Multi-touch Interaction with Gesture Recognition

Espen Solberg Nygård

Master of Science in Computer Science
Submission date: June 2010
Supervisor: Asbjørn Thomassen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

As a prerequisite for this thesis a multi-touch table prototype has been built. The core technology used in this table is the technology DSI (Diffused Surface Illumination), and software based on open standards.

This project should complete and perfect the multi-touch table, and in addition explore the limitations in object recognition and gesture recognition using the DSI technology. A complete gesture recognition system should be implemented, and tested in a multi-touch environment.

Assignment given: 18. January 2010
Supervisor: Asbjørn Thomassen, IDI

Abstract

This master's thesis explores the world of multi-touch interaction with gesture recognition. The focus is on camera based multi-touch techniques, as these provide a new dimension to multi-touch with its ability to recognize objects. During the project, a multi-touch table based on the technology *Diffused Surface Illumination* has been built. In addition to building a table, a complete gesture recognition system has been implemented, and different gesture recognition algorithms have been successfully tested in a multi-touch environment. The goal with this table, and the accompanying gesture recognition system, is to create an open and affordable multi-touch solution, with the purpose of bringing multi-touch out to the masses. By doing this, more people will be able to enjoy the benefits of a more natural interaction with computers. In a larger perspective, multi-touch is just the beginning, and by adding additional modalities to our applications, such as speech recognition and full body tracking, a whole new level of computer interaction will be possible.

Contents

List of Figures

List of Tables

Listings

Preface

This master's thesis was carried out at the Norwegian University of Science and Technology, within the Intelligent Systems Group at Department of Computer and Information Science, Trondheim, Norway. It is, along with a multi-touch table prototype, to be considered as my final delivery. I would especially like to thank supervisor Asbjørn Thomassen for guidance and help during this thesis. I would like to thank the twelve students who helped me test the gesture recognition system implemented. I would also like to thank the people from the NUI group community, which have helped me solve problems that have arisen during the multi-touch prototype build. I would also like to send a big thank to Ida Brenna and Ståle Hoberg for proofreading my thesis. Finally I would like to thank my parents, especially my father Arne Roar Nygård, for inspiration and support.

Trondheim, June 14, 2010

Espen Solberg Nygård

Definitions, Acronyms, and Abbreviations

Provided in Table 1 and 2 are some definitions, acronyms and abbreviations used in this document. Any occurrences should be interpreted through these definitions.

Name	Definition
ANN	Artificial Neural Network.
Band pass filter	Filter used in many cameras to remove light of specific wave lengths. The ”band” defines the wave length of interest.
Blob	Term used in camera based multi-touch. A detected region in an image that differentiates itself from the surroundings. For example, a touch from a finger.
Compliant surface	Term used in the camera based multi-touch technique FTIR. It is a material meant to enhance the contact area between fingers touching the screen surface to create stronger blobs. A layer of silicone is often used for this purpose.
DI	Diffused Illumination, a special optical camera based multi-touch technique.
Diffuser	A material that diffuses light. Used to spread light evenly over a surface.
Direct gesture	A gesture that directly manipulates an object.
DLP	Digital Light Processing, an imaging technique used in projectors.
DSI	Diffused Surface Illumination, a special optical camera based multi-touch technique.

Table 1: List of definitions, acronyms and abbreviations used in this document

Name	Definition
EndLighten	Special type of glass used in DSI. It has thousands of microscopic mirrors inside used to distribute light shining in at the edges out from the surface of the glass.
Fiducial	An object marker in camera based multi-touch. Can be a special pattern recognized by the camera.
FLOSC	Gateway application for TUIO to translate from the protocol UDP to TCP. Mainly used in flash applications because of lack of UDP support.
FTIR	Frustrated Total Internal Reflection, a term used to describe a special optical camera based multi-touch technique.
Gesture	A movement that can be sensed. Often used in multi-touch as an input method.
Gesture stroke	A sequence of points that together make a gesture.
HID	Human Interface Device.
IUI	Intelligent User Interface.
LCD	Liquid Crystal Display, an imaging technology used in monitors and projectors.
LED	Light Emitting Diode.
LED-LP	LED Light Plane,a special optical camera based multi-touch technique.
LLP	Laser Light Plane, a special optical camera based multi-touch technique.

