RWTH AACHEN UNIVERSITY

The Impact of Body Posture on Touchtable Accuracy

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

by
Norbert Dumont

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Irene Mittelberg

Registration date: 09.12.2011
Submission date: 11.06.2012

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

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

Aachen, June2012
Norbert Dumont

Contents

List of Figures

Abstract

Touch input on multitouch screens is known to be inaccurate. This is generally explained by the *fat finger problem*; the softness of the fingertip prevents the system from detecting precisely the touch position, while the occlusion of the target by the finger prevents the users from getting feedback from the target. Holz emitted another explanation for the inaccuracy of the touch input and introduced the *perceived input point* model. This model states that the system detects touches at an offset from the intended target, and that this offset depends on the finger position when touching the surface.

In this thesis, we expanded on Holz' model and investigated the impact of touch sequences on the finger orientation. We formulated two hypotheses; the predecessor touch in a sequence influences the finger orientation and the touch location on the next button and the successor touch influences the finger orientation and the touch location on the previous button. We named these two hypotheses the *predecessor effect* and the *successor effect*.

A literature review revealed several techniques created to circumvent the accuracy. It also gave some example of finger orientation or body posture use in combination with multitouch tables. Different systems for obtaining the finger orientation are presented and were used to decide on our own system. We refined our hypotheses through an iterative process, during which we designed the experiment to verify our hypotheses. The study design, as well as the system components went through several iterations. Our system includes a high definition touch detection prototype and a software presenting the touch sequences to enable us to test our hypotheses and logging the touch data.

We tested our hypotheses by running a study which validated them. The predecessor effect on the finger orientation and the touch location was clearly seen, while no proof of the existence of the successor effect was seen. The predecessor effect could help making touchscreen devices more accurate.

Acknowledgements

I would like to thank my thesis advisors Prof. Dr. Jan Borchers and Prof. Dr. Irene Mittelberg for their guidance and assistance.

Special thanks to Max Möllers, for without his invaluable guidance and competent advice throughout this thesis, this work would not have been possible.

I owe Stefan Ladwig debt of gratitude for his tremendous help in explaining me understanding statistics and making sense of the data we collected.

René Bohne and Mariana Bocoi deserve my thanks, as their knowledge and experience in the FabLab helped me a lot when working on the hardware for this project.

My thanks to Simon Völker for his help about the MultiScreen agent, which made this work possible, and to Chatchavan Wacharamanotham whose opinion and feedback were always useful.

Last, but not least, I am heartily thankful to my parents, who always believed in me and supported me throughout my studies. To them, I dedicate this thesis.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

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

`myClass`

The whole thesis is written in British English.

Chapter 1

Introduction

Multitouch devices have made their way into our daily life. Their smaller versions at least. Mobile phones, tablets, laptops and desktop computers with multitouch screens are common products. Although it is possible to buy large multitouch devices, or even build them by hand, they are not as common as the other ones. Over the years, different technologies have appeared and evolved, enabling tabletops to detect touches in a reliable way. However, touch interactions on the surface lack the precision other input techniques have. The work presented in this thesis aims at improving the accuracy on touch tables by making use of the body posture.

Large Multitouch surface lack accuracy.

In this chapter, we will present the current state of the art in multitouch tables, the different sensing technologies available, their limitations, the motivation of this thesis and we will conclude with an overview of its content.

1.1 State of the Art Touch Detection in Tabletops

There are different technologies available to build multitouch surfaces. The main ones are resistive, capacitive and vision-based.

How resistive screens work.

Resistive screens are made of two conductive sheets separated by an insulated layer. When the screen is touched by any object, the conductive sheets come in contact and the precise location of the touch is registered. Resistive screens can be multitouch, although they are predominantly single touch. Their main application is in trackpads, mobile phones, or monitors. This technology is not used for table sized screens.

How capacitive screens work.

Capacitive screens consist of one conductor layer. When a finger touches the surface, the magnetic field of the person disturbs its electrostatic field. Capacitive screens are multitouch and are common in mobiles phones, tablets, and monitors. The DiamondTouch[1] uses this technology to sense the users's touch on the surface. The surface is illuminated by a video projector attached above the surface.

Two IR light based multitouch tables: using DI or FTIR.

Optical technologies are the preferred choice when building multitouch table, mainly because they are cheaper to build compared to the other technologies. The two possible choices are *Frustrated Total Internal Reflection (FTIR)* and *Diffuse Illumination (DI)*. Both use infrared light and cameras to register touches on the surface.

With DI, objects hovering above the table can be detected, with FTIR, they need to be on the surface.

FTIR diffuses IR light in the surface of the table, and when the finger touches it, the light is reflected downward, enabling the camera to see the touch. DI diffuses IR light from below the table, and when an object is placed on the surface, the light is reflected for the camera to see. The main difference between these two techniques is the fact that DI will see shadows of objects hovering above the table, while FTIR will only detect objects in direct contact with the surface.

1.2 Motivation

The fat finger as an explanation for inaccuracy on touchscreens.

Researchers argue that touch input cannot be accurate because of the *fat finger problem*; the finger occludes the target and the softness of the fingertip prevents to accurately hit small targets. The minimal target size for touch interaction

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

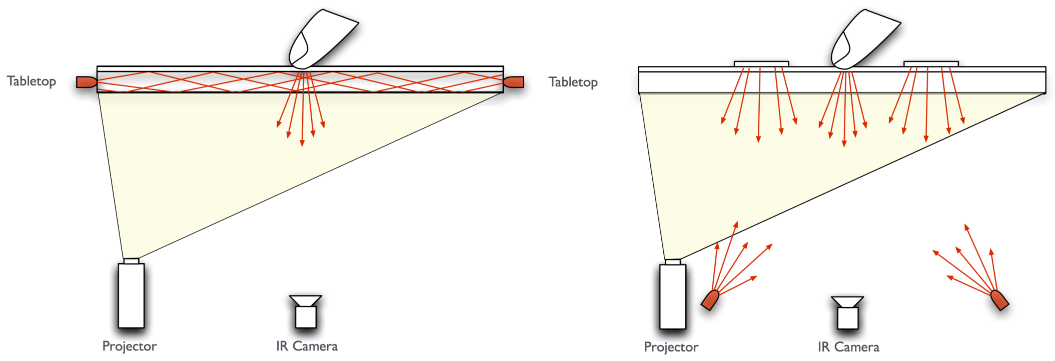


Figure 1.1: Left: FTIR setup, Right: DI setup.

has been noted as between 10.5mm [Vogel and Baudisch, 2007] and 26mm [Hall et al., 1988].

Holz' Generalized Perceived Input Point Model as another explanation for the inaccuracy on touchscreens.

This theory was refuted by Holz' et al. [2010]. In this publication, they argued that touch inaccuracy could be explained by a new model, the Generalized Perceived Input Point Model. Their hypothesis was that touches were registered at an offset from the intended target, and that the offset depended on the finger posture. We present their studies and results in the related work chapter (section 2.1).

We adapted Holz' experiment and expanded the model to multitouch tables.

We were interested in expanding Holz' work to multitouch tables, but with changes compared to their original setup. Holz' results had one major limitation; they showed that their model explained the touch inaccuracy and that it could be corrected, but to do so, Holz' team instructed the participants to be as accurate as possible when hitting the targets. We were interested to see whether the results could be reproduced in a more lifelike situation. When we did our study, we asked participants to hit the buttons in a casual way, without focusing on being accurate. Holz' based his model on the full finger posture, using the pitch, roll and yaw, while we used only the latest in our study.

Use the body posture to expand the model.

Our main hypothesis was that Holz' model could be expanded and that the whole body posture could be used to explain the offsets. To this end, we designed an experiment where participants had to hit targets on a surface. We developed pieces of software to detect the touches on the surface, to move a camera with very high resolution, and to display the targets on the table. During the first months of this work, our hypothesis evolved, and we focused on the impact of touch sequences on the finger orientation and the touch location on the surface. We believe that our findings can help increase the accuracy of everyday touchscreen devices.

1.3 Thesis Overview

The next chapter presents the related work.

The following chapter presents the related work we used for this thesis. It presents in detail Holz' study [Holz and Baudisch, 2010] which stands as the foundation for our work. Additionally, some targeting systems developed to solve the touch inaccuracy issue are introduced, followed by systems using finger and body posture, and finally by techniques enabling the detection of finger orientation.

Hypotheses and study refinement.

Chapter 3 describes our process when selecting our hypotheses and designing the study. It presents the different iterations we went through. It begins with our paper prototype, and how we used the results from the study we ran with it to refine our hypotheses. Then we describe the different studies we ran and how each new iteration was based on the previous one.

Implementation of our system.

Chapter 4 introduces the different software components we developed for this project; the touch detection agent, the moving camera agent with the hardware and firmware, and the software presenting the stimuli during the study.

Results of our main study.

Chapter 5 details the results of our main user study, and answer the question about the predecessor effect and successor effect on the finger orientation and the touch offset.

Summary and future work.

Chapter 6 summarize the work in this thesis, and presents several open questions that can be researched in the future.

Chapter 2

Related work

The inaccuracy of the touches detected by systems using the touch area center point has been linked to the so-called *fat finger problem* [Holz and Baudisch, 2010]; "the softness of the fingertip combined with the occlusion of the target by the finger". This prevents the targets from getting below a certain size or too close from one another. It has been noted by researchers that the minimum target size for targets was between 10.5mm [Vogel and Baudisch, 2007] and 26mm [Hall et al., 1988].

The fat finger problem as the cause for touch input inaccuracy.

In this chapter, we will present another hypothesis for the touch input inaccuracy and present targeting aid systems which have been implemented to circumvent this issue. We will also present systems that have made use of finger orientation or other body parts, and finally we will present techniques to detect the finger orientation.

Related work on finger orientation and touch accuracy.

2.1 The Generalized Perceived Input Point Model

Holz et al. [2010] formulated the hypothesis that the touch inaccuracy is not caused by the *fat finger problem*, and developed the *generalized perceived input point* model. This model is based on Vogel and Baudisch [2007] statement that users tend to perceive the selection point of their finger as being at the top of the fingertip while the system records the center of the touch area several millimeters below (see Figure 2.1). Holz argues that this offset is a systematic effect and can be compensated by adding a reverse offset to the touches. The *generalized perceived input point* model assumes that this offset depends not only on the location of the finger on the 2D surface, but also on the position of the finger in space. This position is analyzed through the use of the angle between the finger and the table; the roll, pitch and yaw. They also assumed that each user has a different mental model and thus needs a personal offset correction.

The Generalized Perceived Input Point model as another possible cause for the inaccuracy of touch input.

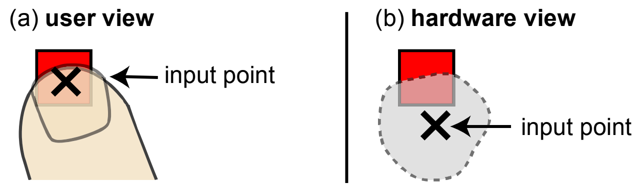


Figure 2.1: The perceived input point: (a) where the user expects the touch to be, (b) where the touch is recorded by the system [Vogel and Baudisch, 2007].

In order to verify their theory, they conducted a user study in which participants were asked to hit a crosshair repeatedly with five different levels of pitch. To limit the impact of other factors, the participants were asked not to move their head during the study to control for parallax and the touch was validated using a foot-switch to avoid inadvertent movements during take-off. The study was repeated for the finger roll and yaw. Figure 2.2 shows the results of the study for the roll and pitch effect for the first six users. The ovals contain 65% of the touches for a particular condition. One can make two observations from this chart; first, the five ovals of a user have a different offset, and second, the ovals distribution is different for each user.

A user study showed that different finger orientations lead to different clusters of touch offsets.

As the findings of the user study tend to support the model, they decided to build a prototype of a touch device that could extract these four parameters (roll, pitch, yaw and user). The *RidgePad* is a touch input device based on a fingerprint scanner. The fingerprint allows the system to differentiate between users and the portion of the fingerprint currently in contact with the surface. This enables the

RidgePad: a prototype using a fingerprint scanner to get the finger orientation was built to correct the offset.

RidgePad to infer the roll, pitch and yaw of the finger. A calibration phase records different positions and assigns an offset to each one of them. During use, the system compares the received fingerprint to all fingerprints in the database to identify the user. *RidgePad* then selects the *k* fingerprints closest to the observed one and averages their offsets. The current touch location is computed as the center of the contact area, corrected by the averaged offsets.

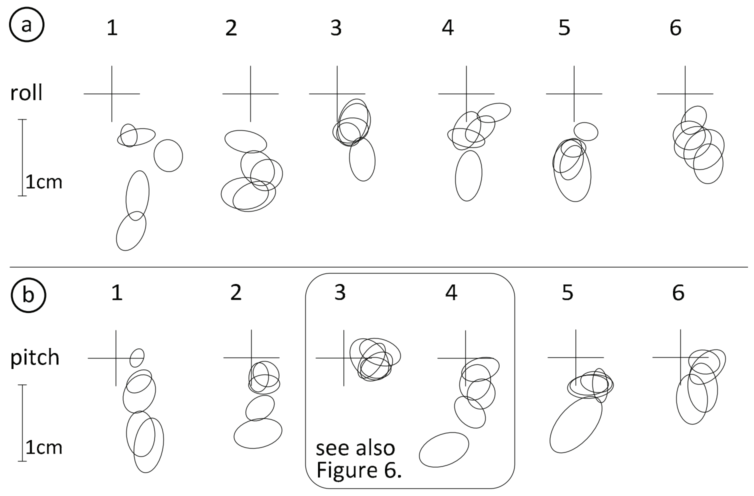


Figure 2.2: Clusters of touch locations for 6 participants. The ovals represent clusters of points for a particular level of roll (a) and pitch (b) [Holz and Baudisch, 2010].

The corrected touch locations were closer to the target than the uncorrected ones.

A second user study was performed to evaluate the performance of the *RidgePad*. Participants were asked to hit a target, a crosshair or a dot. The dot was occluded by the participant's finger and Holz wanted to check whether occlusion problem could be a factor in his model. Per user, the corrected touch locations were averaged and compared to the averaged uncorrected touch locations. *RidgePad* caused an average spread of 1.24mm while the uncorrected locations caused a 2.75mm spread. The spread of touch input is 2.2 times smaller when using the fingerprint touch device compared to a traditional touch input based on the center of the contact area.

There is support for the model, but with limitations in the user study.

Although these results tend to confirm the validity of the *generalized perceived input point* model, some limitations of the studies have to be taken into account. In the first study, the ovals representing the touch locations based on an angle contained only 65% of the touches. Also, people were asked to be as accurate as possible when using the system. It would be interesting to run a study where users could interact with the system as they would in a real-life situation. It is possible that the ovals may be bigger and not as clearly identifiable.

2.2 Targeting Aids

Review of targeting aids systems.

The following section presents an overview of different targeting aids systems specifically designed to address the target occlusion problem. Three techniques will be presented in this section: *Cross-keys*, *Precision-Handle* and *Shift*. Although several other systems have been designed over the years, we believed that those are the ones which had the most significant impact.

2.2.1 Cross-Keys

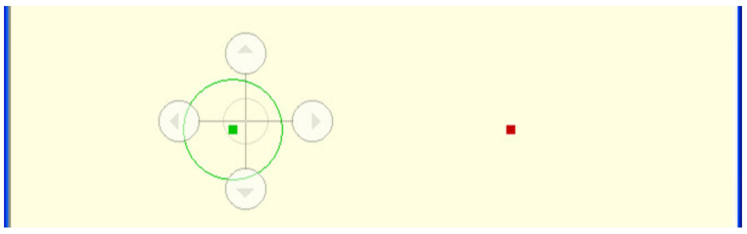


Figure 2.3: *Cross-Keys* (shown) can be used to move left and down to be placed accurately on the current target [Albinsson and Zhai, 2003].

Cross-Keys uses handles to move a crosshair.

Cross-Keys from Albinsson et al. [2003] removes the occlusion problem completely by moving the user's hand away from the target. This system uses a crosshair with control keys to move it and an activation key at the center of the crosshair to select the target.

Crosshair moves in discrete steps.

The first tap on the screen displays the crosshair on the screen. Taps on the handles of the crosshair allow it to move in discrete steps. When the target is reached, it can be selected by tapping the central circle. If the initial tap is too far away from the target, another tap will move it to a different location.

The results show that this selection method had a low error rate. This is due to the discrete movement which allows precise adjustments. On the other hand, the discrete taps increase the selection time. Another drawback is that the occlusion is not completely solved. The user's hand may hide the handles or the crosshair at times.

Crosshair has a low error rate, but it increases the selection time.

2.2.2 Precision-Handle

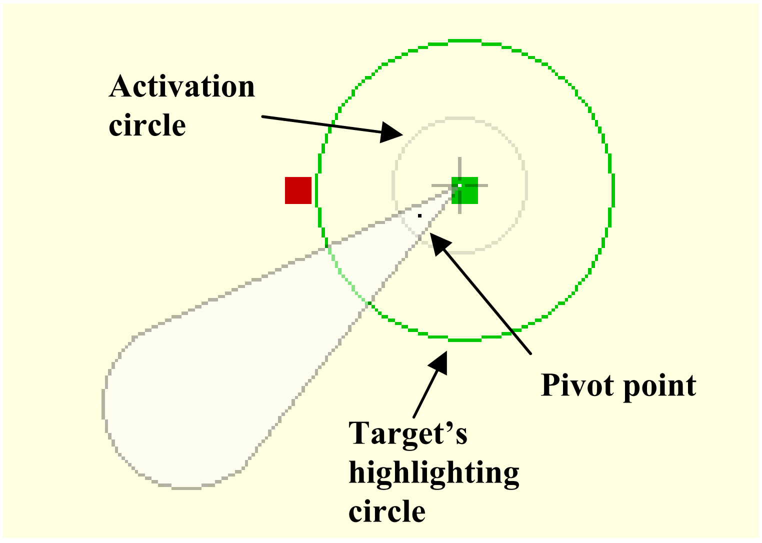


Figure 2.4: with *Precision-Handle*, all movements made at the handle are reproduced at the tip at a smaller scale, allowing for precise manipulation [Albinsson and Zhai, 2003].

Precision-Handle is another technique introduced by Albinsson et al. [2003]. As with *Cross-Keys*, the occlusion problem is removed as the user's interactions are not made on the target. This system consists of a handle and a crosshair at the tip to point at the target.

Precision-Handle removes the finger from the target.

Moving the finger on the handle moves the crosshair at the tip.

The handle appears on the screen when the user taps on it. The movements made at the handle are reproduced at the tip on a smaller scale, which increases the precision. When using the handle, it will shrink or stretch, thus linking its movements to the tip. The current crosshair position can be selected by tapping in the activation circle, as in the previous system.

Participants rated this technique positively.

The error rate for this technique was also low, and the users evaluated it positively regarding its speed, accuracy and comfort of use.

2.2.3 Shift

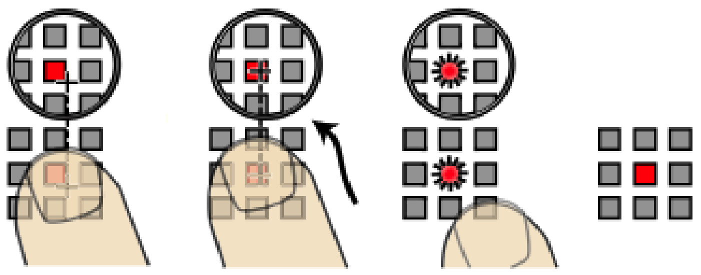


Figure 2.5: With *Shift*, ambiguous target selection is solved by displaying a callout showing the occluded area and the current finger selection point. Moving the finger allows the user to move the pointer on the target and lifting it selects the target [Vogel and Baudisch, 2007].

Offset Cursor moves the cursor above the finger, removing occlusion but also preventing to aim at targets directly.

