Greenberg. Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TouchID Toolkit. *Proc. ITS '11*, pages 21–30, November 2011.

Nobuyuki Matsushita and Jun Rekimoto. HoloWall: Designing a Finger, Hand, Body, and Object Sensitive Wall. In *Proc. UIST '97*, pages 209–210, New York, New York, USA, 1997. ACM Press.

Max Möllers, Patrick Zimmer, and Jan Borchers. Direct Manipulation and the Third Dimension: Co-Planar Dragging on 3D Displays. In *the 2012 ACM international conference*, pages 11–20, New York, New York, USA, 2012. ACM Press.

Max Möllers, Norbert Dumont, Stefan Ladwig, and Jan Borchers. Improving Touch Accuracy on Large Tabletops using Predecessor and Successor. In *the SIGCHI Conference*, pages 755–758, New York, New York, USA, 2013. ACM Press.

Atsuo Murata. Empirical evaluation of performance models of pointing accuracy and speed with a PC mouse. *International Journal of Human-Computer Interaction*, 8(4):457–469, October 1996.

Atsuo Murata and Hirokazu Iwase. Extending Fitts' law to a three-dimensional pointing task. *Human Movement Science*, 20(6):791–805, December 2001.

Donald A. Norman. The Psychology of Everyday Things. New York, 1988.

Chavdar Papazov and Darius Burschka. An efficient RANSAC for 3D Object Recognition in Noisy and Cccluded Scenes. In *Proc. ACCV '10*. Springer-Verlag, November 2010.

David A Rosenbaum, Frank Marchak, Heather Jane Barnes, Jonathan Vaughan, James D Slotta, and Matthew J Jorgensen. *Constraints for action selection: Overhand versus underhand grips.* Lawrence Erlbaum Associates, Inc, 1990.

Hans-Martin Schmidt and Ulrich Lanz. *Surgical Anatomy of the Hand* . Thieme, Stuttgart, Germany, 2 edition, January 2004.

Johannes Schöning, Jonathan Hook, Tom Bartindale, Dominik Schmidt, Patrick Oliver, Florian Echtler, Nima Motamedi, Peter Brandl, and Ulrich Zadow. Building Interactive Multi-touch Surfaces. In *Tabletops - Horizontal Interactive Displays*, pages 27–49. Springer London, London, March 2010.

Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, March 1978.

Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, 1983.

Bruce Sterling. Shaping Things. MIT Press, September 2005.

Yoshiki Takeoka, Takashi Miyaki, and Jun Rekimoto. Z-touch: An Infrastructure for 3D gesture interaction in the proximity of tabletop surfaces. In *ACM International Conference*, pages 91–94, New York, New York, USA, 2010. ACM Press.

Dimitar Valkov, Frank Steinicke, and Gerd Bruder. 2D Touching of 3D Stereoscopic Objects. *Proc. CHI '11*, pages 1353–1362, 2011.

Radu-Daniel Vatavu and Ionuţ Alexandru Zaiţi. Automatic recognition of object size and shape via user-dependent measurements of the grasping hand. *International Journal of Human-Computer Studies*, 71(5):590–607, May 2013.

Simon Voelker. BendDesk: Seamless Integration of Horizontal and Vertical Multi-Touch Surfaces in Desk Environments. Master's thesis, RWTH Aachen University, Aachen, June 2010.

Daniel Vogel and Patrick Baudisch. Shift: A Technique for Operating Pen-Based Interfaces Using Touch. *Proc. CHI '07*, pages 657–666, April 2007.

Feng Wang and Xiangshi Ren. Empirical Evaluation for Finger Input Properties In Multi-touch Interaction. *Proc. CHI '09*, pages 1063–1072, 2009.

Robert Wang, Sylvain Paris, and Jovan Popović. 6D Hands: Markerless Hand Tracking for Computer Aided Design. In *Proc. UIST '11*, 2011.

Colin Ware, Kevin Arthur, and Kellogg S Booth. Fish Tank Virtual Reality. *Proc. INTERCHI '93*, pages 37–42, May 1993.

Mark Weiser. The Computer for the 21st Century. *Scientific American*, 1995.

Malte Weiss. *Bringing Haptic General-Purpose Controls to Interactive Tabletops.* PhD thesis, RWTH Aachen University, 2012.

Douglas Brent West. *Introduction to Graph Theory.* Pearson College Division, February 2008.

Andrew D Wilson. Using a Depth Camera as a Touch Sensor. In *ITS '10: International Conference on Interactive Tabletops and Surfaces*, pages 69–72. ACM Request Permissions, November 2010.

Raphael Wimmer and Patrick Baudisch. Modular and Deformable Touch-Sensitive Surfaces Based on Time Domain Reflectometry. In *Proc. UIST '11*, pages 517–526, New York, New York, USA, 2011. ACM Press.

Wenzhuo Wu, Xiaonan Wen, and Zhong Lin Wang. Taxel-Addressable Matrix of Vertical-Nanowire Piezotronic Transistors for Active/Adaptive Tactile Imaging. In *Science*, pages 952–957, April 2013.

Robert Xiao, Chris Harrison, and Scott E Hudson. WorldKit: Rapid and Easy Creation of Ad-hoc Interactive Applications on Everyday Surfaces. In *Proc. CHI '13*, pages 879–888, 2013.

Alexis Zerroug, Alvaro Cassinelli, and Masatoshi Ishikawa. Invoked computing. In *Proc. VRIC '11*, 2011.

Shumin Zhai, Jing Kong, and Xiangshi Ren. Speed–accuracy tradeoff in Fitts' law tasks—on the equivalency of actual and nominal pointing precision. *International Journal of Human-Computer Studies*, 61(6):823–856, December 2004.

Patrick Zimmer. Touching Illusions - Exploring Techniques for Direct Touch Interaction on 3D Tabletop Displays. Master's thesis, RWTH Aachen University, Aachen, November 2011.

# Index

Typeset November 29, 2013