Table 2: List of definitions, acronyms and abbreviations used in this document

Name	Definition
Multi-user	More than one user using an application at once.
Multi-touch	A person using more than one finger to control an application.
OSC	Open Sound Control, protocol used to transfer TUIO.
Projection surface	A type of material that diffuses an image projected onto it. Visualization of an image.
ROI	Rate Of Income, used to describe how profitable an investment is.
Symbolic gesture	A gesture drawn as a symbol with a specific meaning.
Tracker	An application that takes a series of images, and reports the position, size, and relative movements of objects within images.
TUIO	Tangible User Interface Objects, way of describing what type of objects present in an image.
WIMP	Window, Icon, Menu, Pointing device, used to describe standard user interfaces.

Table 3: List of definitions, acronyms and abbreviations used in this document

Chapter 1

Introduction

This master's thesis is a continuation of a multi-touch interaction project from fall 2009. In the previous project a build of a multi-touch table prototype was started. This table uses a camera-based approach to multi-touch, and therefore this will be the focus in this thesis, including both hardware and software. The technology chosen is the technology I found most promising during the initial research. During this master's thesis the prototype is going to been further enhanced, in respect to design, function, hardware and software.

The goal is to create a fully functional prototype which can work as a test bed for new ways to interact with multi-touch. The prototype should be built on open standards, and use open source software as far it is possible. By using this approach, it will be possible for people to create their own similar table without worrying about licensing. During last fall the focus was on multi-touch technologies, and a comparison of these. This thesis will shift the focus more over to the software part of multi-touch, including gesture recognition and object recognition. A complete gesture recognition system will be implemented, and different gesture recognition algorithms will be tested in a multi-touch environment.

I will in this introduction explore the real world utility of multi-touch to familiarize the reader with the opportunities that multi-touch offer. With the opportunities in mind, it will be easier to see the benefits when reading about technical aspects of multi-touch later in this thesis. Before we dive into the technical details, the motivation behind multi-touch, and some multi-touch application design guidelines will also be presented.

1.1 Motivation

To be able to talk about motivation, it is important to look at the goals of the field of artificial intelligence, especially the subfield *Intelligent User Interfaces*(IUI). According to [17], Intelligent User Interfaces are human-machine interfaces that aim to improve the efficiency, effectiveness, and naturalness of human-machine interaction by representing, reasoning, and acting on models of the user, domain, task, discourse, and media (graphics, natural language, gesture). As a consequence, this interdisciplinary area draws upon research in, and lies the intersection of, human-computer interaction, ergonomics, cognitive science, and artificial intelligence and its subareas. In example, vision, speech and language processing, knowledge representation and reasoning, machine learning/knowledge discovery, planning and agent modeling, user and discourse modeling.

This thesis documents the completion of a human-machine interface, that tries to make interaction more efficient and natural using multi-touch with gesture recognition and object recognition. Important goals is to make the user interface immediately accessible and transparent to the user. Although this project is only a start, it serves as a stepping stone in the right direction for a more complex solution using the prototype built. In figure 1.1 a high-level architecture of the components of a intelligent user interface is shown. The intelligence in IUIs that distinguishes them from traditional interfaces is indicated in bold in figure 1.1. It includes mechanisms that perform automated media analysis, design, and interaction management. The highlighted area is where this project contribute. Although this illustration is old, it is still fully valid with some small adjustments, like that we now have more input and output mediums.

The motivation for this project is more effective, efficient and natural interfaces to support access to information, applications and people. One day multi-touch, and other intelligent user interfaces technologies will be available for everyone, integrated into your home. My goal is not met before we use this technology every day, without noticing it. First then can we say we have accomplished something huge.

1.2 Multi-touch application design considerations

Even though we always try to think outside the box to realize a new concept to its full potential, doing so is challenging. But of course we should ask ourselves, do we need a multi-touch enabled device for this application, or is it just a cool gadget? To utilize the full potential of multi-touch we have to consider some design guidelines so that we don't walk into a trap and use multi-touch in an application that would have been better off in the standard WIMP domain.