Shift, developed by Vogel and al. [2007], builds on the idea of the *Offset Cursor* [Potter et al., 1988]. *Offset Cursor* creates a pointer on the screen at a fixed distance above the finger touch area. The finger can be dragged on the screen to move the pointer. The target is selected when the finger is lifted off the surface. Although this technique allows for precise selection, it has drawbacks. The most important one being that the user cannot aim at the target directly anymore. Instead, he has to counterbalance the offset by consciously aiming below the target. As there is no visual representation on the screen until the finger is touching it, estimating the offset is hard and needs more iterations.

Shift aims at keeping the benefits from *Offset Cursor* while fixing its drawbacks. To achieve this, it does not only offsets the pointer, but also the part of the screen that is under the finger. Figure 2.5 shows a scenario when the target selection is ambiguous (i.e. more than one target around the pointer) and *Shift* displays the callout. When there is no ambiguity, the target is selected and no callout is displayed. Once the callout containing a view of the occluded area is shown, the user can pinpoint the target by sliding his finger on the surface. The target is selected when the user lifts his finger.

Shift creates a callout above the finger to display the occluded area and the cursor.

Unlike *Offset Cursor*, *Shift* keeps the speed and simplicity of direct touch interaction. It makes the system easy to use in a walk-up situation. *Shift* solves the occlusion problem but does not improve the accuracy issue. When the user lifts his finger, the pointer moves as the contact area changes and it may become off target.

Shift solves the occlusion problem, but does not increase touch input accuracy.

2.3 Finger and Body Posture Use

The finger orientation, the position of the arm, head and other body parts have been used to enhance the interaction with multitouch surfaces. All of them however, focus on creating new interaction techniques and not on improving the accuracy of such surfaces.

Review of systems using finger orientation or body posture.

2.3.1 Finger Input Properties

Wang and Ren [2009] investigated the different properties of a finger and designed four widgets that made use of the touch area shape, its size and the finger orientation.

Four widgets using the finger orientation.

Their first widget is the *Finger Combination Cursor*. This technique relies on the contact area to define an area cursor as well as the contact point to define the point cursor. While most systems use the contact point to select a target, Wang and Ren make use of these two parameters. When a touch is detected, the system follows this algorithm to se-

Finger Combination Cursor uses contact area and contact point to select targets.

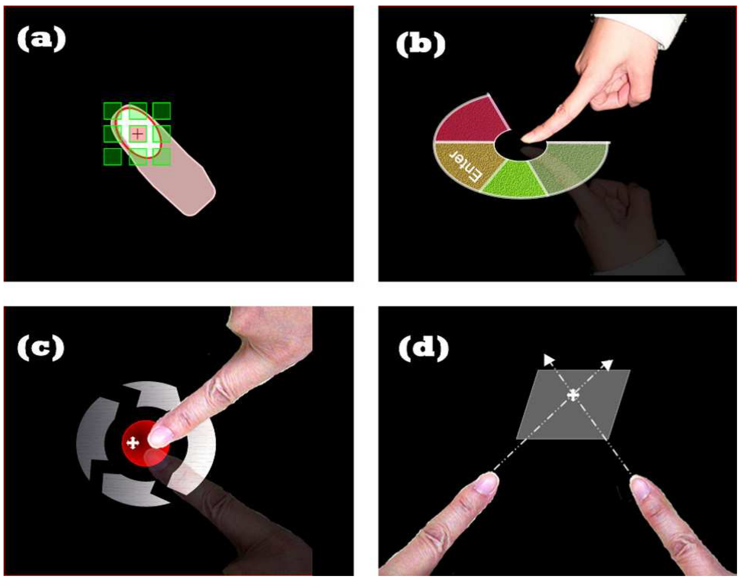


Figure 2.6: Widgets using the finger orientation. (a) Finger Combination Cursor, (b) Finger Sector Menu, (c) Finger Pointing Stick, (d) Finger Cross Selection [Wang and Ren, 2009].

lect a target. If only one target is inside the contact area, it is selected using the area cursor; if more than one target is covered by the contact area, the point cursor is used and the target closest to it is selected. This technique allows fast target selection with relatively low accuracy when using the area cursor, while at the same time retaining the fine precision of the point cursor.

Finger Sector Menu displays pie menu around the finger.

Finger Sector Menu comes from the observation that in multitouch systems, pie menus are often launched by a finger touch and that some items are always occluded by the hand. The touch area shape is used to determine the finger orientation (more details in the following section). Once this is known, the position of the hand can be deduced and the items displayed on a pie menu around the hand, but not under it. Items can be selected by rocking the finger or rotating it. This widget solves the occluded item issue and makes item selection more natural, as no item is ill placed regarding the wrist position.

Finger Pointing Stick is based on the pointing stick, the isometric joystick used as a pointing device one can find on laptop mice and keyboards. The finger works like a joystick; the cursor can be moved using the changes in the contact area, rotating the finger fine-tune the cursor and rocking the finger moves the cursor in the same direction.

Finger Pointing Stick uses the finger as a joystick.

Finger Cross Selection aims at making the selection of distant targets easier and faster by eliminating the need for the user to reach all the way to this target. When using two fingers on the surface, radial lines show the orientation of the finger and the intersection of these lines is used to select the target.

Finger Cross Selection enables selecting distant fingers by using two fingers.

2.3.2 Pointing Gesture Recognition

Nickel and Stiefelhagen [2003] investigated body posture in pointing task. They tested three approaches to establish the pointing target. The first one was the line of sight between the head and the hand, the second was the forearm orientation and the last one was the head orientation. The line of sight between head and hand, as well as the forearm orientation were determined using stereo camera, while the head orientation was given by a magnetic sensor.

Use of body postures for pointing at objects in space.

They noted that participants tended to look at pointing targets while gesturing toward them. Comparing the three approaches, their results show that the head-hand-line produced the most accurate estimate for pointing direction. This proves the usefulness of using other body parts and not only the hand in pointing tasks.

The line between the head and the hand is the best estimate for pointing directions.

2.4 Finger Orientation Detection

2.4.1 Computer Vision

Malik et al. [2004] developed the *Visual Touchpad*, a stereo hand tracking system that provides the 3D positions of the user's fingertips as well as their orientation on and above

Using stereo cameras to track hands in space.

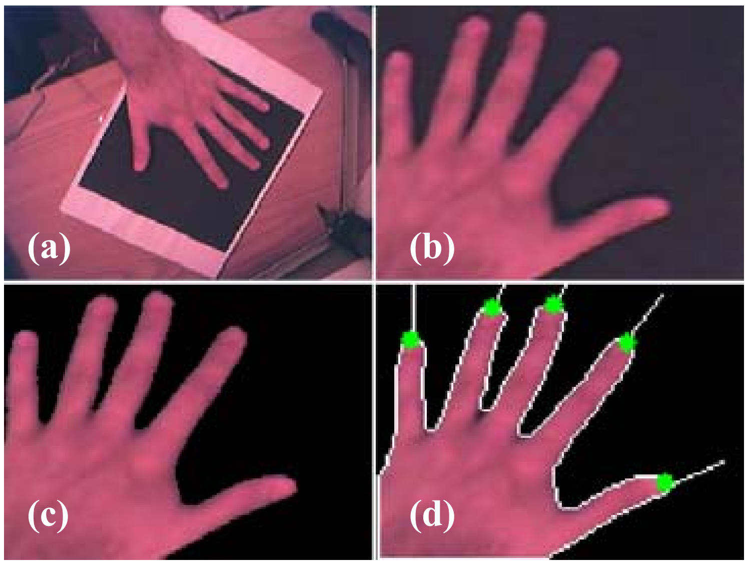


Figure 2.7: Hand detection steps. (a) Original image, (b) Warped Image, (c) Background subtracted image, (d) Fingertips position and orientation detected [Malik and Laszlo, 2004].

a surface. The system is composed of two video cameras mounted above a touchpad, a regular piece of paper.

Extracting finger orientation from the camera images.

The first step of their fingertip and orientation detection is to map the camera image to the screen coordinates and create a warped image of the touchpad on the screen, as can be seen in figure 2.7a and figure 2.7b. The background is then subtracted from the image. The high contrast between the black of the touchpad and the hands allows the system to robustly deals with shadows and different lighting conditions. The hands are detected using a flood-fill function. The contours of the hand are then detected, and the fingertips are found by looking at strong peaks along the borders. The vectors from a contour point *k* to *k*+*n* and *k*-*n* are computed for a fixed *n*, and if the angle between those vectors is below a certain threshold, that point is marked as a fingertip. The orientation is determined by computing the median line from the fingertip *k* and the points *k*+*n* and *k*-*n*. Figure 2.7d shows the resulting fingertips and finger orientation.

2.4.2 Contact Area Shape

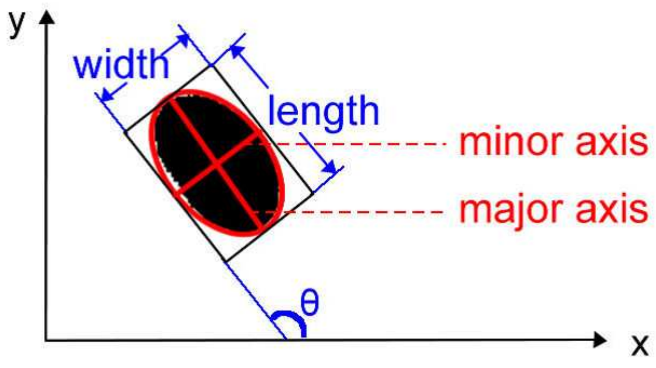


Figure 2.8: Contact area shape fitted to an ellipse. The major axis shows the undirected finger orientation [Wang and Ren, 2009].

Wang et al. [2009] designed a new approach for detecting unambiguously finger orientation using the contact area shape of the finger on a surface. Their system relies on a tabletop based on Frustrated Total Internal Reflection (FTIR). The contact area of the finger is fitted to an ellipse, and if the ratio between the main axis and the minor axis of this ellipse is above a certain threshold, the main axis gives the undirected finger orientation.

Using the contact area shape for detecting finger orientation.

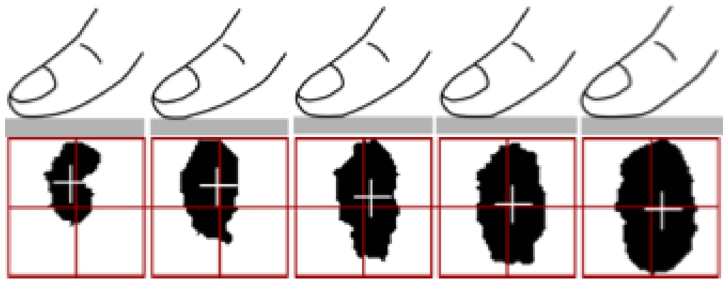


Figure 2.9: Contact area shape deformation when a finger touches a surface [Wang et al., 2009].

The disambiguation of the finger direction is done by looking at the way the finger lands on the surface. As the finger has a soft texture, it does not come in contact with the sur-

The contact area shape changes as the finger lands on the surface and gives the finger orientation.

face in one step. As we can see in figure 2.9, the contact area shape changes while the finger gets on the surface and the center of this area moves towards the palm of the user. By analyzing the movements of the center of the contact area, Wang et al. are able to determine the direction of the finger orientation. The results show that disambiguation algorithm has a success rate of 96.7%, while the angle has an average detection error of 2.69°.

2.4.3 Fiduciary-Tagged Gloves

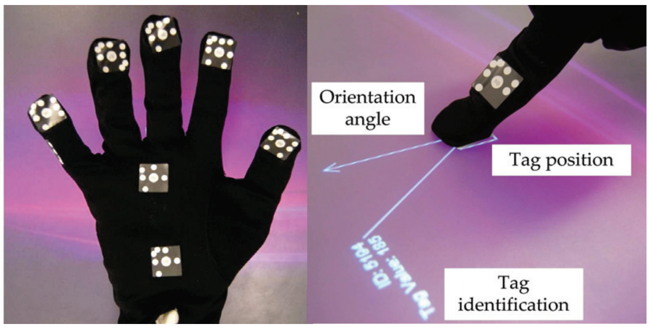


Figure 2.10: Fiduciary-Tagged Gloves (bottom view and touching a tabletop) [Marquardt et al., 2010].

Using fiduciary tags on fingertips to get the finger orientation.

Marquardt et al. [2010] introduced the *Fiduciary-Tagged Gloves* as a reliable and expressive way to collect information regarding several parts of the hand. We are especially interested in the ability of the system to detect the finger orientation. Glove-based tracking is well used in domains such as virtual reality and augmented reality, while it is not very common with tabletops. Marquardt et al. use a Microsoft Surface table - with Diffuse Illumination - and a glove tagged with multiple fiduciary markers. The table is able to detect several markers at the same time and differentiate between them. The finger orientation is given by the orientation of the tag on the table. The advantages of this system are the relative low cost of its implementation, its ease of use and the possibility of detecting and differentiating several parts of the hand. All of the above allow for rapid prototyping, but cannot make for an end-user system, as wearing gloves is not a natural way to interact with tabletops.

2.4.4 Hand Shape

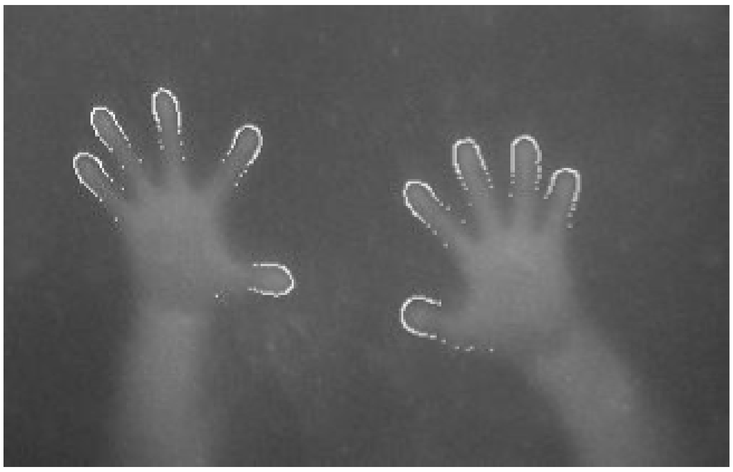


Figure 2.11: Finger Contours determine the symmetrical lines around each finger and use them to compute the finger orientation [Dang and André, 2011].

Dang et al. [2011] employ a multitouch table working with Diffuse Illumination to obtain the shape of the hand as shown in figure 2.11. The idea behind Dang's work is that each finger touch can be represented by two quasi symmetrical lines converging at the fingertip and that those lines can be used to determine the finger orientation. The position of the finger is obtained by removing the background from the raw image to get the bright spots representing the fingers. The center of each contact area determines the finger position. Lines are drawn from the center point and the first dark point found along the line belongs to the contour and is stored in a list. Once all the contour points have been found, the points at the fingertip are excluded and the remainder points are used to compute the two quasi symmetrical lines. The angles between those lines and the x-axis are computed and averaged to determine the finger orientation. The authors compare their method to the ellipse method with 180-adjust and show that the contour method is more accurate and gives a recognition rate above 93% with an error range of 10°.

The hand shape is used to detect fingertips and finger orientations.

2.4.5 Capacitive Sensors

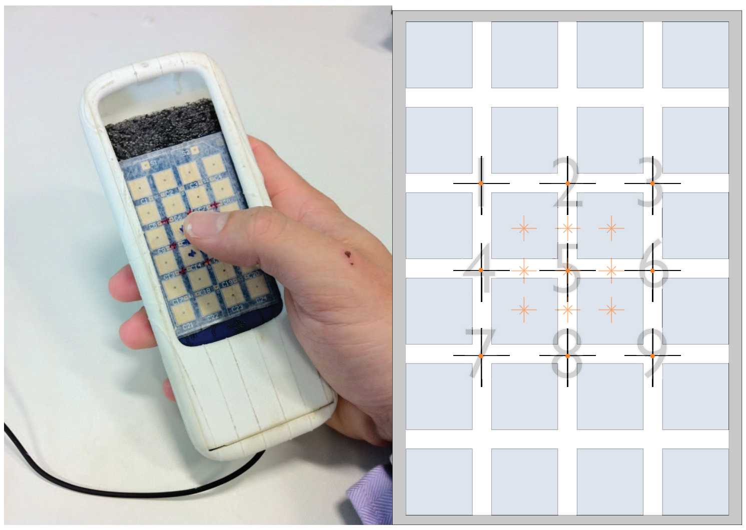


Figure 2.12: Capacitive proximity sensors are used to determine the finger position and also its pitch and yaw [Rogers et al., 2011].

Capacitive sensors are used to detect the finger pitch and yaw.

Rogers et al. [2011] aim at improving the touch accuracy by using the finger pitch and yaw in addition to the finger position, as Holz et al. [2010] showed that varying these parameters led to different touch location for the same target (see 2.1). An array of capacitive proximity sensors is used to feed a model which generates the finger position, the contact area size, and the finger pitch and yaw. The model uses a particle filter to infer the finger position based on the data provided by the sensors. Their results confirm the observations made by Holz et al. [2010]. The position given by the model is more accurate than a simple interpolation. They suggest that they can achieve a 95% accuracy for buttons with a 5mm radius.

Chapter 3

Study Design and Hypotheses Formulation

As stated in the introduction (see Section 1.2), the aim of this thesis is to expand on Holz' findings [2010]. In other words, we want to investigate if different body postures, and not only the finger position, can create different offsets when selecting a target on a tabletop. This section presents the iterated study design process and the refinement of our set of hypotheses. The study designs and hypotheses were tested in several experiments.

For testing our hypotheses, we created several studies iteratively.

3.1 Paper Prototype

We designed our paper prototype to check whether our hypotheses were sensible. We ran a short experiment with it to see whether the task we asked the users to perform and the procedure we followed were adequate to test our hypotheses. Our main hypothesis was that different body postures would create different touch offsets. Other research questions were investigated in this experiment:

We ran a study with a paper prototype to test if our task and procedure were adapted.

- for a given hand position, does the distance between the target and the body have an impact on accuracy?

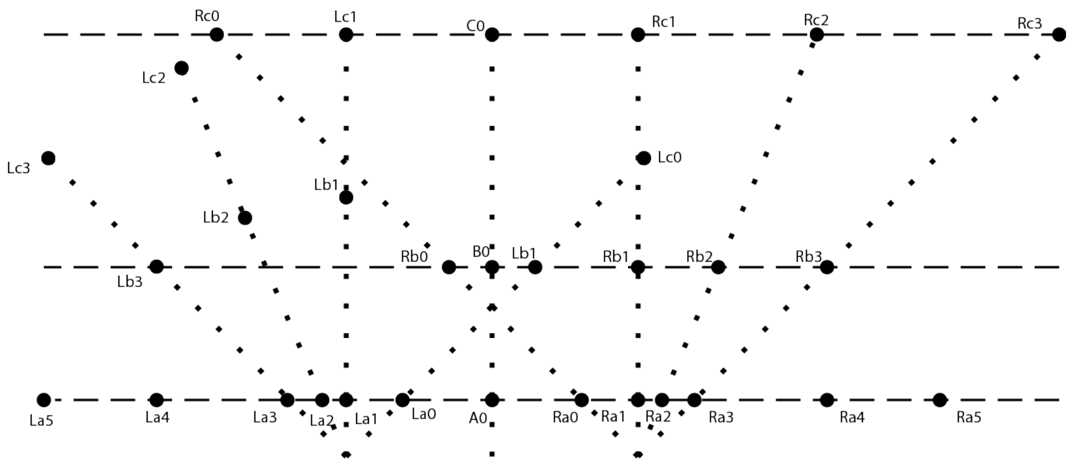


Figure 3.1: Paper Prototype: target layout.