It is of course hard for a developer to start developing applications without the normal behavior, therefore the CEO at *Infusion*, Greg Brill [5] made some interesting design guidelines for application development with Microsoft Surface, Microsoft's multi-touch enabled table. This thesis present an adaption loosely based

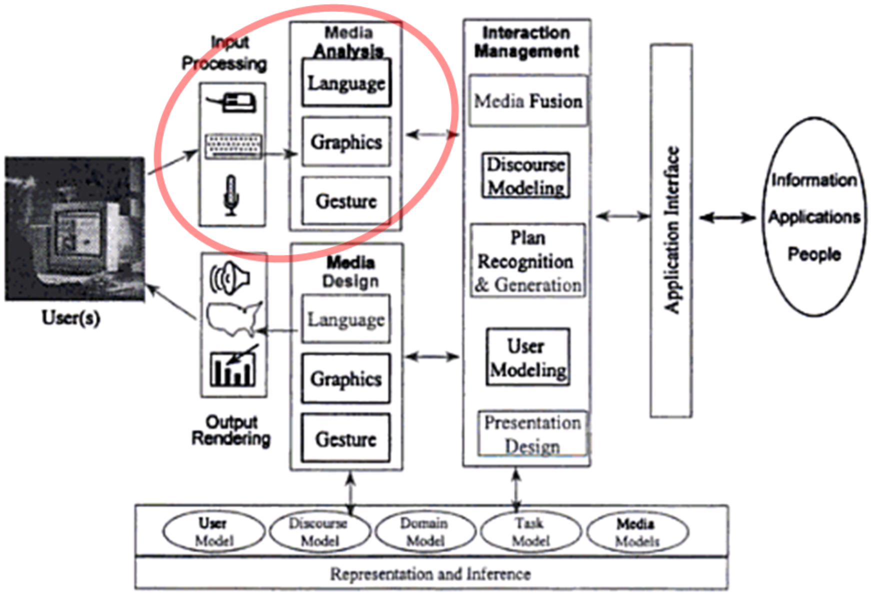


Figure 1.1: High-level architecture of the Intelligent User Interface. Illustration courtesy of [17]

on his ideas with focus on implementation of multi-touch applications regardless of hardware used.

You have to ask yourself three questions to find out if an application should have multi-touch, or be implemented as a regular WIMP application.

Does my application require multi-touch? If the application can work well on either the web or a traditional computer, you most likely do not need the capabilities of multi-touch. According to Brill you should ask yourself if your design incorporates the following:

- A. Is the application collaborative? Will anyone come in and use it from any direction?
- B. Will multiple people be realistically involved and work together on it?
- C. Is multi-touch genuinely required and helpful?
- D. Could the same mechanics work on a single-touch or mouse-driven system?

If the answer to all A, B and C is no, and the answer to D is yes, then Brill advises not to use a multi-touch application approach, because it is likely that your application will fall flat and fail to impress in the way you had imagined.

Is there ”magic” to the application? If the application can make people smile when interacting with the application or it looks like they are playing with it, you have an application that might fit for multi-touch. If it looks like work, it often is, and then the application does not belong in a multi-touch environment. But of course there are always exceptions, and many work situations may benefit from the power of multi-touch.

Is it just ”cool” or is there a clear ROI? It is still important to have in mind that development costs to create, deploy and support the application are higher than for regular applications at the moment. If it is not evident how multi-touch may be profitable, and you cannot articulate a metric or mechanism to show a return, your application will not be successful.

1.3 Real world utility

Despite many of the warnings by Greg Brill we have today already seen some great ideas connected to usage of multi-touch. Microsoft has proved that there are many industries that really can benefit from this technology[18]. Especially financial services, health care, service, retail and public sector may be able to have a huge benefit from interactive multi-touch as showed by Microsoft:

Within financial services such as banks, multi-touch applications running on interactive tables, such as *Microsoft Surface* can help replace complicated product descriptions with a dynamic and easy-to-use interface that is empowering and engaging. For example, an employee and a customer may use the interactive table as their collaborative meeting space, reviewing portfolio options and creating a more personal and informed customer experience that drives higher customer loyalty, while improving brand image.

Within health care such as hospitals, multi-touch can help improve the doctor and patient relationship. For example, patients can receive a better experience by allowing self check-ins or researching and reviewing treatment options in collaboration with a doctor like reviewing X-rays or 3D MR scans. Doctors can explain complicated medical procedures, so patients will be more aware of the risks. For clinicians, an interactive multi-touch wall could be used as a smart white board for the emergency room or medical center, helping teams collaborate on treatment options.

When it comes to service an interactive table could offer entirely new ways to inform and entertain customers, helping boost revenue without incurring higher personnel costs. For example, loyalty cards can be placed on table to trigger purchases of a customer’s favorite drink or meal. Or companies could extend their digital marketing campaigns to include an on site presence, with interactive content, games, and virtual tours.

In the retail business multi-touch application installations can connect customers with more and richer product information and create collaborative sales engagements within retail outlets, connecting the physical store, sales staff and digital content. For example, a multi-touch table recognizes mobile devices, allowing customers to download product information, compare products, cross-sell related items, and select add-on features.

Within the public sector, multi-touch installations may create a seamless way of connecting people with important and complex data. For example, security or emergency agencies can create a collaborative digital command center for planning and orchestrating large scale events, with agency personnel working together with real time mapping and location data.

Education is also an area where multi-touch can utilize its true potential. By using multi-touch enabled devices in classrooms, children can interact with education material in a whole new way than before. Learning can be organized as a multi-user game where many people can interact simultaneously.

The game industry may benefit from multi-touch, especially games where collaboration is important. You will be able to control more units at once, or more people can control different groups of units at the same time collaborating on the same multi-touch table.

My personal favorite application of multi-touch is in the living room. By replacing the living room table with a huge interactive screen, many possibilities appear. The table can be used like a ”command central” for the home, where you can control lighting, temperature, blinds, alarms, monitor your electricity usage along with other systems you may install in your home. Through the table you get all the information you need about your daily schedule such as meetings and other appointments. Even looking out the window becomes superfluous, because you will have the entire weather forecast directly on the table.

When you just need to relax, the table becomes your personal media station where you can see what is on TV, or flick through your movie collection. When you have friends visiting, several people can simultaneously view and interact with the photos from the last party or vacation. A possible future scenario is illustrated in figure 1.2.

What has been mentioned here is just the beginning, as people start opening their eyes for this technology new areas of use will be discovered. The possibilities for new interactions and ways to interpret multiple touches are limited only by the creativity and imagination of the programmer. This allows for some very natural and intuitive applications to be programmed.

Figure 1.2: Picture illustrating a future multi-touch scenario. Illustration modified from its original source www.aboutbar.com

Chapter 2

Multi-touch hardware solutions

Before we dive into the details, this chapter will give a short overview of different multi-touch technologies which exist today. Since this project focus on camera-based multi-touch, solutions based on this technology will be presented most thoroughly.

Three primary types of touch technologies exist: resistive touch, capacitative touch and camera-based touch, which are all based on different principles.

Resistive touch technology has been the most common technology used in consumer electronics up until now. Resistive technology is cheap, and thus readily available for high volume applications. Resistive screens require a degree of force when used, and are technically not touch screens. Resistive technology is often used in everything from hand held devices to cash registers in stores.

Capacitive touch, in contrast to resistive touch, does not require the user to apply force when using a touch screen. As described in [26] capacitive sensors are non-contact devices capable of high-resolution measurement of the position and/or change of position of any conductive target. Capacitive sensors use the electrical property of ”capacitance” to perform measurements. Capacitive technology is often used in high end mobile phones, and products in need of robustness.

For a complete and more thorough explanation of resistive and capacitive multi-touch, the reader is referred to read Appendix A. In the end of the Appendix A there is also a small overview of other touch technologies currently not able to sense multiple touches. The development of new technologies, and new techniques are improving every day, so it may not be long before some of these technologies will be capable of multi-touch as well.