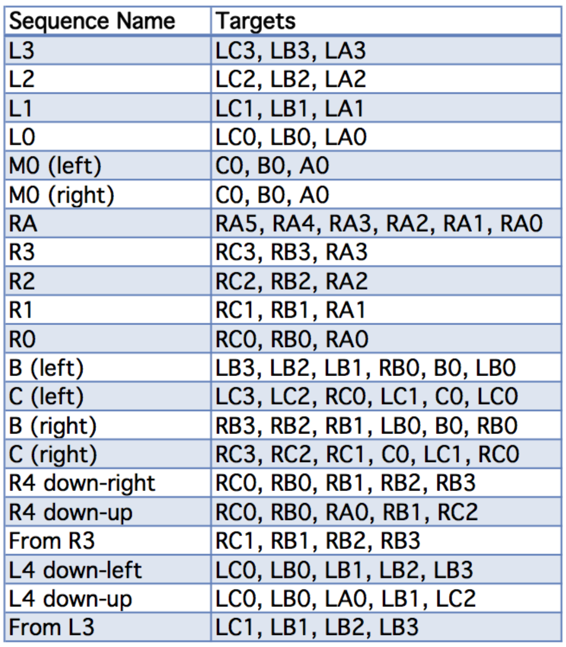
- when moving from one target to the next, will the finger stay in the same position while the arm moves or will it move as well?
- when the next target is occluded, will participants change their posture for the current target to see all of the targets?
- can we compare the dominant and non-dominant hand? Is there a mirroring effect or less accuracy when using the non-dominant hand?

Targets and lines were drawn on a table.

In order to verify our hypotheses, we designed an experiment in which participants had to touch targets on a table in a specific order, while being seated in front of the table. The setup was made of a sheet of paper covering a table. Targets were drawn onto the sheet and were connected by several lines (see Figure 3.1). The horizontal span of the targets was 155cm and the vertical one was 60cm. A video camera placed above the table filmed the participants as they did the experiment.

Participants had to touch targets following sequences.

For this study, participants were asked to touch particular targets in a row using either their left of right index finger. The sequence of targets was announced by the experimenter, along with which finger they had to use. The touches of the sequences were announced in advance by the experimenter. The sequences used can be seen in figure 3.2. There were 21 sequences, and each of them was repeated 5 times. The order of the sequences was randomized. Once every sequence had been used once, they would be repeated using a different order.



Sequence Name	Targets
L3	LC3, LB3, LA3
L2	LC2, LB2, LA2
L1	LC1, LB1, LA1
L0	LC0, LB0, LA0
M0 (left)	C0, B0, A0
M0 (right)	C0, B0, A0
RA	RA5, RA4, RA3, RA2, RA1, RA0
R3	RC3, RB3, RA3
R2	RC2, RB2, RA2
R1	RC1, RB1, RA1
R0	RC0, RB0, RA0
B (left)	LB3, LB2, LB1, RB0, B0, LB0
C (left)	LC3, LC2, RC0, LC1, C0, LC0
B (right)	RB3, RB2, RB1, LB0, B0, RB0
C (right)	RC3, RC2, RC1, C0, LC1, RC0
R4 down-right	RC0, RB0, RB1, RB2, RB3
R4 down-up	RC0, RB0, RA0, RB1, RC2
From R3	RC1, RB1, RB2, RB3
L4 down-left	LC0, LB0, LB1, LB2, LB3
L4 down-up	LC0, LB0, LA0, LB1, LC2
From L3	LC1, LB1, LB2, LB3

Figure 3.2: Paper Prototype: sequences of targets used. The first letter of the sequence name refers to the performing hand.

Sequences were chosen to see particular body postures.

We decided to use sequences with the belief that they would influence which joints were moved when the participant's hand went from one target to the next. We did not want to explicitly ask participants to adopt specific limb postures as it could have influenced the results.

The setup was captured by a video camera.

The camera mounted above the table recorded the experiment for all of the users. The video files for each participant were processed and snapshots of the touches have been extracted. These pictures were then labeled with the name of the hit target, the name of the previous target and the following one. For instance, the snapshot of the target *LB2* in the sequence *L2* would be labeled *LB2.LC2.LA2*.

Participants were young students.

This study was ran with four participants (one female), all studying Computer Science. All participants were right-

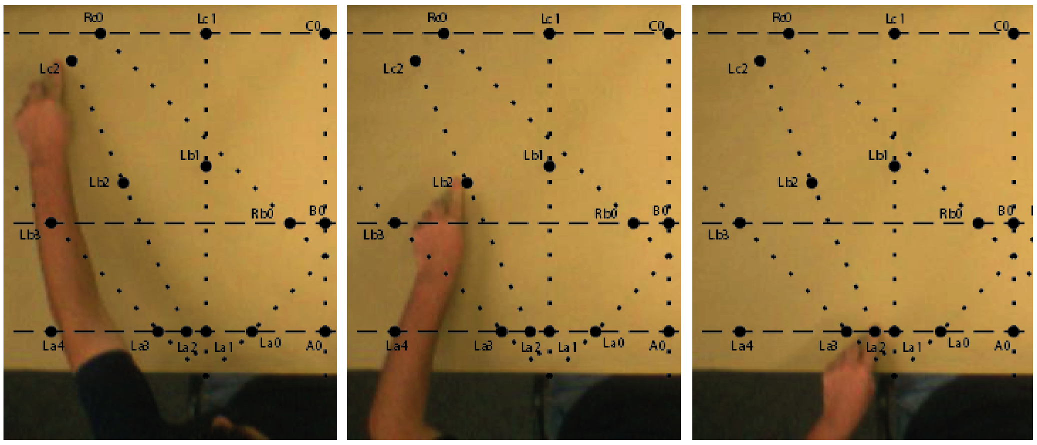


Figure 3.3: Paper Prototype: participant touching targets in a row. Different joint positions can be observed.

handed and between 22 and 25 years old. Drinks and snacks were provided during the study and no other incentive was offered.

Some targets were ill positioned.

When analyzing the videos and interviewing the participants after the study, it was clear that some targets were not comfortable to reach for multiple reasons. Some of them were too far from the body (*e.g. RC0, RC3*), while some were too close to the chest or too close from one another (*e.g. LA1, LA2, LA3*).

Different joints are moved from one position to the next.

Figure 3.3 shows the different body postures adopted by a participant when performing the touch sequence *LC2 - LB2 - LA2*. We can observe that the finger and hand orientation do not change between *LC2* and *LB2* but that the movement is made by the wrist, elbow and shoulder, while to reach *LA2* all joints are used.

The body posture depends on the position of the target on the table.

When looking at the snapshots, different postures were seen depending on the position of the target on the table. When comparing body posture for mirrored points, it seems that the gesture is mirrored and not completely different. It was also mentioned during an interview that announcing the whole sequence in advance may have an influence on the accuracy. This participant said that when touching a target, he was already thinking about the next one.

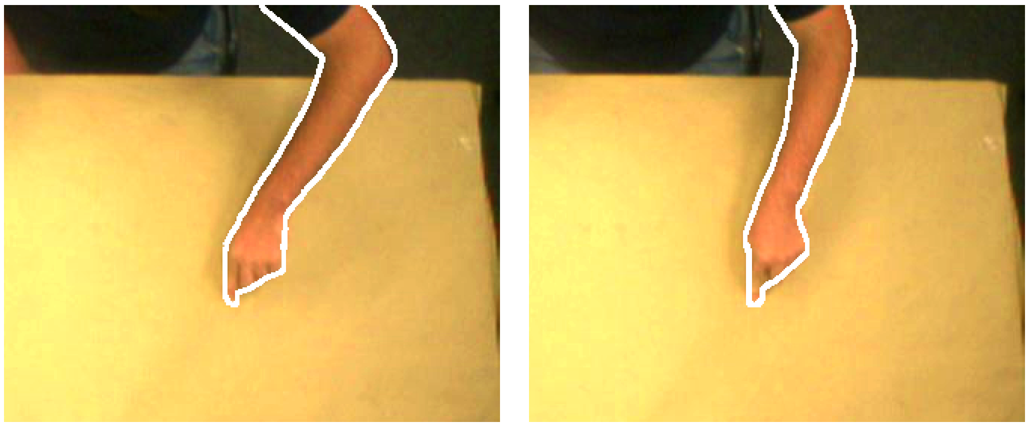


Figure 3.4: different body posture for the same target in different touch sequences.

Another finding regarding the way people dealt with occlusion was made. Some sequences were made in such a way that the following target would be hidden by the participant's arm when he reached for the first one. Some participants reached for the first target using the most natural and direct gesture, thus occluding the next target, while others chose not to occlude them and created an arc with their arm to go around the targets. For this experiment, the targets had a label next to them so that the participant would know which one to tap on. It is possible that the participants' movements were influenced by them.

People deal with occlusions in different ways.

The most interesting result was that the body posture when touching a particular point was different depending on the touch sequence it was part of. For instance, when touching *LB1* the elbow of the participant in figure 3.4 was at more acute angle in the sequence *LC1, LB1, LA1* than in the sequence *LA0, LB1, LC2*.

Different body postures are observed for the same target, when it is part of different sequences.

These findings lead us to formulate a new set of hypotheses:

- For a given position on the table, different predecessor and successor points will create different postures and offsets. In the rest of this thesis, this will be referred as the predecessor and successor effect.
- Different arm postures for the same finger and hand posture will create different offsets

- Different angles between the gaze and the finger direction will create different offsets (i.e. finger and gaze direction crossing at different angles)
- If the target becomes hidden by their arm, participants will change their posture
- Offsets for mirrored posture are similar
- Same touch sequences made on different parts of the table will produce different offsets.

3.2 Next Design

We split the experiment in two parts: the sequence effect and the position of the target on the table.

These hypotheses caused a number of changes in the design and implementation of the experiment. As we have seen, the body posture may be influenced by the position of the target on the table, but also by the predecessor and successor touch (or sequence effect). These two hypotheses cannot be tested at the same time, thus the experiment was split in two parts. The first part of the study would focus on the sequence effect while the second one would verify the other hypotheses.

3.2.1 Touch Sequence Effect

We placed the targets on a fixed circular layout.

In order to check the predecessor an successor effect without being influenced by the position of the target on the table, we designed an eight pointed star (see Figure 3.5). Each point of the star and its center was a target in the experiment. The distances between the points of the star and the center were the same, thus eliminating an independent variable. The distance between the outer points and the center was 20 cm. All the points were in a 40 cm diameter area in front of the user, which made them easily reachable without stretching.

Three points patterns were used, all going through the center, and on this point, the predecessor and successor effect can be investigated. 24 patterns were selected out of the 64

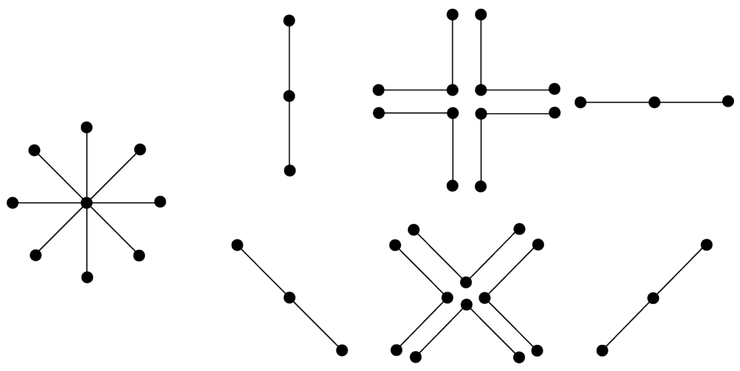


Figure 3.5: Position of the points and the sequences used. Each sequence was done in both directions.

patterns possible. This set is large enough to see different patterns, while at the same time small enough to analyze the results. We planned to repeat this pattern at different places on the table to see if it had an influence on the touch sequence effect.

We used three-points patterns, all going through the same middle point.

3.2.2 Body Posture and Position on Table

The second part of the study was to investigate the effect of the body posture on the touch offset without the possible interference of the predecessor and successor effect. To this end, we decided to use a grid of evenly distributed points on the table. The points were all within arm length without stretching. To remove any touch sequence effect, the participant's hand would move back to a resting position between touches. This way, all touches would have the same predecessor and no successor. The order of the points would be randomized using a latin square design. All of the points would be used the same number of times, and sequences would be counterbalanced.

Test the effect of the position of the target on the table with a grid of evenly distributed points.

During the rest of this thesis, we focus on the successor and predecessor effect on the touch offset. The body posture effect on different positions of the table remains to be investigated.

We focused on the predecessor and successor effects.

3.3 First Prototype

All the targets of the sequence were visible from the start.

The first software prototype was created to test the sequence effect hypothesis. A multitouch table was used to record the finger position and its orientation for each participants, and to display the targets. The patterns described in section 3.2.1 were used. The application displayed three crosshairs representing the points of a pattern. The crosshair for the current target was larger than the other ones so that the participants' finger did not completely occluded it when selected. In addition to the bigger crosshair, the current target had a red circle to be clearly identifiable. The targets did not have any label next to them not to influence the users. The application displaying the stimuli was later used to develop the Touch Sequence Agent for the final study. Section 4.3 details the implementation of the Touch Sequence Agent. This first prototype worked following the same principles.

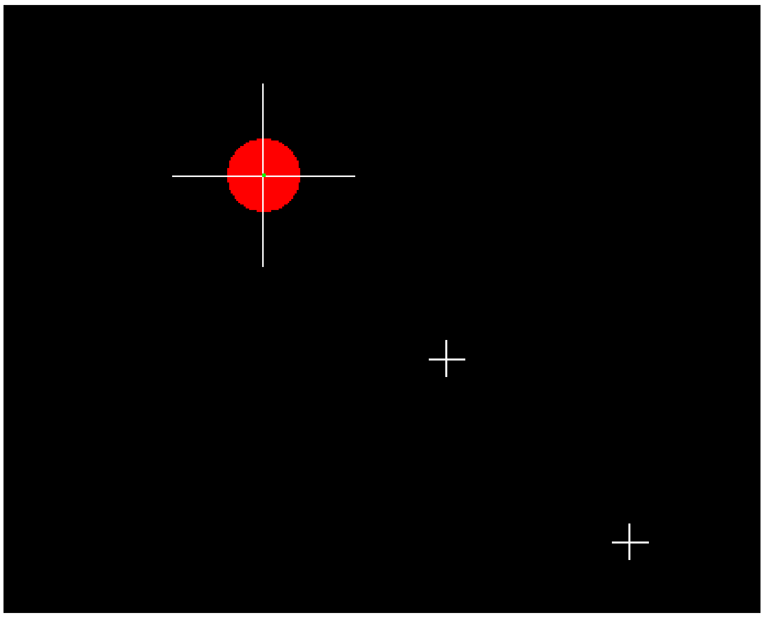


Figure 3.6: A three-points pattern used in the first prototype. The red circle designates the current target.

Used DI and a single IR camera.

The technology used by the multitouch table to capture the position and the orientation of the finger of the participants was Diffuse Illumination (DI). A single infrared camera looked at the whole surface from below and the information about the finger was extracted from these images. The touch detection was performed using the original MultiScreen agent presented in section 4.1.

Participants were standing in front of the table.

Compared to the study with the paper prototype where participants were seated, in this one, the participants were standing in front of the table, which is a more common position when using a tabletop. Participants were asked to touch the targets one after the other. Once the touch had been registered, the next target would become active and display the red circle.

The finger orientation and position were recorded for the middle touches.

The patterns were displayed in a random order and each one was repeated three times. For the central touch of each pattern, the finger position and orientation were recorded along with its predecessor and successor. This information was exported to an external log file for analysis.

Five Participants took part in this study.

The study was conducted with five participants (one female) who did not take part in the paper prototype study. They were also all Computer Science students, right-handed and between 24 and 28 years old. Drinks and snacks were provided during the study and no other incentive was offered.

The resolution of the system was too low to detect the finger orientation.

When we started analyzing the raw data, it became clear that we could not extract a lot of information from it for two reasons. With only one infrared camera looking at the whole surface, the quality of the touch input was good enough to detect the touch in the target area, but not sufficient to correctly detect the shape of the finger and its orientation. Furthermore, all users except one used a high pitched finger when touching the table, so only the fingertip was in contact. The recorded touches were very round, and it was not possible to extract the finger orientation.

For one participant, a clear distinction was seen between the touches oriented to the left from the ones oriented to the right.

One participant used the pad of his finger to touch the targets, thus the contact shape was a better ellipsoid and the finger orientation could be seen (see Figure 3.7). It was possible to see a clear distinction between the touches oriented to the left from the ones oriented to the right. Although, due to the low resolution of the images, the precise location of the touch input could not be determined and we could not infer more results.

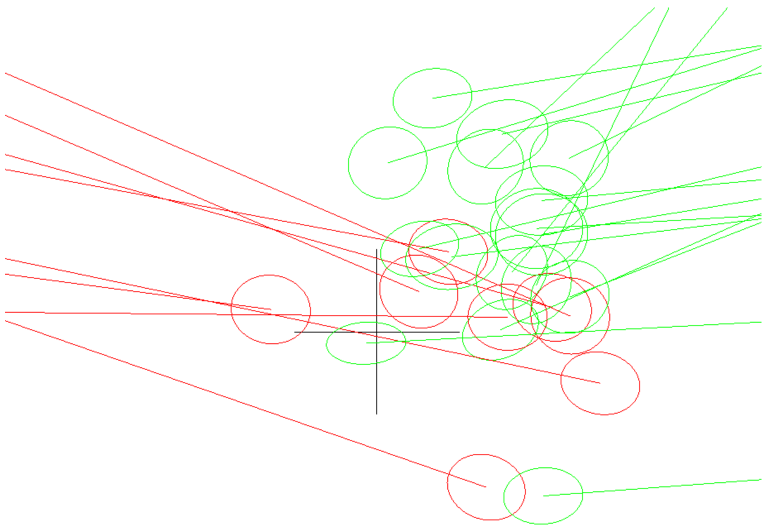


Figure 3.7: Touch areas with finger orientation for a participant.

The resolution of the touch detection system needed to be improved.

From this study, we gained valuable information for the study design. The Diffuse Illumination is a technology that can be easily set up, but the resolution of the captured image needs to be increased to be useful. Also, the minimum size of the contact area has to be increased. This will constrain the participants to use their finger pad, and the contact shape will be more ellipsoidal and the finger orientation will be visible.

3.4 Pre-study on Handedness

This study was conducted to compare the behavior of left-handed people and right-handed people.

Following the study with the first prototype, another study was conducted to compare left-handed and right-handed people. The aim was to see whether any major differences could be seen in the touch imprints. The same patterns, software and study protocol as in the previous study was used. Compared to the previous study, the touches were not validated as soon as the participant touched the surface, but after a small delay. This delay was to ensure that the finger was in its final position and that the touch would not be recorded only with a small imprint. Like in the previous study, the patterns were randomized and each one was repeated three times.

Used FTIR and two IR cameras.

We used a different multitouch table which was based on Frustrated Total Internal Reflection (FTIR) and had two infrared cameras positioned under the surface to capture the touch inputs. The switch from DI to FTIR was made to remove the background noise from the camera image caused by the hovering hand and to get sharper touch area.

Same task as in the previous study, with the addition of a wack-a-mole game.

The task the participants had to perform was the same as in the previous study. In addition, to familiarize them with the system, a "wack-a-mole" like game was implemented; targets appeared randomly on the table and the participants had to hit them before they disappeared. Users played with it until they were confident in using the system.

We instructed the participants to use the pad of their finger.

It was stressed to the participants that neither speed nor accuracy was a factor in the experiment and that they should use the system as they would any touchscreen application. Participants were asked to use the pad of their finger when touching the surface. This was to increase the size of the imprint and the chances of getting a good ellipsoid from which to infer the finger orientation. As we were looking for touch sequence effects, participants were asked to keep their finger on the surface until the touch was registered and then moved directly to the next target without going through a rest position.

14 participants took part in this study.

The study was conducted with 14 participants (4 females), out of which 4 were left-handed (all male). The participants' handedness was determined using the Edinburgh Handedness Inventory [Oldfield, 1971]. All participants were studying Computer Science and between 21 and 28 years old. Drinks and snacks were provided during the study and no other incentive was offered.