2.1 Camera-based multi-touch

Camera-based multi-touch is the technology used building the prototype in this project. There exist five different techniques to build a camera-based multi-touch system. This section will focus on the common features within these techniques, and in depth present the current technique used in the prototype. For a complete overview of the five techniques, the reader is referred to Appendix B.

All camera-based multi-touch devices utilize one or more cameras to *sense* touches. Because of this, solutions using camera-based multi-touch have to be larger than devices utilizing resistive or capacitive touch. This makes the technology unsuitable for handheld devices. To unlock the true potential of camera-based technology, a device is often either a multi-touch table, or a multi-touch wall. In this section a table is used as the medium, unless otherwise stated.

Depending on the technique used, the camera can be either behind, or in front of the surface you want to sense touches on. To obtain maximum view of the touch surface, the camera should be placed behind the surface. This eliminates the problem with objects and hands shadowing the view of the camera.

If the goal is to make an interactive multi-touch screen, an image source is also needed. For this purpose, either a projector or a LCD screen can be used. If you choose to use a projector, it is the most appropriate to build the projector inside the wall or table. The projector will then light up the screen surface from behind, in the same way as back projection TVs work. This mean that the projection surface must be some kind of diffuser, a semi transparent material, to stop the light from shining right through it.

Camera-based multi-touch rely on a camera *seeing* your fingers through a surface, and you may want to minimize the noise from the environment where the table is placed. To optimally do this, we have to make the camera less sensitive to ambient light. Instead of relying on ambient light that changes throughout the day, a smart multi-touch build utilize the infrared spectrum. Further details are explored after the sensing method is explained.

The camera can sense you finger in two ways, either by *seeing* the shadow of your finger, or seeing your finger light up on top of the surface. This depends on where the infrared light source is placed. If your light source is placed away from the table, your finger will make a shadow on the point you touch. This happens because you block the light from entering through the surface with your finger, and that spot turns dark. The rest of the table surface will be flooded in light, and therefore no touches will be registered here. Multi-touch is possible by making several dark spots. This can been seen in illustration 2.1.

If the infrared light source is placed inside the table, you will rely on reflection instead of a shadow. The light source inside the table will send infrared light out

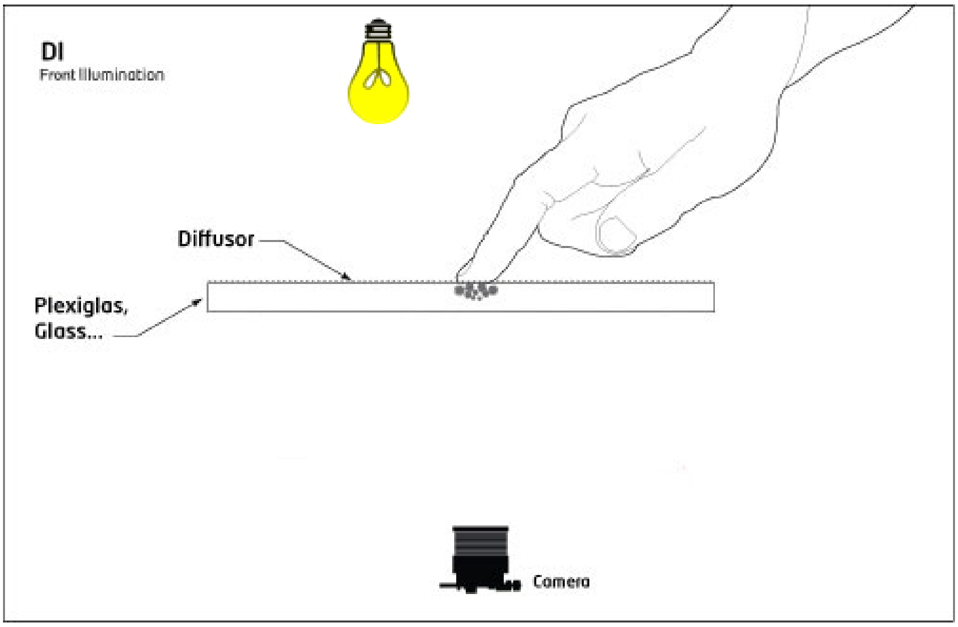


Figure 2.1: Front Diffused Illumination. Original image courtesy of Tim Roth (Image has been modified for this report).

of the box, and by using your finger you can stop the light from escaping. The points you touch will light up, and the camera is able to see your touches. There exist many different methods for touch sensing using light reflection. See figure 2.2 for an illustration of this concept. The method explained here is the one used in the techniques rear diffused illumination and diffused surface illumination. For an overview of the other methods please read Appendix B.