Several observations were made during and after the study. For most users, the delay to register touches was confusing. They expected the touch to be registered instantly, as it would be on any other touchscreen device. Because of this behavior, participants would touch a target and move their finger towards the next one before realizing that the touch had not been registered and move back to tap it again. When this happened for the middle touch, the sequence was not as planned; the movement between the predecessor and the middle touch was not direct anymore and an extra variable was introduced. Also, after the paper prototype study, we decided to remove the labels of the targets, as they may influence the participants. The drawback was that they did not know which one of the two targets would be the next, and the successor was not clear in their mind.

The touch registration delay was too long and confusing.

Although the participants were asked to use the pad of their finger, the touch imprints were not bigger. Additionally, the ellipses were very round; for each user the roundness – the ratio of the minor axis divided by the main axis – ranged between 0.83 and 0.93. For a particular user, the average roundness was 0.88 with values ranging from .57 to 1, while the averaged main axis of the imprint was 27 pixels and ranging between 8 and 40. Because of the roundness and size of the imprints, for a given predecessor and successor, the angle of the finger was very different from one trial to the next, and it was not possible to see any consistency in the raw data.

The touch imprints were round, we could not detect the finger orientation.

We tried to clean the data by removing the outliers points and see whether the remainder would make sense. The criteria to exclude points were the roundness and the size of the imprints. The orientation cannot be reliably estimated if the touch was either too round of too small. After that step, as we had only three repetitions for each pattern, there was not enough data left to work with.

We removed outliers, but then we did not have enough data left.

Following this, changes were made for the final study design; the number of repetitions for each pattern was increased to five, so even if bad data had to be removed, analysis would still be possible. The touch registering delay was removed and replaced by a minimum size for the imprint, to make the use of the system more fluid and provide better data. Labels were reintroduced, but instead of being next to the targets, they were directly on them, not to change the focus of the participants.

We changed the study design to obtain better data.

3.5 Final Study

For the final study, we worked with the Department of Work and Cognitive Psychology on the design of the experiment. We decided to change the targets from a crosshair to a round button so that the shape would not influence the participants, and to make our system look more like a real application. A number was added to each button to remove any confusion in the sequence. The current button was shown in green, while the others were grayed to stress that they were inactive.

We replaced crosshairs by numbered round buttons.

We thought about the fact that participants do not have the same height, and looked into the benefits of making them stand on sized boxed so that the angle of their finger when touching the surface would be the same. We ruled this idea out, because as the length of the arm is related to a person's height, asking smaller people to stand on a stool would force them to lean on the table. However, to see whether the results would be influenced by the physiognomy of the users, we decided to measure the participants' height, arm length and how far they could reach on the table with a straight arm and a straight back. Participants were instructed to stand in front of the table and without leaning, to push a small piece of wood until their arm was straight.

We measured the participants height, arm length and reach, as they could have an effect.

As in the previous studies, the buttons were displayed in a circular layout (see Figure 3.8). The buttons were slightly smaller than a fingerprint (15 mm diameter) and the circle had a diameter of 40 cm. The distances between the points were chosen so that the middle one would be in the usual work area, which is the most easily and comfortably place reached on a table, within forearm reach. The northernmost point should be reachable with a minimal amount of stretching or leaning. During the test, buttons were touched in sequences of two or three. Instead of using an eight-pointed star and a subset of patterns, we reduced the number of points to six and decided to test all possible combinations.

Buttons were on a circular layout and were all comfortably reachable.

Each sequence of touches was repeated five times, and each of them was made under four conditions. These conditions were introduced to differentiate the successor and prede-

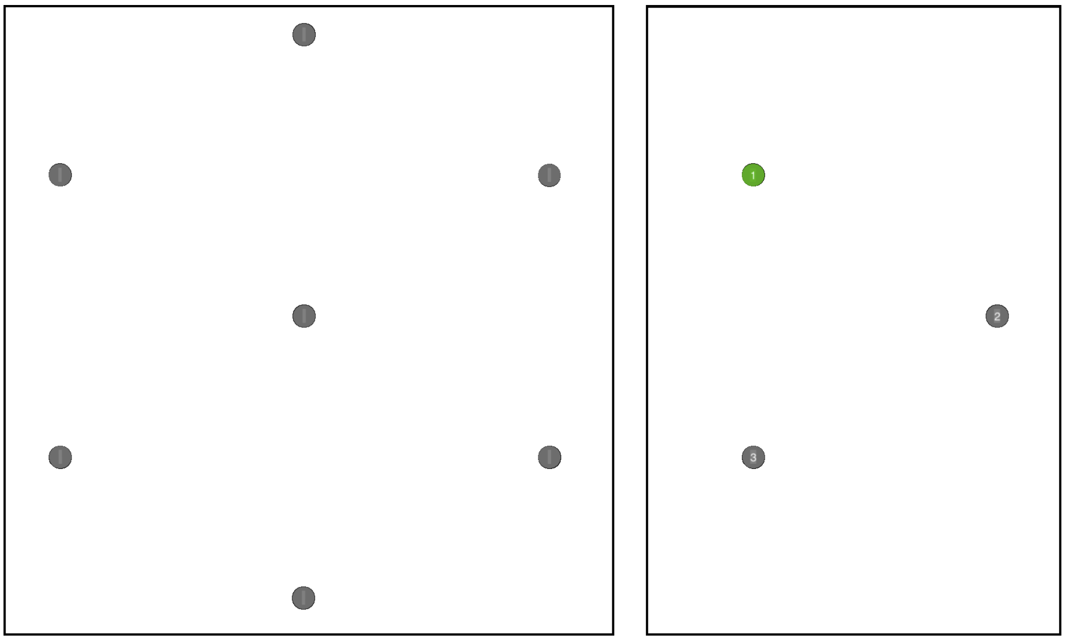


Figure 3.8: [Left] Layout of the buttons. Sequences were made of two or three of them, one to three of them were visible at the same time. [Right] One of the sequences used.

cessor effect.

We used different conditions to differentiate the predecessor from the successor effect.

1 the first, middle and last buttons are displayed simultaneously, thus the finger orientation and the touch location on the middle button is influenced by both its predecessor and successor.

2 the first and middle buttons are displayed at the beginning, the third one appears after the middle one has been touched. The finger orientation and the touch location on the middle button is influenced by its predecessor only.

3 the middle and the last buttons are shown at the same time, there is no predecessor for the middle button, thus only a successor effect should be observed on the finger orientation and touch location on that button.

4 the middle button is shown, the last one is displayed after the middle one has been touched. Neither predecessor nor successor effect should be seen on the middle button.

Participants were told that neither speed nor accuracy was a factor and that they should interact with the system as they would with any other touchscreen device. In order to have a common starting position for the sequences, participants were asked to hover above the table with their hand. This behavior is close to real life usage and the initial posture of the arm should be comparable for all trials. Before each sequences, an auditory stimulus was played to prepare the subject. To avoid rhythm-like response, the waiting time between sequences was randomized and ranged between one and three seconds. A limitation was imposed to the participants as we asked them to use the pad of their finger when touching the surface and not only their fingertip.

No speed or accuracy measurements, we asked the participants to use the system casually.

Before the test, a calibration phase was done. Participants were asked to tap the middle button ten times. This way, we could see the average spread of touches for each users. Following this, the sequences were displayed. For the first four participants, each condition was done once, then they were repeated four more times. The order of the conditions was determined using a balanced Latin Square. Subsequently, upon advice from the psychology chair, all the trials for a particular condition were made in a row, and a small break was introduced after each condition. Within each iteration of a condition, the order of the sequences were determined using a balanced Latin Square.

Each condition was repeated 5 times, and the order of the conditions was randomized.

The number of sequences in each condition is not the same, as depending on the condition, sequences of two or three buttons were used. In the first two conditions, there was 36 possible combinations, while in the last two, only 6 combinations. The only way to balance the number of trials in all conditions would have been to repeat all sequences six times in condition one and two, and to repeat them 36 times in condition three and four. In the current design, with five repetitions for each sequence, participants have to tap 420 sequences ([36 + 36 + 6 + 6] x 5). Balancing the number of trials would have brought this number up to 864 sequences (36 x 6 x 4). By doing this, the duration of the experiment would have been much over an hour, while it was already 40 minutes.

The number of trials in each condition was not balanced.

The study was conducted with 14 participants (2 females), 14 participants took part in this study.

all right-handed. The participants' handedness was determined using the Edinburgh Handedness Inventory [Oldfield, 1971]. Eleven participants were studying Computer Science, one Biology, one Electrical Engineering and one Mathematics. They were between 20 and 31 years old. Drinks and snacks were provided during the study and a raffle for a 20 euros voucher at Amazon was made.

Next chapter presents the implementation.

In the following chapters, we will present the software and hardware implementation that were done to realize this study. We will then discuss the results we obtained after running said study and analyzing the data.

Chapter 4

Implementation

In order to test the hypothesis that the predecessor and successor touches influence the orientation of the finger on the surface and that different touch offsets can be seen, a fast and reliable method was required to capture the finger position and its orientation on the table.

Needed reliable method to detect the finger orientation.

The touch position can be easily obtained on a multitouch table using FTIR or D.I technology. However, the tables with those technologies at our disposal were not accurate enough to extract the finger orientation as well. The input resolution of these tables was between 18 and 20 ppi. Several options were considered to get that information. Optical tracking, like the Vicon[1] system, offers one of the most accurate methods to track limbs real-time in space. There are multiple disadvantages in using such a system. The Vicon system uses Infrared cameras to track markers on users. The markers are not inconspicuous and can influence the way people move. Furthermore, a recent study showed that users tended to aim at targets using visual features located on top of their finger [Holz and Baudisch, 2011]. One or several markers on the participants' fingertip could impact their mental model and the findings would be biased. Lastly, the IR light emitted by the camera could overflow the multitouch table and interfere with the touch tracking. A Kinect would solve the marker issue, but its infrared laser projector could still hinder the multitouch table

Optical tracking systems are not accurate enough or have the inconvenient of requiring markers.

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

system. Also, the Kinect is not accurate enough to be used as a primary source for the finger orientation.

Tagged gloves are too conspicuous.

Fiduciary gloves, such as the ones presented by Marquardt [2010] were considered, as a tag on the finger pad could be easily seen by the multitouch table and the orientation of the finger could be inferred from it. This option was quickly discarded, as the influence on the participants' state of mind would have been ever greater than with markers.

We decided to use a IR based table, but with a greater accuracy.

The solution chosen was to use the multitouch table, with FTIR or DI, but with an increased accuracy. As we have learned from the pre-study, touch imprints made by the finger pad produce an ellipse. If this ellipse is big and clear enough, the finger orientation can be extracted from it. Originally, the multitouch table we used had two Infrared cameras with large scope below it to look at the whole surface.

Get high accuracy by using one camera per target.

As we had seven targets on a circle of 40cm diameter, one camera would not provide the required level of accuracy. Using one camera with a narrow focus per target would enable the cameras to provide us with high resolution images. However, the amount of data to be transferred from the cameras to the computer to be analyzed in real-time to find touches and the finger orientations would have been too much. The workload could have been divided if the touch recognition algorithm did not use the input from the seven cameras, but only the one looking at the current button or the two or three cameras involved in a sequence.

Use only one camera, looking at different position on the table using a mirror.

As seven cameras would have been hard to manage, we decided to use only one camera with a narrow focus to look at the targets in turn. Our first idea was that the camera would be attached below the table and by pointing it at a rotating mirror (see Figure 4.1), we could have high resolution images of different parts of the table.

Use one moving camera looking directly at the table.

The fixed camera and the moving mirror were discarded on account that the distance between the camera, the mirror and the table would be too great and the resolution not high enough. Building on this idea, we decided to use one moving camera below the table, pointed directly at the surface. Using a narrow focus scope, the camera could see a 10

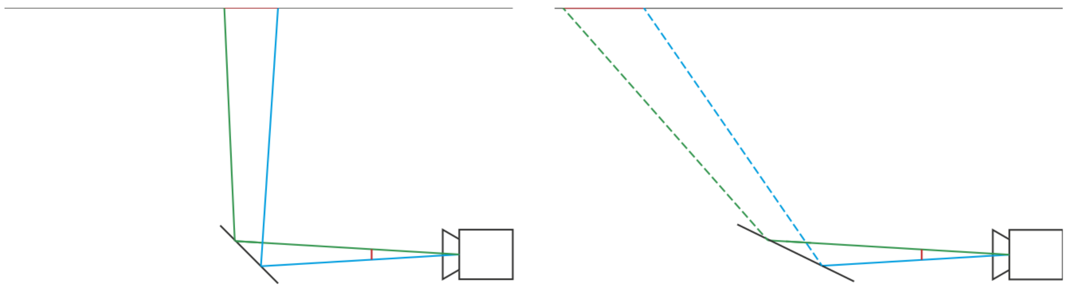


Figure 4.1: A fixed camera below the table would be pointed at a moving mirror. By rotating the mirror, different parts of the table could be seen with high resolution.

x 10 cm square on the table. For each target on the surface, there was a different camera position. The resolution of this camera was of 160ppi, eight times more accurate than the original systems we used with our first prototype and the handedness study.

The next sections present our applications.

In the following sections, we will present the MultiScreen agent used to capture the touches on the surface, including their position and orientation, the Camera Control Agent used to move the camera to the predetermined positions, the Touch Sequence Display used to present the stimuli. Finally, we will show how these three components communicate with one another.

4.1 MultiScreen Agent

Our touch detection system is based on the MultiScreen Agent.

The touch detection software used in this thesis is based on the existing MultiScreen Agent, which we have modified to fit our needs. The MultiScreen Agent is a framework developed in Objective-C at our chair by Malte Weiss and Simon Voelker. It was created for the BendDesk system [2010] and is now used in all our multitouch surface related projects. The MultiScreen Agent is a touch detection system which can work with any number of cameras and video projectors. The touch detection can be performed using either FTIR, DI or both. The agent enables us to calibrate the screens used to display the interface on the surface and the cameras used to detect the touches. When the agent is running, touch events are distributed through the NSNotificationCenter to other applications. The display application needs two frameworks, the MultiScreenRenderer and the MultiTouchServer. The former enables us to draw on the surface using OpenGL, while the latter processes the notifications sent by the MultiScreen agent when touches are performed.

We will present the MultiScreen Agent.

We will present the main components of the agent, how it is configured and used, and which modifications we have implemented to fit our requirements.

4.1.1 Main Components

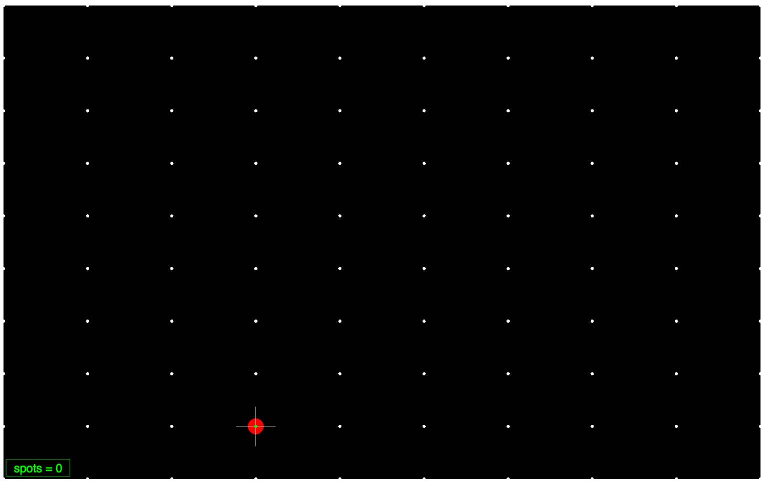


Figure 4.2: Grid used for the camera calibration. The user must touch the red dot. Depending on whether it is seen by a camera or not, the mapping for each camera can be computed.

Four main objects.

The MultiScreen Agent is composed of four main objects; a list of screens, a list of cameras, and for each camera, a spot detection object and a mapping object.

The screens and cameras can be calibrated.

The list of screens contains the projectors used to display our application. Each one can be calibrated so that the display could fit on curved surfaces if needed. The cameras are infrared ones below the table used to capture touches on the surface. Each one has its own parameters, such as video mode, frame rate, brightness, etc.

Each camera has a Spot Detection object, which contains a set of values defining a touch like the brightness of the pixels or the size of the spots. It can also contain a background image. This image is taken when no touches occur on the table, and at runtime, the brightness value of each pixel from the background image is subtracted from the incoming image. This step helps in removing artifacts and noise from the input images.

There is one Spot Detection object per camera defining touches.

The cameras have their own mapping, which convert the coordinates of the touches from the camera to global coordinates on the display.

The touches coordinates are mapped to the coordinates of the table.

Before runtime, the cameras must be calibrated with the surface. A grid of points is displayed on the surface, and by touching the active position, the mapping for each camera can be realized. After this step, each camera is linked to a part of the surface.

The cameras must be calibrated to realize the mapping.

At runtime, all cameras are used at the same time. For each camera, the image is processed by the Spot Detection Algorithm. Pixels within range of the brightness levels are grouped into regions, and if these regions are within the spot size levels, a Principal Component Analysis (PCA) is performed to generate the ellipse fitting of the spot. The spots' coordinates are then mapped to the display and touch notifications are broadcasted.

The Spot Detection algorithm processes the images and broadcast touch notifications.

4.1.2 Modifications

Single Camera With Multiple Positions

The main modification to make on the MultiScreen agent was to replace the multiple physical cameras used by one camera with multiple positions. In the original architecture, each camera object used is tied to a physical one, and each camera object has its own parameters, spot detection object and mapping. A partial and simplified object relationship diagram illustrates the architecture (see Figure 4.3).

We needed to replace the multiple physical cameras by one with multiple positions.

The first attempt to have multiple camera objects while us-

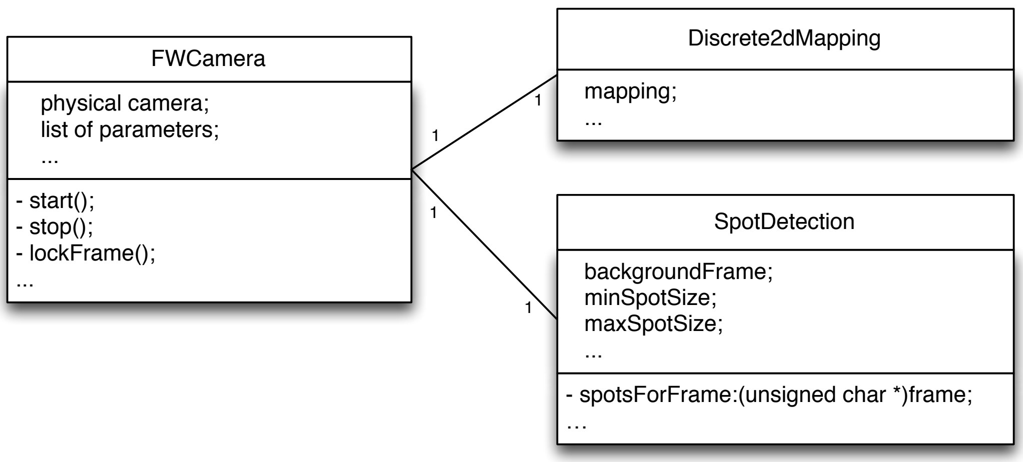


Figure 4.3: Simplified object relationship diagram of the original MultiScreen Agent. Each camera object references a physical one and has its own mapping and spot detection object.

We could not create multiple camera objects with a pointer to the same physical camera.

ing one physical camera was made with this architecture. We wanted to create several camera objects, all with their own spot detection and mapping objects. This attempt was unsuccessful for a simple reason. Each camera object had a pointer towards the physical camera. When the pointer was created, the API used to interface with the firewire camera would give control of the physical camera to the first object, and the other ones could not access it.

We changed the architecture and added a quadrant object between the camera and the Spot Detection objects.

The solution was to change the architecture. The camera object would still be linked to the physical camera, but it would be a unique object, hence removing all pointer related issues with the hardware. With only one camera object however, the spot detection and mappings objects could not be directly linked to it anymore. A quadrant object was created to fill in the gap between them. Quadrants would represent the different positions of the physical camera. In the new architecture, the unique camera object would have multiple quadrants, and each one of them would have its own spot detection and mapping objects, as can be seen in figure 4.4.

In the implementation, the relationship between the camera, spot detection and mapping objects is not based on ob-

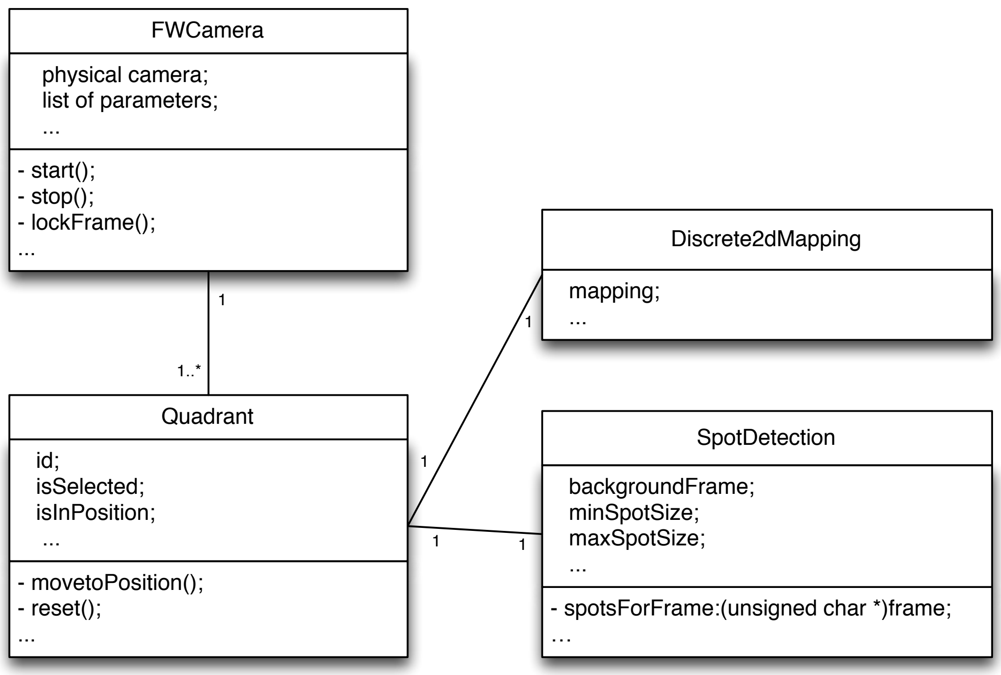


Figure 4.4: Simplified object relationship diagram of the modified MultiScreen Agent. Each camera object references a physical one and a quadrant. Each quadrant has its own mapping and spot detection object.

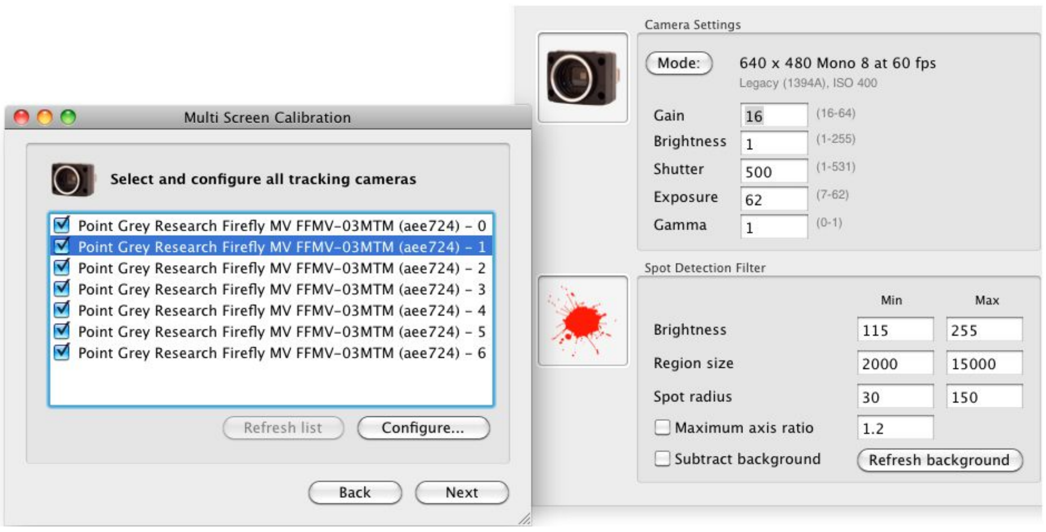
ject composition. The main object, MultiScreenCalibration, contained a list of camera objects, one of spot detection objects and another one of mapping objects. As the objects were independent from one another, their indexes in the lists were used to link them together.

The objects were linked together using indexes, not through object composition.

With the addition of the quadrant object and the replacement of the camera list by a single camera object, the code was inspected and every instance where the camera list was used was modified to fit the new architecture. Although the code was altered in numerous places, the modifications related to three activities; the setup and calibration, the run-time, and the communication with the Camera Control agent.

We modified the MultiScreen agent code to fit the new architecture.

In the setup phase, the code for cameras selection needed to be updated, so instead of displaying a list of available cameras, it listed the quadrants. After modification, it was possible to configure each quadrant, in the same way the cam-


Multi Screen Calibration

Select and configure all tracking cameras

- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 0
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 1
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 2
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 3
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 4
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 5
- ☑ Point Grey Research Firefly MV FFMV-03MTM (aee724) – 6

Camera Settings

Mode: 640 x 480 Mono 8 at 60 fps
Legacy (1394A), ISO 400

Gain	16	(16-64)
Brightness	1	(1-255)
Shutter	500	(1-531)
Exposure	62	(7-62)
Gamma	1	(0-1)

Spot Detection Filter

	Min	Max
Brightness	115	255
Region size	2000	15000
Spot radius	30	150
☐ Maximum axis ratio	1.2	
☐ Subtract background		

Figure 4.5: Parts of the quadrants selection and configuration screens. The camera settings are common for all quadrants while Spot Detection parameters are specific to each quadrant.

eras were in the original agent. The camera settings were common to all quadrants, while the spot detection parameters were specific for each of them (see Figure 4.5). The next stage in the setup phase was the calibration step. In the original agent, this step computed a mapping between the screen coordinates and the camera image coordinates for each camera; a grid of points was projected on the surface, and each one of them was touched in turn. Depending on whether the touch was seen by one of the cameras, a calibration grid was created for each camera at the same time. In our version of the agent, the grid for each quadrant needed to be done separately, as the camera had be moved to a different position. When the calibration grids were created for all of the quadrant, the mappings were then computed. At the end of the setup phase, every quadrant had its own spot detection object with its specific settings, and its own mapping.

The calibration had to be modified to use the new quadrants objects.

In our user study, all of the quadrant were configured for each users. As each person has different finger shape and size, the parameters for the spots detection were tweaked to personally fit each and every participants.

During our study, the calibration was modified for each participant.

The next part of the agent that needed to be changed was the run function. In the original design, when the MultiScreen agent was running, in the tracking function, all the cameras were capturing at the same time. Spots were searched for in each frame, mapped to screen coordinates and if the same spot was seen by multiple cameras, they were merged before touches events were generated by the notification center. In our version of the agent, only one quadrant could be active at a time. The tracking function needed to know which quadrant to use and which mapping to apply. To this end, quadrant objects had two boolean attributes, `isSelected` and `isInPosition`. The tracking function would select the quadrant for which the `isSelected` attribute was true, and when the `isInPosition` attribute was also true, the spot detection was done for this quadrant, using its specific parameters and mapping.

At runtime, only one quadrant was active and the correct mapping had to be used.

The last challenge to overcome in adapting the MultiScreen agent for quadrants was to find a way to update the quadrant objects at runtime. The agent already had an `NSNotificationCenter` implemented. We decided to use this method to control the tracking function. When the camera needed to be moved to a new position, the `isSelected` and `isInPosition` of all quadrants would be reset to false, while the `isSelected` of the new quadrant would be set to true. When the camera was in position, the `isInPosition` attribute would be updated and the tracking would resume. Details about the communication between the MultiScreen agent, the Camera Control agent and the Touch Sequence display will be discussed in section 4.4.

The tracking was paused when the camera was moving.

The 180° Adjust

In the original agent, the spot detection algorithm gave the center of a touch, along with the main and minor axis of the fitting ellipse. In order to detect the finger orientation, this algorithm was not sufficient. The orientation could be wrongly skewed by 180°, as the main axis for the ellipse could be oriented in the wrong direction. The algorithm needed a few extra steps to ensure that the main axis was oriented in the correct way and that it could be used to de-

The finger orientation could be skewed by 180°.

termine the finger orientation.

The orientation was computed using PCA.

The original algorithm performed a Principal Component Analysis (PCA) on the images once the spots were detected. The PCA was used to compute the mean position of the pixels in the spot and the eigenvectors of the covariance matrix. With those, the oriented bounding box was calculated. Once those steps were done by the PCA, the main and minor axis of the ellipse were computed.

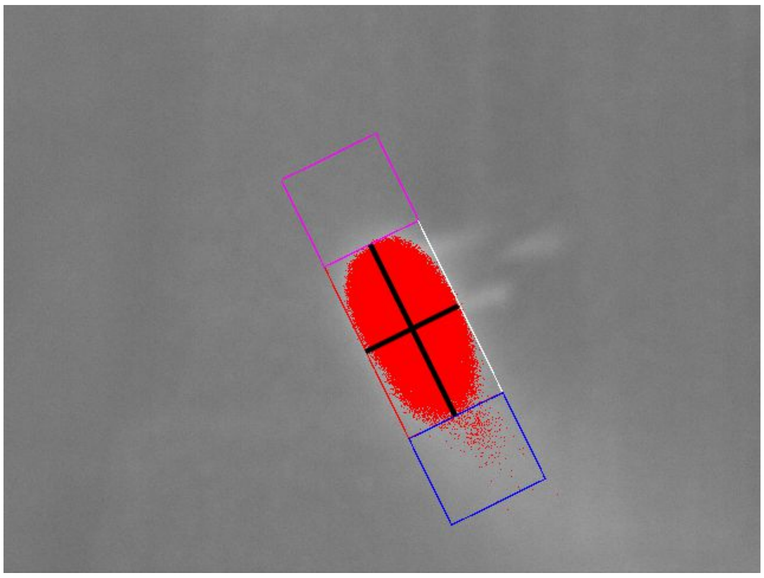


Figure 4.6: Spot as seen by the camera. The oriented bounding box, main and minor axis for the touch can be seen, along with the two boxes above and below the touch used for the 180° adjust.

Using DI, the hovering finger can be seen.

As we stated above, the direction of the main axis we obtained could be wrong. In order to make sure that it was correctly oriented, we decided to use Diffuse Illumination and to look at the shadow of the finger on the table. As we can see in figure 4.6, the part of the finger touching the surface is the brightest and will be used by the spot detection algorithm. We can also see the shadow of the finger hovering above the table.

The shadow of the finger was used to identify its direction.

We used the oriented bounding box computed by the PCA and created a box above and below the first one. The intensity of the pixels in said boxes were summed up, then

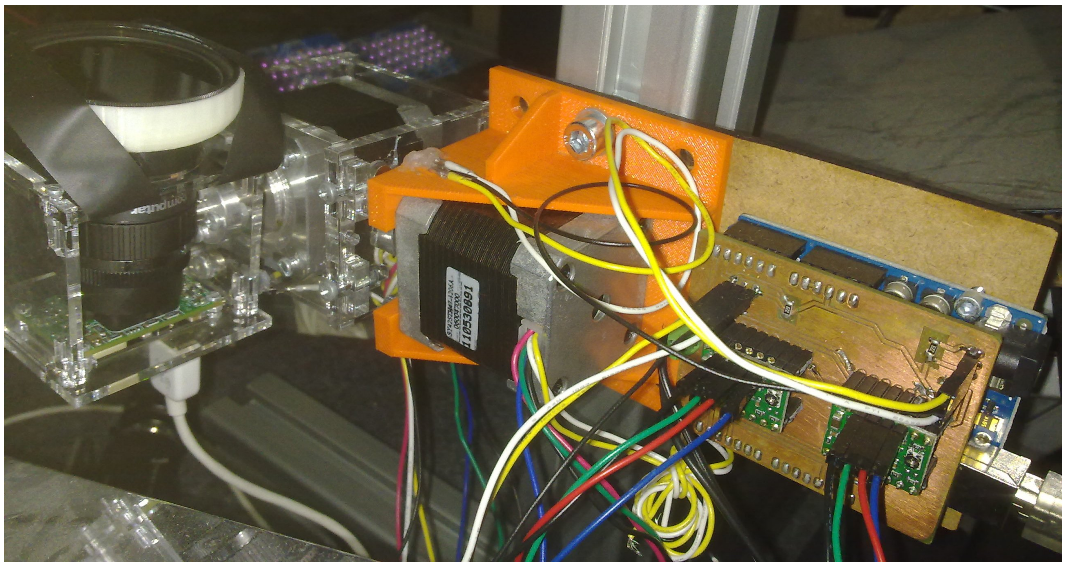


Figure 4.7: Camera attached to two stepper motors and controlled by an Arduino microcontroller. The communication with the computer is done through USB.

compared. The one with the highest value would be where the finger was, while the other one would be in front of the finger. If the main axis is oriented towards the upper box, then its direction is correct. Otherwise, the orientation of the main axis is reversed.

4.2 Camera Control Agent

Once the MultiScreen agent was ready to work with one camera and multiple positions, we needed a system to move the camera from one position to the next. As we stated at the beginning of this chapter, for each target on the surface, we needed a different camera position. This way, the camera was looking at a 10 x 10 cm square with the target in its center. The camera needed to move fast enough so that in the final study, when sequences were displayed, the camera could move between targets faster than the participant's finger. The second requirement was that the camera needed to move to the different positions as accurately as possible. As the calibration for each quadrant was done for particular camera position, if the camera was not ex-

We needed a fast and accurate system to move the camera.

actly pointed the same way, the touch information from the MultiScreen agent would be wrong; the position would be displaced and the angle would also be affected. The last requirement was that a two-way communication was needed between the moving camera and the computer.

we will present the hardware, firmware and software.

In the remainder of this section, we will present the hardware chosen to build the moving camera system, the firmware controlling it and the software enabling the communication between the applications on the computer and the hardware.

4.2.1 Hardware Selection and Assembly

Stepper motors were moved precisely and they held in position.

In order to fulfill our requirements, we decided to use stepper motors to move the camera. This kind of motors move in discrete steps, and thus, they can be used to move the camera precisely to chosen positions. Another advantage of these motors resides in the fact that they use electromagnets to hold their shaft in place. As long as they are powered on, they will stay in the same position; the load weighting on the shaft will not make them turn. We chose motors with enough torque (3.17 kg-cm) to hold another motor and a camera.

The revolution of the motor was made in 1600 steps.

The stepper motors each required a driver carrier, which was connected between the motor and the microcontroller. The microcontroller we chose (Pololu A4988) also provides microstepping for the motors. In normal use, our stepper motors would do a revolution in 200 steps. With these drivers, the revolution was made in 1600 microsteps, which allowed for even more precise positioning of the camera.

The camera could be pointed at any part of the table.

As we can see in the sketch shown in figure 4.8, with the camera attached to the first motor, and this motor being itself attached to the second one, the camera could be moved along two axis. Once placed under the multitouch table, the camera could be pointed at any part of the table.

The initial positions of the motors were the zero position.

The main drawback with stepper motors is that when the power is cut off, they will not retain their positions. The weight attached to their shafts will not be held back any

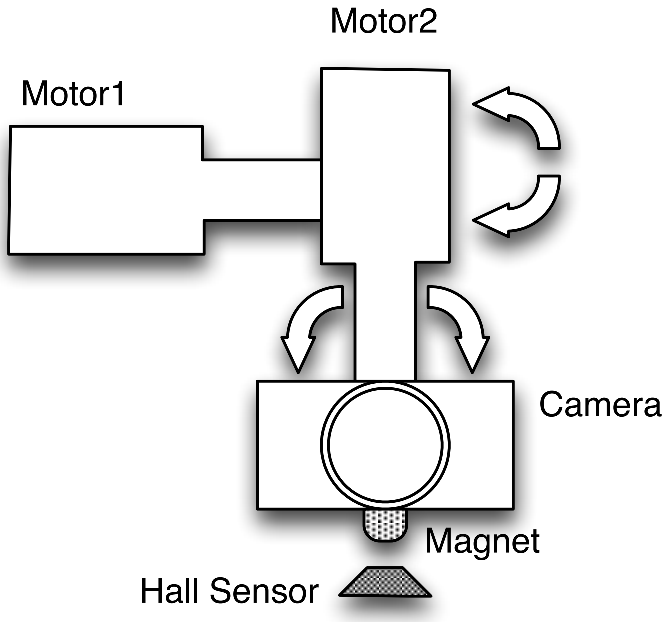


Figure 4.8: In this early working setup, the camera is attached to a first motor, which is itself attached to a second one.

more and they will turn. Also, there is no zero position on the motors. When they are powered on, their initial position is the starting position. From there, they turn the number of steps they were asked to.

In our application, to reach one of the predetermined camera positions, each motor had to turn a certain number of steps in a particular direction. The starting position had to be fixed for the camera to be positioned correctly. Otherwise each time the power would be switched off and on again, the starting position would be different, and the camera positions reached for each quadrant would not always be the same.

We needed a fixed starting position.

The solution we applied to have a fixed starting position and to avoid recalibrating the MultiScreen agent every time the power was switched off was to add a hall sensor to the setup. Hall sensors are transducers that change their out-

We wanted to use a single hall sensor to reset the starting position.

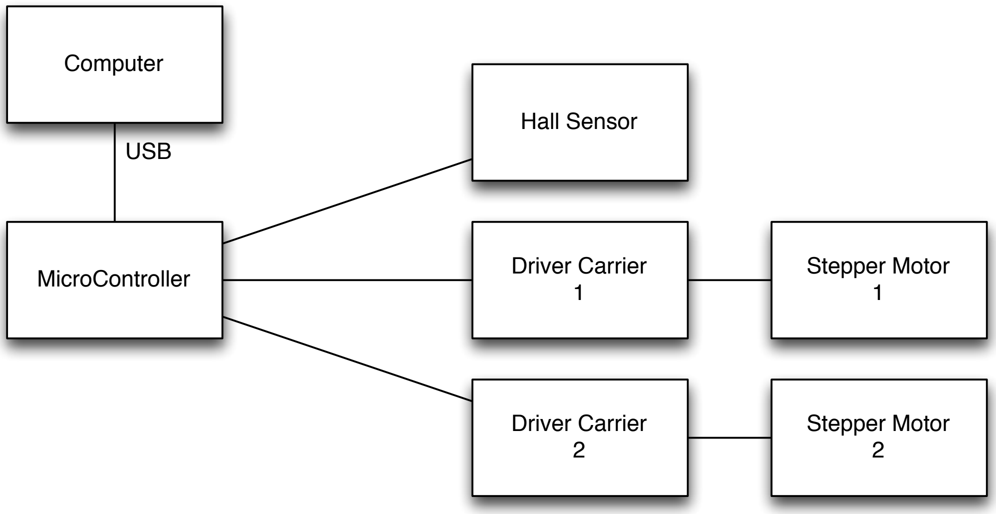


Figure 4.9: Simplified communication between the Camera Control main components. The detailed schematic and board layout an be found in Appendix A.

put voltage depending on the magnetic field they sense. In our design, we used a binary hall sensor; when a magnet was close to the sensor, its output would be turned on, and when the magnet was moved out of range of the sensor, the output would be off. In our first hardware setup, we planned to use this single hall sensor to move the camera and the two motors to a starting position.

Arduino was the best choice for the microcontroller.