Since the light source is infrared, we have to prepare the camera to *see* as much infrared light as possible, and at the same time *see* as little of the ambient light as possible. To do this the camera is optimally fitted with a bandpass filter with the same wavelength as the infrared light source. Ideally the camera will only *see* the light transmitted by the infrared light source, and nothing else. LEDs are most commonly used as light source, as these are both efficient and long lasting. See illustration 2.3 for the wavelength diagram of a filter used for 850nm infrared light.

For low cost solutions, Unibrain Fire-i camera, Phillips SPC900-NC, or Sony PlayStation 3 camera [6] is often used, as they are cheap and powerful. All the cameras mentioned are capable of a high frame rate, which is considered the most important aspect when choosing a camera. By having a high frame rate, we will be able to process many images in a short time, and the screen will feel more re-

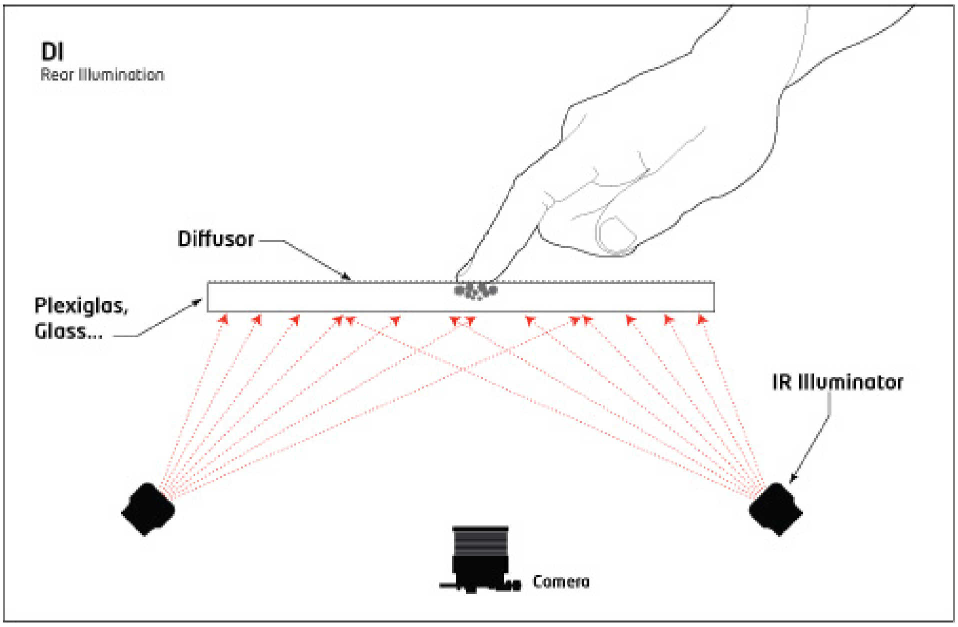


Figure 2.2: Rear Diffused Illumination. Original image courtesy of Tim Roth (Image has been modified for this report).