The microcontroller used to control the motors and receive the output from the hall sensor was the Arduino Uno, which was chosen for several reasons. Arduino is an open-source prototyping platform, which allows for fast and easy development. The stepper motors and the hall sensor could be connected to it without the need of extra libraries, and the Arduino board offers communication with applications running on computers using USB connection. The microcontroller can be easily programmed through the Arduino IDE in C. Also, the amount of tutorials and examples available hasten the development phase of the firmware.

We needed our own PCB.

Once the parts were selected, work started on the printed circuit board (PCB) on which all the electronic components would be attached. Figure 4.9 shows a basic communication between the main components of the board. The PCB was designed using the Eagle software[2]. We used this software for its ease of use, its power, the availability of a freeware version and the amount of tutorial available.

We created the schematics for the PCB.

The first step in designing the PCB was to create the schematic. This document holds all of the electronic components needed, as well as the electrical parts such as resistors and power supplies. When all components were added to the schematic, we wired them together. The datasheets for the electronics were invaluable for connecting them to the microcontroller.

We made the physical layout.

The next step in creating the PCB was arranging the components and wiring them to the physical layout. In Eagle, when the schematic is complete, which means when all components have been added and the wiring is correct, the components can be imported onto the PCB layout. They can be then moved around to form the wanted disposition. In our case, we placed the components in such a way that they would fit on a board, which could be directly plugged in the Arduino microcontroller. The final step was to create the conductive pathways, or routes, between the components on the PCB. Eagle has a very powerful tool, the auto-router, which as its name says, can create the routes automatically and save the user a lot of time.

The PCB was milled at the FabLab.

When the PCB files were ready, we used the milling machine at our FabLab[3] to produce the board. We operated the milling machine following the instructions contained in the tutorial[4] written at our chair. We finished the board by soldering all the components onto it.

At the same time, we built holders for the motors and the camera, so that they could be attached to one another, as we can see in figure 4.8. We designed these holders using CorelDraw and we cut the parts in acrylic using the laser cutter at our FabLab. In the final version, the holder attached to the first motor and the structure of the table (see Figure 4.7) was designed and build by Hendrik Kolvenbach from the FabLab. We went through several iterations

[2]available here: http://www.cadsoftusa.com/

[3]http://hci.rwth-aachen.de/fablab

[4]http://hci.rwth-aachen.de/fablab/fablab_mill_tutorial_en

Several holders were produced at the FabLab.

of our holders as during the test of the first prototype, we realized that the camera we used was too heavy and we replaced it by a lighter one. Also, the components attached to the motors' shafts were sliding, while the shafts themselves stayed in position. Components were attached to the shafts of the motors with a mounting hub, which were held in position with a screw. This screw however, did not have enough friction on the round shaft to hold. We needed to flatten the shafts so that the screw would have more grip.

We designed and built a new PCB with two hall sensors.

During the testing of the first prototype made with the PCB and all the hardware, it became clear that one hall sensor would not be enough to set a starting position. At an earlier stage, we thought that a magnet could be attached to the camera and that the sensor could use it to fix a default position for the camera. This proved impracticable and we decided to add a second hall sensor. This way, each sensor would be attached to a motor and its starting position could be set independently. The schematic and board layout were altered and a second iteration of the PCB was created using the method described above.

4.2.2 Firmware

Tutorials were used to learn how to use Arduino.

The microcontroller had to be programmed in order to control the stepper motors, read the data from the hall sensor and communicate with the computer. Before the implementation of the firmware could take place, we had to get familiar with the Arduino development environment and learn how to program the microcontroller and interface components. We used the tutorials available on the Arduino website[5] to this end.

The first part of the firmware we decided to work on was how to control the stepper motors. In theory, rotating a motor to a new position is very straightforward; the only information required is the number of steps the motor needs to turn, and in which direction. In practice, the actions required are more complex. The motors were each controlled by a driver carrier, and it was with these compo-

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

nents that the Arduino interacted. The drivers had five pins connected to the microcontroller (`enable, reset, sleep, step` and `direction`). In the setup function of the firmware, these pins had to be configured as input or output pins, which told the microcontroller if he should receive data from these pins or if he should send data to them. The drivers' pins were all set as output, as they are used to move the motors and do not send any feedback. During this setup phase, the `reset` and `sleep` pins had to be turned off, while the `enable` pin had to be turned on. This step ensures that the drivers were ready to receive instructions (`enable`) and that they would not power the motors down after they reached their position (`sleep`).

The pins had to be set up on the microcontroller before runtime.

We then wrote the piece of code to actually rotate the motors. The rotation is done through the use of two of the pins, `direction` and `step`. The direction is selected by setting the pin to high or low. The rotation process is done by alternating the value of the step pin and waiting a few microseconds between each instruction. This cycle had to be repeated until the number of steps had been reached.

The motors are rotated using two pins: direction and step.

```
funtion rotate(boolean dir, int steps){
  //rotate a specific number of microsteps

  setDirection(dir);
  for(int i=0; i < steps; i++){
    stepMotor(HIGH);
    wait();
    stepMotor(LOW);
    wait();
  }
}
```

When we first compiled and ran our code, the motors would not move, although no error was generated. After days of investigating, we realized that we configured the pins, but did not initialized them. This essential step was not clearly documented in the tutorials we used, and as this was our first endeavour in Arduino, we did not find the error straight away.

The pins had to be initialized for our code to run.

Once the motors were controlled by the Arduino and could

The microcontroller exchanged data with the computer through a serial connection.

be rotated as we wanted, the next step was to implement a method to communicate with the computer, so that we could send new coordinates to the motors. At this point, we were focused on developing read and write functions for the firmware, and we were not working yet on the software on the computer. This is why we used the Serial Example XCode project from Gable Ghearing[6] on our Mac, while we developed the firmware. Ghearing's code was able to connect to the Arduino and to send and receive data through a serial port, USB in our case.

The Arduino received a direction and a number of steps to rotate the motors.

During the design phase, we decided that the coordinates for each camera position would not be saved on the microcontroller, but on the agent running on the computer. The Arduino would only receive a direction and a number of steps to move each motor. We decided that the instructions for both motors would be sent as one string, in a predefined format; the regular expression for the string is:

```
X(+|-)[0-9][0-9][0-9][0-9]
Y(+|-)[0-9][0-9][0-9][0-9]
```

A regular expression was used for sending instructions to the microcontroller.

The letters represent the motors, the "+" and "-" symbols the direction, the numbers are the number of steps to perform from the origin. For instance, to move the motors to the position (+90;-30), the following string should be sent to the microcontroller: "X+0090Y-0030".

Characters received were stored in a buffer then the string was parsed.

When the agent sent such a string, the Arduino would receive one character at a time and store them in a buffer. Once the buffer contained 12 characters, the string was parsed, the direction and the number of steps for each motor were extracted and the `rotate` function was called. The previous positions for the motors were stored, so they could rotate to the new ones without going to the starting position first.

The starting position could be reset using the hall sensors.

The last functionality we implemented in the firmware was the possibility to move the motors to a zero position using the hall sensors. When this function was called, each motor would rotate until the magnet attached to the shaft was

[6]http://www.ghearing.com/Arduino%20Serial%20Example.zip

seen by the hall sensor. At which point the current position was saved as the starting position. Before this function could be called, the motors needed to be positioned so that when they rotated, the sensor would meet the magnet without having to perform a full revolution. This extra step was needed as when the hardware was fixed under the multitouch table, it was not possible for the motors to do a complete revolution anymore.

4.2.3 Software

An application on the computer was written to communicate with the microcontroller.

After the work on the firmware was complete, we implemented the agent that would run on the computer and interact with the firmware. This agent needed to connect to the microcontroller, set the starting position and the camera coordinates, and get feedback from the microcontroller. The agent would also store the list of camera positions and the associated motors' coordinates.

Our application used some already available functions.

During the firmware implementation, we used the Serial Example open source application from Gable Ghearing on the computer to test our code on the microcontroller. We used parts of this application in our agent, as it is a reference for the Arduino website. We reused the function opening the serial connection, the one sending a string and the one creating a separate thread listening to data coming from the Arduino. In addition to this code, we wrote new functions and created a new user interface to fit our needs.

The communication was done through a serial connection.

The agent could not call directly the functions on the Arduino. The only way for them to communicate was by exchanging messages through the serial connection. As we explained above, the agent sent a formatted string to move the camera to specific positions. The agent could also sent a special version of this string. When it was received by the microcontroller, the string would not be used as new coordinates, but instead the function setting the starting position would be called.

The agent needed to received feedback from the Arduino to confirm that the instructions had been carried out successfully and that the program could continue. In order to

The Arduino sent feedback to the agent when actions were completed.

achieve this, we modified the firmware to send these feedback messages to the agent. The firmware was sending three different messages; serial connection opened, motors in position, and starting position set. We used numerical codes to represent these messages for two reasons; they are shorter than their alphabetical counterparts, so they can be sent through the serial connection faster, and codes are easier to handle on the agent's side.

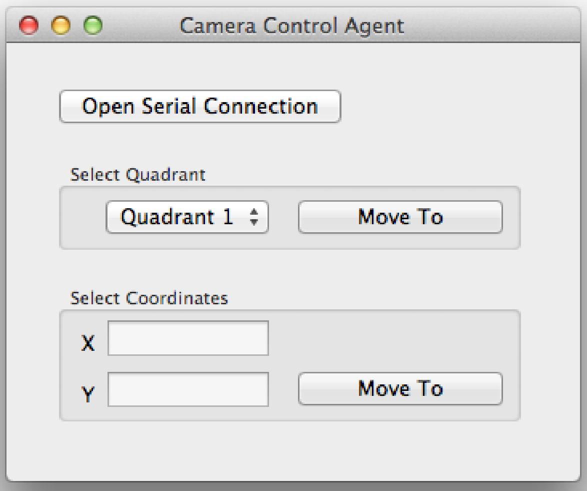


Figure 4.10: User interface for the Camera Control Agent. The connection to the microcontroller can be opened from that screen, the camera can be moved by selecting a quadrant or entering coordinates.

Messages were sent in several packets and were concatenated once received.

Despite the short length of these messages, they were not sent as one packet on the USB, but as several. We wrote a function on the agent that concatenated the bits of messages received from the Arduino. We also framed our messages with special characters, so the function would know when the message was complete. The agent was then able to receive feedback from the microcontroller, to differentiate the messages and act upon them.

The user interface we created was made as simple as possible (see Figure 4.10). The serial connection with the microcontroller could be opened with a click on a button, and the camera could be moved either by selecting one of the quadrants or by typing new coordinates. The starting position was reset by entering "+9999" in both coordinates' fields. As this agent was meant to run in the background when the Touch Sequence application was running, there was no output window for the feedback from the Arduino. These messages were logged and would appear in the console.

The interface was kept simple.

4.2.4 Starting Position Issue

After the moving camera was installed beneath the multi-touch table and testing began, we noticed that when setting the starting position, the motors did not always stop at the same position. This resulted in an inconsistent starting position. After investigating the firmware, it turned out that the code was working properly, and that the issue was hardware related; the hall sensors we used to position the motors were not sensitive enough. The distance from which they would sense the magnets was not always the same, and it explains why the motors were not always stopping at the same spot.

The hall sensors were not sensitive enough to set the starting position precisely.

As the hall sensors were not reliable enough, we designed another method to set the starting position, using the camera and the display on the table. We projected a crosshair on the table, and we aligned a piece of paper on it. The camera view was shown on the surface with a square drawn on it. Using the Camera Control Agent, we could move the camera to align the marker with the square. When the paper was within the square, we could reset the starting position by sending the (-9999;-9999) coordinates to the microcontroller. With this method, we were able to set a consistent and reliable starting position throughout the user study.

We designed another method to accurately set the starting position.

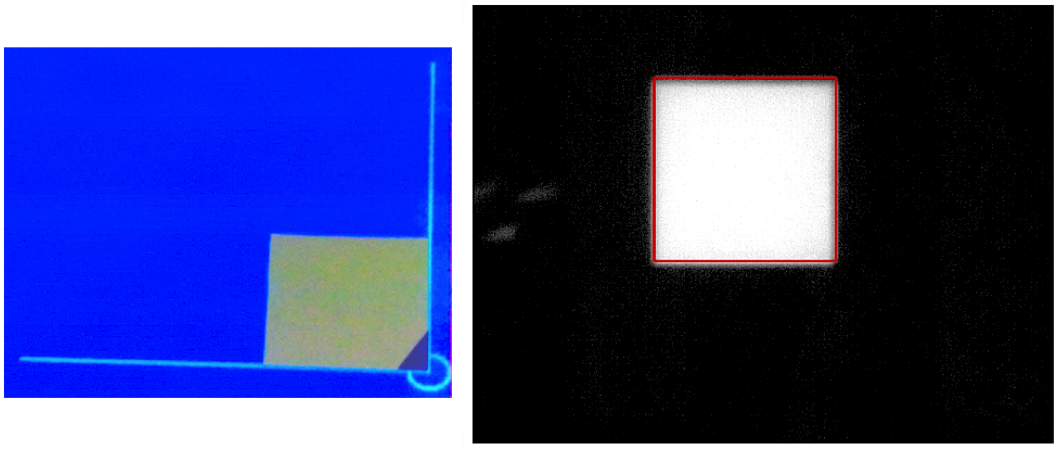


Figure 4.11: Left: view from the table with the calibration crosshair and marker. Right: camera image with frame to fit the marker.

4.3 Touch Sequence Display

The Touch Sequence Display presented the stimuli to the participants.

The Touch Sequence Display was the application used to run the final study presented in section 3.5. It presented the stimuli to the participants, received touch events from the MultiScreen agent and recorded data about these touch events.

The visualisation depended on the condition.

The stimuli used in the final study were six buttons displayed on a ring, with an additional button in the middle. These buttons were presented in sequences of two or three, they could be all visible from the start or were appearing one after the other, depending on the condition. The application contained a list of button objects with their coordinates on the surface and the quadrant they appeared in. The application also had sequences objects, which contained a condition id and pointers to button objects.

The conditions and sequences were ordered using balanced Latin Squares.

At each run, the application created a new unique list of sequences and conditions. The order of the conditions and the sequences inside were chosen using balanced Latin Squares to counterbalance the possible effect of the conditions' order. The last positions used in the Latin Squares were saved in a file, so for each user, a different order was used.

The sequences of a condition were shown one after the other, with the program looping them. After each condition, the program paused, allowing the participant to take a short break. Resuming the application was done by the experimenter hitting a key on the keyboard.

During the study, a break was made after each condition.

When a sequence was displayed, the active button was green while the others were grayed. Beneath the active one was a hidden rectangle, acting as an activation area. When the Touch Sequence Display received a notification from the MultiScreen agent that a touch had been registered, it checked whether the center of this touch was within the activation area. If so, the touch was validated, the next button would be activated and would receive the activation area. The application would also send a request to the Camera Control Agent to move the camera to the required quadrant. We will detail the communication between the different programs in section 4.4.

When a touch was registered on a button, the camera moved to the next one, which became active.

We decided to have an activation area twice as large as the button for several reasons; the test we performed to validate a touch was to check that the center of the touch was inside the activation area. If only half of the fingerprint was in the activation area, the center could be out of it, and then the touch would not be validated. Also, we did not want participants to try to be as precise as possible, which they would have done, if they felt that touches were not being registered. Finally, even if the touch was not recorded, the participant would still move to the next target, before coming back to the first one. In that situation, the touch sequence would not be admissible and the data for the predecessor effect would be corrupted.

The activation area was twice as big as the button.

While we were testing the Touch Sequence Display in conjunction with the MultiScreen agent and the Camera Control agent, we discovered that sometimes the Touch Sequence Display would not receive touch events from the MultiScreen agent any longer, although the latter was still running and no errors were generated. As we could not find the source of this issue, we designed a workaround.

The communication between the different agents would sometimes randomly fail.

If touches were not received by the Touch Sequence Display, we could export the data generated up to this point to a file, terminate the program, and restart it at the exact

The application could be restarted at the last sequence used.

same point we stopped, with the same order for the conditions and sequences.

4.4 Modules Communication

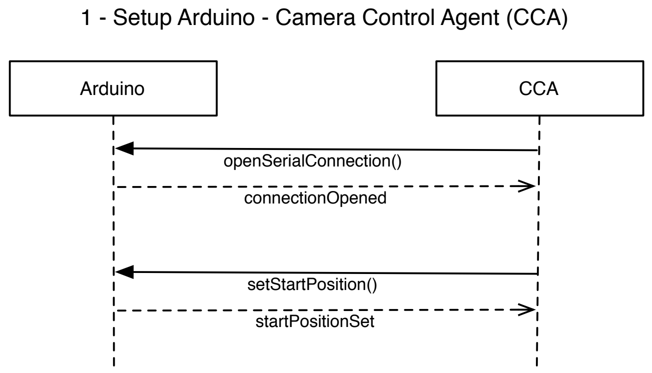


Figure 4.12: Sequence diagram detailing the messages exchanged to initiate the camera position. Messages are exchanged through the serial connection.

Our system was made of three independant applications.

In designing the architecture for the software we would develop, we decided to use three independent applications instead of having all functionalities in a single program. The MultiScreen agent was developed by our chair as a stand alone application, not as a piece of code that should be included in bigger project. We decided that the functions we wrote to control the camera movements should be in an independent program as their actions are very specific to the motors' control, but at the same time, they could be used by any other application that needed to drive motors. Hence, the Camera Control Agent and the Touch Sequence Display being separate programs.

The three applications and the Arduino needed to exchange information with one another.

The architecture was made of four objects; the MultiScreen agent, the Camera Control agent, the Touch Sequence Display and the Arduino microcontroller. These objects needed to communicate with one another. As we stated earlier, the Arduino and the Camera Control Agent exchanged messages through a dedicated serial connection. The three applications were running on the same computer,

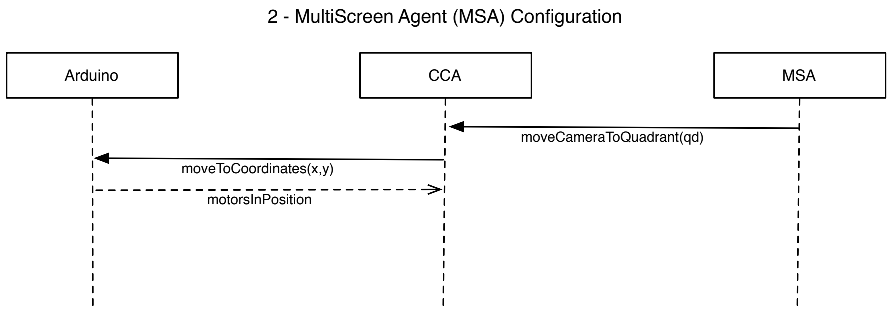


Figure 4.13: Sequence diagram detailing the messages exchanged when configuring the quadrants during the setup phase.

and were written in Objective-C. This allowed us to use notifications between the different objects of our applications.

Notifications are part of the Foundation framework, which is a base layer for Objective-C classes. Notifications are objects containing information about an event. They can be posted to Notification Centers, which broadcast them to all registered objects. Any object can register for any notification, which makes notifications a very practical system to exchange information between applications.

Notifications are convenient to exchange data between applications on the same computer.

Each application had a notification center and broadcasted particular events. The applications registered to the notification centers from which they required information.

Each application had a notification center.

During the setup phase of the MultiScreen agent, the camera needed to be moved to each quadrant to configure the Spot Detection and do the mapping. The MultiScreen agent used its notification center to broadcast a request to move the camera to a new position. The new quadrant was included in the broadcast. The Camera Control agent was an observer of that notification. When it was broadcasted, it was notified and instructed the microcontroller to move the camera to the new coordinates. The message flow can be seen in figure 4.13.

A notification was sent from the MultiScreen Agent the Camera Control Agent to to move the camera.

When the experiment was conducted, the MultiScreen agent, the Touch Sequence Display and the Camera Control agent needed to exchange information with one another. The message flow during runtime can be seen in figure

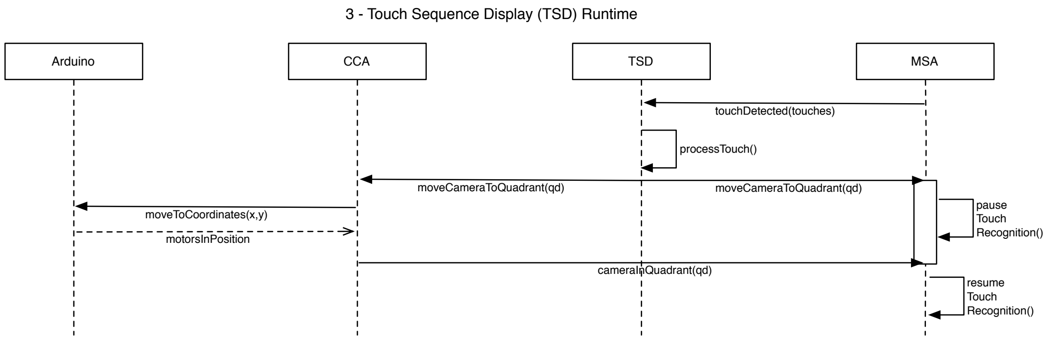


Figure 4.14: Camera control board layout.

Wehn a touch was registered, all the components of the system interacted.

4.14. When the participant touched the surface, the MultiScreen agent notified the Touch Sequence Display, which checked the touch and recorded its information. It then notified the Camera Control and the MultiScreen agents that the camera needed to be moved. The Camera Control agent worked as described above. The MultiScreen agent used this notification to pause the touch recognition process until the camera was in position. The touch recognition process was resumed using the new quadrant object.

The Touch Sequence Display did not receive feedback from the other applications.

The Touch Sequence Display did not receive any notification about the state of the camera or the state of the touch recognition process. This was not necessary for two reasons; first of all, when the camera was moving and the touch recognition was suspended, it was transparent for the display application, as the only difference was that it did not receive any touch events. The whole process of changing quadrants and moving the camera was done in a fraction of a second, which meant that the camera was in position and the MultiScreen agent was capturing before the participant would reach the button.

Chapter 5

Results

We had two hypotheses; the predecessor touch had an impact on the finger orientation on the next touch and on its position. In the same way, we enunciated that the successor touch influenced the finger orientation and position on the previous touch. In order to validate our hypotheses, we ran the study described in section 3.5. This chapter presents the results of this experiment.

Two hypotheses: predecessor and successor effect on the finger orientation and touch offset.

In our study, we had four different conditions; two experimental ones and two control ones:

User study with four conditions.

1 *Control1 (with both effects)*: the first, middle and last buttons are displayed simultaneously, thus the middle button is influenced by both the predecessor and successor touches.

2 *Control2 (with no effect)*: the middle button is shown, the last one is displayed after the middle one has been touched. Neither predecessor nor successor effect should be seen.

3 *Experimental1 (with the predecessor effect only)*: the first and middle buttons are displayed at the beginning, the third one appears after the middle one has been touched. The touch on the middle button should presents evidence of the predecessor effect.

4 *Experimental2 (with the successor effect only)*: the middle and the last buttons are shown at the same time,

there is no predecessor for the middle button, thus only a successor effect should be observed on the touch on the middle button.

The buttons were numbered and on a ring.

Six buttons were displayed on a ring, with an extra button in the center of the layout (see Figure 5.1). The buttons were numbered, and we will refer to these numbers when discussing particular buttons.

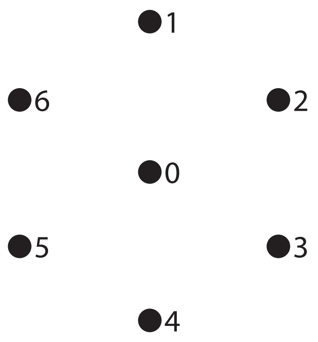


Figure 5.1: Button layout during the experiment, with their associated id.

Participants had experience on small touchscreen devices.

Once the participants finished the experiment, we asked them to fill a short questionnaire (see Appendix B.1). The results from this questionnaire show that our participants had experience using computers (average of 4.15 out of 5) and using touch screens as big as a phone (average of 4.15 out of 5) or a monitor (3.46 out of 5). On the other hand, they had little to none experience with bigger touch display like tables (average of 1.77 out of 5) or wall (average of 1.23 out of 5).

Participants found the task easy to understand and to perform.

When asked about the difficulty to understand the task or to accomplish it, participants were satisfied. The understandability was rated 5 out of 5, while the easiness got a score of 4.69 out of 5. They reported no problem when using the system, but 3 participants felt that the system was responding too slowly to their touches.

We recorded information about each touch.

During the experiment, for each touch on a button, we recorded the finger orientation, the touch location, the condition we were in, and the predecessor and successor touch. We used this data to run several repeated measures one-way ANOVA to analyze the successor and predecessor effects.

The study was conducted with 14 participants (2 females), all right-handed. The participants' handedness was determined using the Edinburgh Handedness Inventory [Oldfield, 1971]. Eleven participants were studying Computer Science, one Biology, one Electrical Engineering and one Mathematics. They were between 20 and 31 years old. Drinks and snacks were provided during the study and a raffle for a 20 euros voucher at Amazon was made.

14 participants took part in this study.

In the next section of this chapter, we will present the results of the predecessor effect and successor effect on the finger orientation on touch 0 (the middle button), then in the following section, we will look at these effects on the touch location, on touch 0.

We will look at the effects on the finger orientation then on the touch offset.

5.1 Effects on the Orientation of The Finger

5.1.1 Predecessor Effect

We ran a repeated measures one-way ANOVA using the data from the *Experimental1* condition, where only the predecessor effect should be seen. Having a different predecessor touch had a significant effect on the finger orientation ($F_{5,60} = 22.367, p < 0.001$). Pair-wise comparisons using Bonferroni-corrected confidence interval showed that the finger orientations were significantly most different when the previous touch was either 1 or 2 on one hand, or between 3 and 6 on the other hand (all $p < 0.043$).

The predecessor touch had a significant effect on the finger orientation.

As we can see in figure 5.1, we can group touches 1 and 2 in a region, and the other ones in another region. Another one-way ANOVA was performed after we aggregated the data in these two regions. We found a significant main effect of the region on the finger orientation

After grouping buttons, the predecessor effect was stronger.

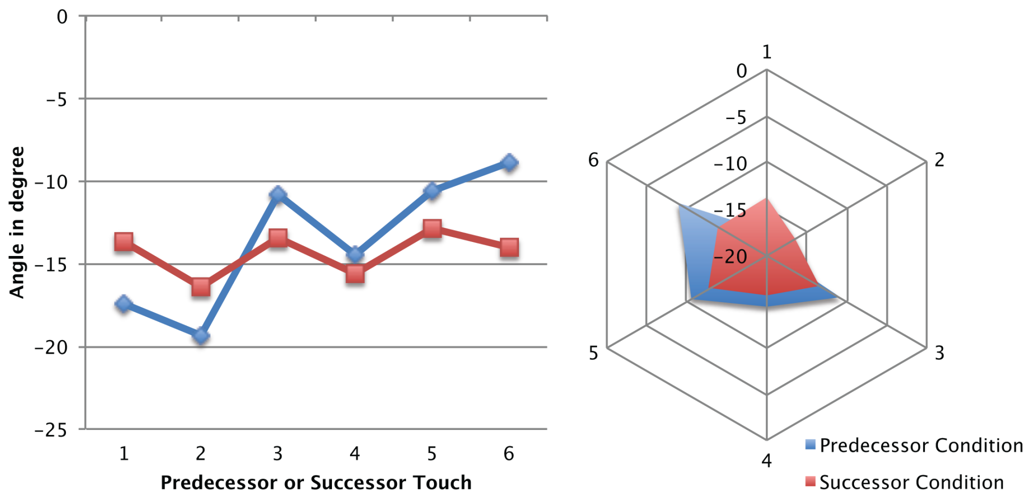


Figure 5.2: Orientation of the finger depending on the predecessor or successor touch only.

$(F_{1,12} = 50.048, p < 0.001)$.

The predecessor touch had a significant effect on the orientation even in the Control1 condition.

We then compared the data from the *Experimental1* condition (*predecessor only*) to the *Control1* condition (*predecessor and successor*). In *Control1*, sequences had three buttons, with a predecessor, the middle button and a successor. In order to compare the data to *Experimental1*, where only two buttons were used (a predecessor and the middle button), in *Control1*, we aggregated all the trials per predecessor touch, thus removing the successor. A repeated measures one-way ANOVA found a significant main effect of the predecessor touch on the finger orientation $(F_{5,60} = 26.680, p < 0.001)$.

There was an interaction between the Experimental1 and the Control2 conditions.

Comparing the *Experimental1* condition with the *Control2* condition (*no predecessor, nor successor*), a repeated measures one-way ANOVA showed a main effect of the predecessor touch $(F_{5,60} = 14.307, p < 0.001)$. This effect was significant because it was already significant in the *Experimental1* condition alone. A clear interaction between the two conditions was seen $(F_{5,60} = 11.339, p < 0.001)$.

There was no main effect in the Control2 condition.

There was no interaction between the *Experimental1* condition and the *Control1* condition and no main effect of the condition (both $p > 0.317$). Comparing the *Experimen-*

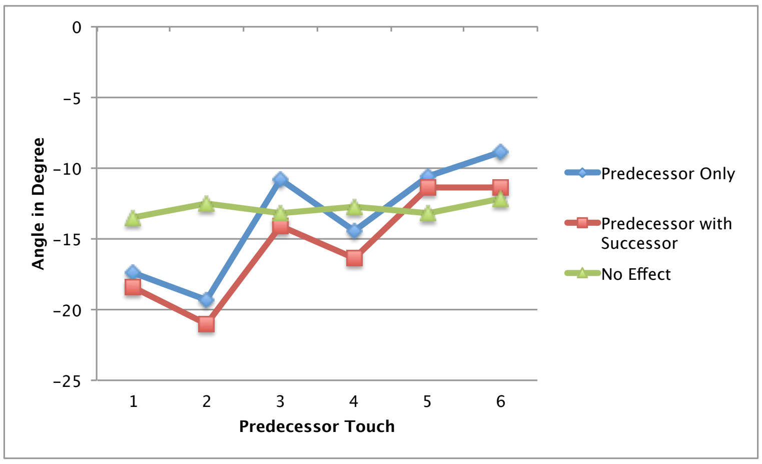


Figure 5.3: Orientation of the finger depending on the predecessor touch. Each line represents a different condition.

tal1 condition with the *Control2* condition, a repeated measures one-way ANOVA showed that the condition without predecessor nor successor did not have any main effect ($F_{1,12} = 0.190, p = 0.671$).

5.1.2 Successor Effect

Using the data from the *Experimental2* condition, we removed the outliers and ran a repeated measures one-way ANOVA. The outliers were defined as being more than two standard deviations away from the mean. From this, we found that the successor touch had only a small underlying effect on the finger orientation ($F_{5,35} = 1.658, p = 0.171$).

After filtering the data, there was a small underlying effect of the successor touch on the finger orientation in Experimental2. There was a main effect of the successor touch on the finger orientation when comparing Experimental 2 with Control 1.

We then compared the *Experimental2* condition with the *Control1* condition. As for the predecessor effect, we aggregated trials from *Control1*, but per successor this time. A repeated measures one-way ANOVA found a significant main effect of the successor touch on the finger orientation ($F_{5,60} = 6.287, p < 0.001$).

A one-way ANOVA, without removing the outliers from the *Experimental2* condition, showed no significant main effect of the successor touch ($F_{5,60} = 1.567, p = 0.183$).

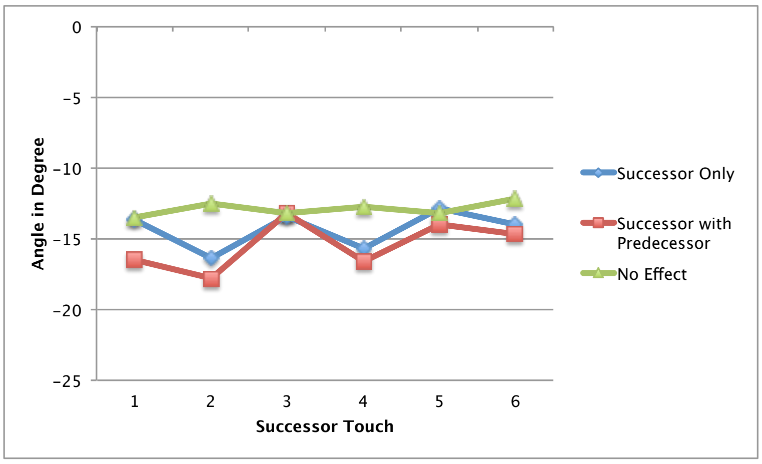


Figure 5.4: Orientation of the finger depending on the successor touch. Each line represents a different condition.

Without filtering the data, there was no main effect on Experimental2.

Bonferroni-corrected pair-wise comparisons yielded no information. There was no interaction between the *Experimental2* and the *Control1* conditions, nor any main effect of the condition (both $(p > 0.585)$.

There was no main effect when comparing Experimental2 with Control2.

Comparing the *Experimental2* condition with the *Control2* condition, a repeated measures one-way ANOVA found no main effect for the successor touch, for the condition, nor any interaction between the two conditions (all $p > 0.126$).

5.2 Effects on the Touch Location

We split the touch offset in an x-offset and a y-offset.

We investigated the predecessor and successor effect on the touch location in the same as we did for the finger orientation. In order to analyze these effects on the touch location, we decided to look separately at the offset on the x-axis, and on the y-axis. This way, we looked for a predecessor effect and a successor effect on the x-offset and on the y-offset.

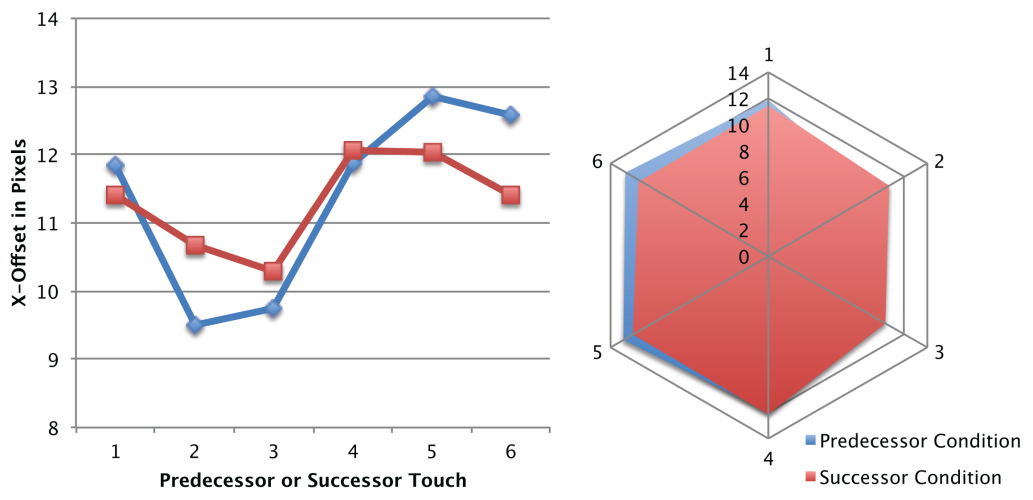


Figure 5.5: X-offset depending on the predecessor or successor touch only.

5.2.1 Predecessor Effect

A repeated measures one-way ANOVA found a significant main effect of the predecessor touch on the x-offset ($F_{5,60} = 12.082, p < 0.001$) in the *Experimental1* condition. We also found a significant main effect of the predecessor touch on the y-axis ($F_{5,60} = 6.830, p < 0.001$).

There was a main effect of the predecessor touch on both offsets.

We then compared the data from the *Experimental1* condition with the *Control1* condition. As for the finger orientation, we aggregated all the trials per predecessor touch, thus removing the successor. A repeated measures one-way ANOVA found a significant main effect of the predecessor touch on the x-offset ($F_{5,60} = 13.793, p < 0.001$), and on the y-offset ($F_{5,60} = 7.372, p < 0.001$).

There was a main effect on both offsets even in Control1.

Comparing *Experimental1* with *Control2*, a main effect of the predecessor touch was observed on the x-offset ($F_{5,60} = 4.631, p < 0.001$. This effect was significant because it was already significant in the *Experimental2* condition alone. An interaction between the two conditions was seen on the x-offset ($F_{5,60} = 3.376, p < 0.009$), and on the y-offset ($F_{5,60} = 5.201, p < 0.001$).

A main effect on the x-offset was observed in Control2.

An ANOVA on the y-offset found no significant main effect of the predecessor touch ($F_{5,60} = 1.491, p = 0.206$). There

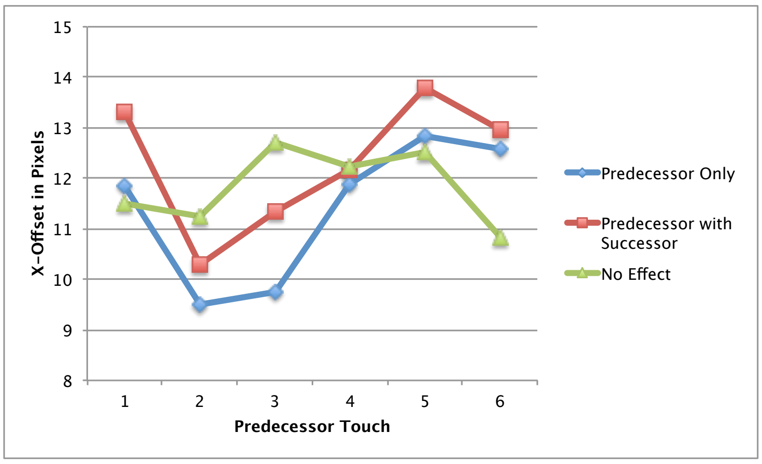


Figure 5.6: X-offset depending on the predecessor touch. Each line represents a different condition.

There was no main effect in Control2.

was no interaction between the *Experimental1* and *Control1* conditions, neither on the x-offset, nor on the y-offset (both $p > 0.317$). A comparison of *Experimental1* with *Control2* showed that *Control2* had no main effect on the x-offset or y-offset (both$p > 0.238$).

5.2.2 Successor Effect

There was a significant effect on the x-offset when comparing Experimental2 with Control1.

We compared the *Experimental2* condition with the *Control1* condition; as for the predecessor effect, we aggregated trials from the first condition, but per successor this time. A repeated measures one-way ANOVA found a significant main effect of the successor touch on the x-offset ($F_{5,60} = 4.660, p < 0.001$).

There was no main effect on the y-offset.

An ANOVA on the y-offset found no significant effect ($F_{5,60} = 1.357, p = 0.253$), and no interaction between the two conditions, either on the x-offset or on the y-offset (both $p > 0.151$).

Comparing Experimental2 with Control2, there was no main effect on any offset.

Comparing the *Experimental2* condition with the *Control2* condition, a repeated measures one-way ANOVA found no main effect for the successor touch on the x-offset or on the y-offset (both ($p > 0.217$), nor any interaction between the

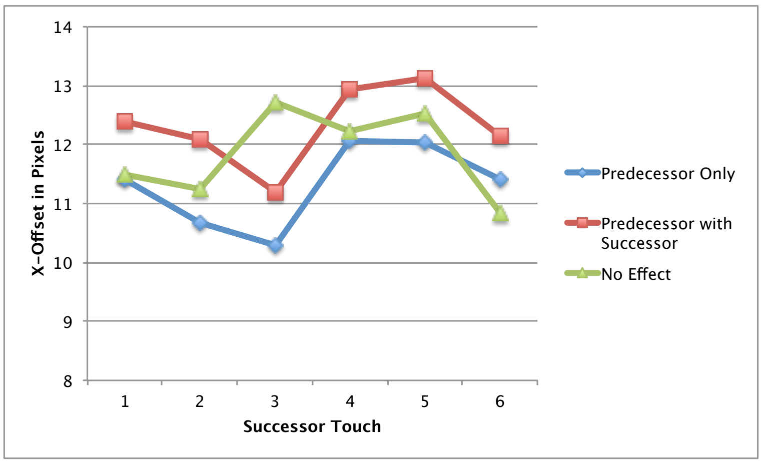


Figure 5.7: X-offset depending on the successor touch. Each line represents a different condition.

two conditions, either for the x-offset or the y-offset (both $p = 0.477$).