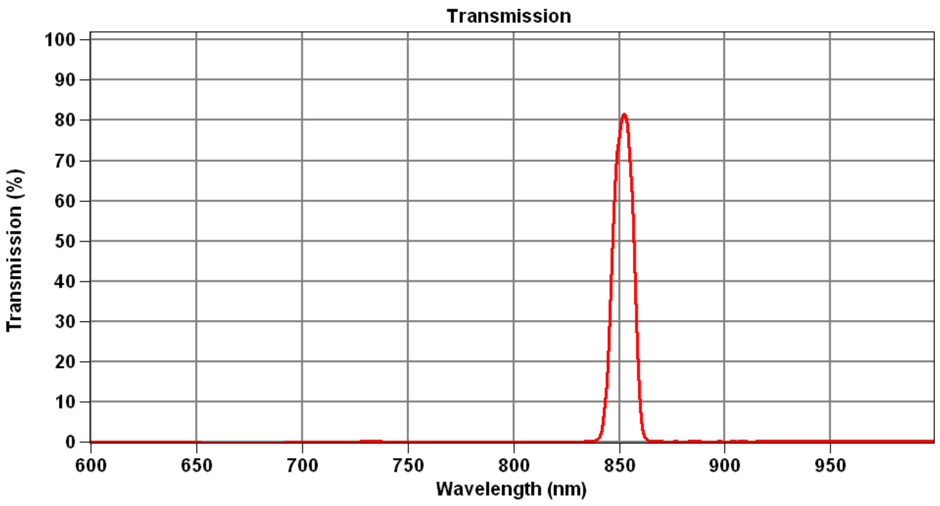


Figure 2.3: Wavelength diagram for 850nm infrared bandpass filter.

sponsive during interaction. Images from the camera are continuously streamed to a computer, where advanced image processing algorithms calculate the [x, y] coordinates of the touched points in the image. Techniques and software for doing this will be explained in chapter 3.

The next section will present the technology currently implemented in the prototype. The last section will give an overview of existing multi-touch solutions that are already either commercially available or will be in near future.

2.2 Diffused Surface Illumination

Diffused Surface Illumination (DSI) was invented by *Tim Roth* as a response to a problem with the standard Diffused Illumination(DI) method. In a Rear DI setup the infrared light source is placed behind the projection surface pointing upward. Usually a cluster of infrared LEDs is used. This illuminates the surface from the non-interactive side, and by placing fingers (or objects) on top of the surface, they will reflect infrared light down towards the camera. Depending on the size and configuration of the table, it can be hard to achieve a uniform distribution of light across the surface.

By using DSI instead of DI, the problem with non-uniform distribution of infrared light is eliminated. In [30] *Tim Roth* presents the DSI technique in detail. Instead of using clusters of LEDs as an infrared source, a special acrylic glass is used to distribute the light evenly across the surface.

This special glass is called PLEXIGLAS EndLighten, and contains thousands of micro mirrors. When you shine IR light into the edges of this material, the light gets redirected and spread to the surface of the glass. This technique will behave in the same way as DI when a finger touches the surface. Depending of the properties of the diffuser in regard to how much infrared light is transmitted and diffused, objects close to or touching the surface will create bright spots in the image captured by the camera. Se figure 2.4 for an illustration of the concept.

Compared to the other camera-based techniques, DSI have many advantages. Since this is an experimental prototype, the table was constructed with the possibility of changing sensing technique. By just swapping the PLEXIGLAS EndLighten sheet, and replacing it with a regular acrylic glass you have converted the table to use the FTIR technique(Appendix B).

In contrast to DI, DSI does not require a closed box, and can be a touch wall or table without walls if this is preferred.

Object recognition is also made possible as objects and hands do not need to touch the surface. Dependent on the calibration, DSI can be adjusted to *see* hovering of objects or hands above the table. And since the infrared light is evenly distributed across the surface, we avoid hotspots. As a bonus, the whole setup con-