There was no main effect on any offset in Experimental2.

We ran a one-way ANOVA using the data from the *Experimental2* condition; no significant main effect of the successor touch was found, either on the x-offset or on the y-offset (both $(p > 0.344)$. Bonferroni-corrected pair-wise comparisons yielded no information, neither for the x-offset, nor for the y-offset.

5.3 Discussion

There is a predecessor effect on the finger orientation.

Results showed that the finger orientation was influenced by the predecessor touch. This effect could be seen in the *Experimental1* condition only, but also when we compared it against the other conditions. In *Control1*, the predecessor effect was still visible, although it could have been diminished by the successor effect or the longer touch sequence. In *Control2*, the finger orientation was constant. This tend to show that it was not influenced by anything, which concur with our hypothesis. When compared to *Experimental1*, the main effect of the predecessor touch that we observed, combined to the clear interaction between the two conditions, showed a strong influence of the predecessor touch on the finger orientation.

There is no successor effect on the finger orientation.

Regarding our second hypothesis, results showed that there was no successor effect on the finger orientation. No main effect of the successor touch was seen, except when *Experimental2* was compared to *Control1*. Although there was no interaction between the two conditions. This effect can be disregarded, as the values for the finger orientation ranged within a 3° interval only.

The x-offset is influenced by the predecessor effect.

The touch location was impacted by the predecessor touch, and this was more obvious for the x-offset. For this one, a main effect was observed in all the conditions. Also, there was no interaction between the *Experimental1 Control1*; the predecessor effect on the x-offset could be seen even in the longer touch sequence. The interaction between *Experimental1* and *Control1* was another indication of the predecessor effect on the x-offset.

The predecessor effect on the y-offset is weaker than on the x-offset..

The predecessor effect on the y-offset was also observed in *Experimental2* and *Control1*. The comparison between *Experimental2* and *Experimental1* yielded less clear results. The hypothesis that the touch location was influenced by the predecessor touch was also validated, although it would seem that the power of the predecessor touch is bigger on the x-offset than on the y-offset.

There is no successor effect on the touch offset.

As for the finger orientation, results showed that there was no successor effect on the touch location, either on the x-offset or on the y-offset.

Chapter 6

Summary and Future Work

This work produced two contributions, a minor technical one and a major scientific one. The technical contribution is a system combining hardware and software that allows a multitouch table to have a camera resolution of 160ppi and to capture highly accurate touches positions and shapes. The scientific contribution expanded on Holz' work [2010]. The study we conducted showed significant results for the predecessor effect on the finger orientation and the touch location, but no proof of the successor effect has been found. The predecessor effect can be used to increase the accuracy of any touchscreen device.

We built a high definition touch detection system and proved the existence of the predecessor effect.

In this chapter, we will summarize our thesis and introduce the areas that will be investigated as part of our future work.

6.1 Summary

Multitouch tables are slowly coming into our lives. Researchers have been exploring the field for decades and commercial versions of these tables have been on the market for some years now. The accuracy on touch screens has been an issue from the very beginning and is still a signifi-

Multitouch tables suffer from a lack of input accuracy.

cant one. The most common explanation for the inaccuracy is the *fat finger problem*, although a review of the existing work on the subject showed that Holz et al. presented the *perceived input point model* as an alternative explanation.

Holz stated that the touch location depends on the finger orientation in space.

They state in their model that touches on a screen are not reported at the actual touch location but at an offset from the intended target. The study they conducted showed that the offset depends on the finger location on the screen but also on the finger position in space. This offset can then be compensated to correct the touch location. They created a prototype based on a fingerprint scanner, which used the fingerprints to identify the user and the finger position in space. The study they ran with this prototype confirmed their findings. Other researchers worked on addressing the occlusion problem. Instead of trying to make touch screens more accurate, they designed targeting aid systems to work around these limitations. With these systems, it was possible to select very small targets, either by decoupling the user's touch and the target, or by using callouts to show the occluded part of the screen beneath the finger.

Systems use finger orientation, but not to increase input accuracy.

Research has been done on the use of body posture and finger position in particular. Several systems make use of the finger orientation, not in order to increase the touch accuracy, but to provide more functionalities and new interaction techniques. Body posture has been used for a long time in the field of augmented or virtual reality. Body posture has also been used to implement a pointing gesture based on the arm orientation, the line of sight between the head and the finger and the head orientation.

Several techniques exists to detect the finger orientation.

The literature offers several ways of detecting the orientation of a finger. The whole shape of the hand can be used in computer vision to detect the fingertips and deduce the orientation or the changes in the contact shape area allows to detect the orientation by tracking the movements of the center of the touch. Additional hardware can be used, like gloves with tags on the fingertips or an array of capacitive proximity, which can detect the hovering finger and its orientation.

Our hypotheses focused on the touch sequence effect.

Our set of hypotheses was refined, at the same time as our study design matured throughout the different iterations.

We focused on touch sequences and how they could influence the position of the finger and the touch location on the table.

We built a high definition touch detection system.

We implemented a touch detection agent based on the MultiScreen Agent from our chair, which worked with one moving camera and had a very high resolution of 160ppi. We built the hardware and wrote the code to control it. For our study, we also designed and implemented the application showing the stimuli to the participants and recording the data.

We proved the existence of the predecessor effect.

We tested our hypotheses in a user study. Participants used our system to touch buttons arranged in touch sequences, while we recorded the orientation of their finger and the touch location. The analysis of the data proved that the predecessor effect hypothesis was correct and showed significant effects for both the finger orientation and the touch location. The successor effect hypothesis was not proven to be true.

6.2 Future Work

The final user study yielded interesting results and confirmed one of our hypotheses. The results also opened the door for several new directions in the future work.

The study could be run again without instructing the participants to use their fingerpad.

During the course of our study, in order to have touches that were big enough and with a good ellipsoid shape, we instructed the participants to use the pad of their finger. This is not the most natural way people interact with touch screens. Usually, they try to have the smallest contact area possible, using their fingertip, precisely because of the inaccuracy of this input method. We believe our study should be run once more, without this limitation. This, however, means that the touch detection system must be rethought; the actual software and hardware cannot detect the finger orientation with only the fingertip touching the surface. Optical tracking could be used, but the marker and accuracy issues still need to be solved (see Chapter 4).

The impact of the finger pitch and roll on the predecessor effect could be investigated.

In our study, we narrow our hypotheses on the predecessor and successor effect by looking only at the yaw angle of the finger with the surface. It would be interesting to add the pitch and roll as well to the equation. The question would be to see whether the added parameters would help correcting the offset even more, or if the yaw is sufficient for this task.

A deeper study could be made to assess the successor effect.

The results of our study showed clear evidence of the predecessor effect, while the presence of the successor effect was not seen. The study could be adapted to balance evenly the number of trials in each condition, and it study should be continued with a larger number of participants, and the data analyzed between subjects. The presence or absence of the successor effect could then be clearly solved. If the successor effect is proved to exist, we could then compare its power to the predecessor effect.

We could built a prototype using the predecessor effect to correct touch offsets.

Now that the predecessor effect has been identified, we could design a model that would calculate offsets for a particular user, based on the predecessor and finger orientation. A program could implement that model, and a study could be run to measure the effects of that model on the touch accuracy on tabletops. This model could also be implemented in any touchscreen device to improve its accuracy.

Research the impact of body posture on accuracy accross a table.

Finally, in section 3.2.2 we designed a second part to our experiment, in which the body posture effect on accuracy when using tabletops could be investigated without having any interference from the predecessor effect or successor effect. This research would complement our work on the predecessor and successor effect.

Appendix A

Camera Control

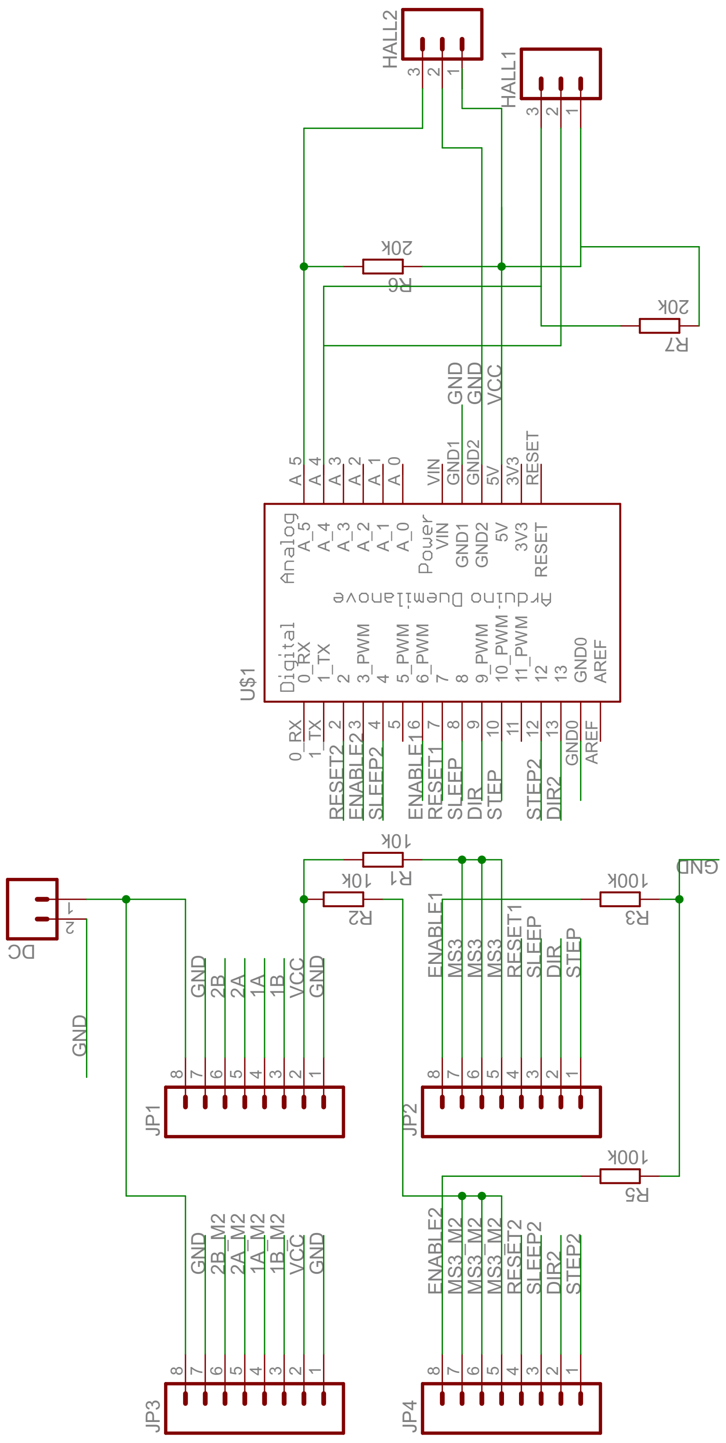


Figure A.1: Camera control schematic

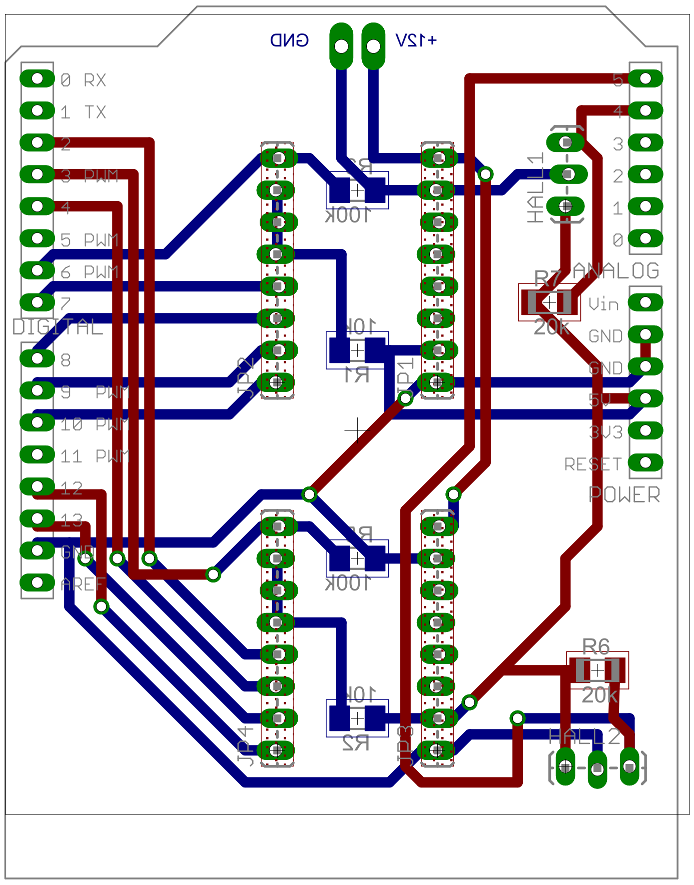
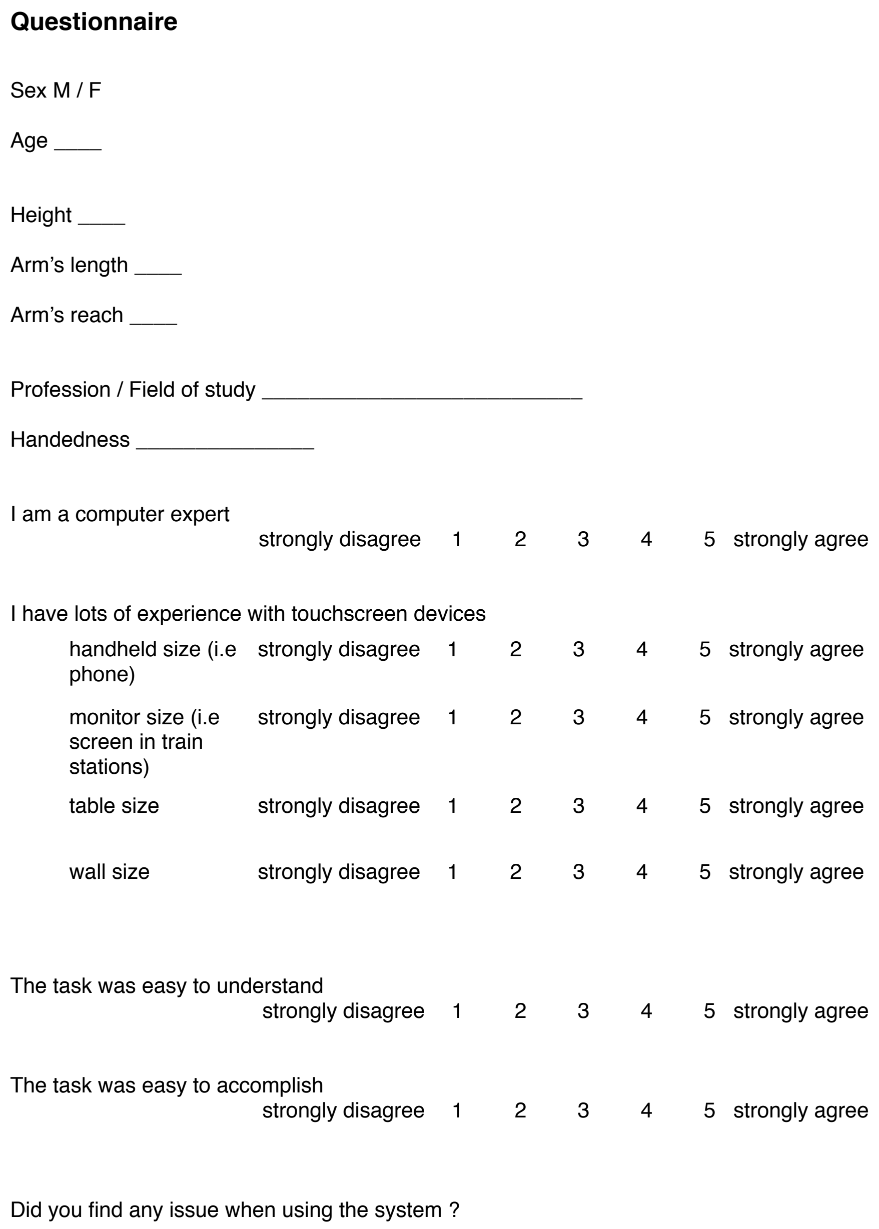


Figure A.2: Camera control board layout

Appendix B

Questionnaire for User Study



Questionnaire

Sex M / F

Age ____

Height ____

Arm's length ____

Arm's reach ____

Profession / Field of study ____________________________

Handedness ________________

I am a computer expert

strongly disagree	1	2	3	4	5	strongly agree

I have lots of experience with touchscreen devices

handheld size (i.e phone)	strongly disagree	1	2	3	4	5	strongly agree
monitor size (i.e screen in train stations)	strongly disagree	1	2	3	4	5	strongly agree
table size	strongly disagree	1	2	3	4	5	strongly agree
wall size	strongly disagree	1	2	3	4	5	strongly agree

The task was easy to understand

strongly disagree	1	2	3	4	5	strongly agree

The task was easy to accomplish

strongly disagree	1	2	3	4	5	strongly agree

Did you find any issue when using the system ?

Figure B.1: Questionnaire for User Study

Bibliography

Pär-Anders Albinsson and Shumin Zhai. High precision touch screen interaction. CHI '03, pages 105–112, 2003.

Chi Tai Dang and Elisabeth André. Usage and recognition of finger orientation for multi-touch tabletop interaction. INTERACT '11, pages 409–426, 2011.

Anthony D Hall, James B Cunningham, Richard P Roache, and Julie W Cox. Factors affecting performance using touch-entry systems: Tactual recognition fields and system accuracy. *Journal of Applied Psychology*, 73(4):711–720, 1988.

Christian Holz and Patrick Baudisch. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. CHI '10, pages 581–590, 2010.

Christian Holz and Patrick Baudisch. Understanding touch. CHI '11, pages 2501–2510, 2011.

Shahzad Malik and Joe Laszlo. Visual touchpad: a two-handed gestural input device. ICMI '04, pages 289–296, 2004.

Nicolai Marquardt, Johannes Kiemer, and Saul Greenberg. What caused that touch?: expressive interaction with a surface through fiduciary-tagged gloves. ITS '10, pages 139–142, 2010.

Kai Nickel and Rainer Stiefelhagen. Pointing gesture recognition based on 3d-tracking of face, hands and head orientation. ICMI '03, pages 140–146, 2003.

R.C. Oldfield. The assessment and analysis of handedness: The edinburgh inventory. *Neuropsychologia*, 9(1):97 – 113, 1971.

R. L. Potter, L. J. Weldon, and B. Shneiderman. Improving the accuracy of touch screens: an experimental evaluation of three strategies. CHI '88, pages 27–32, 1988.

Simon Rogers, John Williamson, Craig Stewart, and Roderick Murray-Smith. Anglepose: robust, precise capacitive touch tracking via 3d orientation estimation. CHI '11, pages 2575–2584, 2011.

Daniel Vogel and Patrick Baudisch. Shift: a technique for operating pen-based interfaces using touch. CHI '07, pages 657–666, 2007.

Feng Wang and Xiangshi Ren. Empirical evaluation for finger input properties in multi-touch interaction. CHI '09, pages 1063–1072, 2009.

Feng Wang, Xiang Cao, Xiangshi Ren, and Pourang Irani. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. UIST '09, pages 23–32, 2009.

Malte Weiss, Simon Voelker, Christine Sutter, and Jan Borchers. Benddesk: dragging across the curve. ITS '10, pages 1–10, 2010.

Typeset June 4, 2012