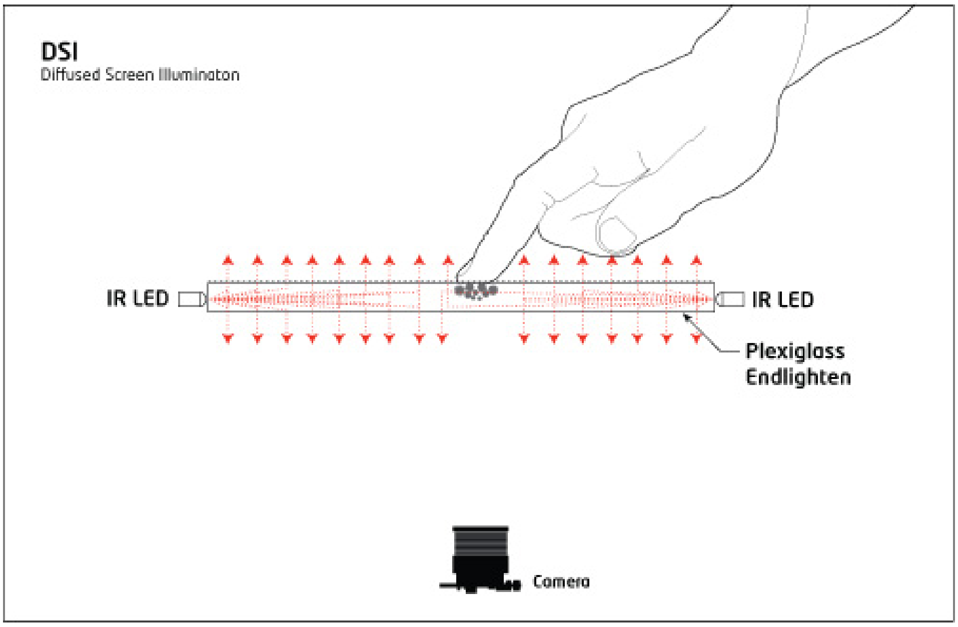


Figure 2.4: Diffused Surface Illumination. Original image courtesy of Tim Roth (Image has been modified for this report).

sist of fewer components than other setups, making it easier to build.

On the downside, Endlighten Acrylic costs more than regular acrylic. You may also get less contrast compared to normal DI set-ups as the surface material also redirects the IR toward the camera, and not only upwards which would have been ideal.

2.3 Existing solutions

Since the start of this project, new companies has joined the multi-touch race. Wealthy, resourceful companies are exploring the multi-touch market, seeing potential profits and markets. This results in many companies building their own prototype of either multi-touch tables or walls. The technology continuous to evolve as more money goes into research. Although many companies have prototypes, few are actually offering commercial solutions. At least two companies offers a complete camera-based solution.

Microsoft offers a camera-based multi-touch solution for companies as well as for personal use. Their multi-touch table *Microsoft Surface* is sold in two editions, one for developers, and one for regular customers. This approach demonstrates the early stage these multi-touch tables are in. The *Microsoft Surface* is under continuous development, so if you buy a table today, next month there will be a new version with some modifications and a new version number. There is an image of the table in figure 2.5.

Another newcomer to the game is a company called Evoluce[1]. Evoluce offer a complete multi-touch solution to the industry called Evoluce ONE[2]. Evoluce ONE was released to the market in middle of May 2010, and has specifications far beyond what Microsoft has in their Surface. Figure 2.6 show the table in action.

Figure 2.5: Microsoft Surface. Microsofts interactive multi-touch table. Image courtesy of Microsoft.

In table 2.1 some of the most important features and properties for the *Microsoft Surface* and *Evoluce ONE* is listed. For comparison I have included the specifications [18] of my own prototype(chapter 5).

From whats shown in table 2.1, Microsoft focuses on stability rather than cutting edge technology. Microsoft Surface is based on the Windows Vista SP1 operating system and use camera and image recognition in the infrared spectrum to recognize various objects. This input is then processed by the computer, and the resulting interaction is displayed using rear projection. So it sees what it interacts with, whether it is fingers, hands, real-world objects, devices, or tags.

In contrast to *Microsoft Surface*, *Evoluce ONE* present cutting edge technology.

Figure 2.6: Evoluce ONE. A cutting edge multi-touch table. Image courtesy of Evoluce.

Property	**Microsoft Surface**	**My prototype**	**Evoluce ONE**
Physical Size	108 x 69 x 54 cm	99 x 64 x 90 cm	115 x 70 x 50 cm
Screen size	30”	37”	47”
Screen format	4:3	16:9	16:9
Resolution	1024x768	1280x720	1920x1080
Image source	DLP projector	LCD projector	LCD
MT technology	DI	DSI	ITSO
Price	150 000 NOK	30 000 NOK	80 000 NOK

Table 2.1: Comparison of the specifications between Microsoft Surface, Evoluce ONE and my own multi-touch prototype.

Evoluce ONE uses an LCD-screen instead of a projector as the image source, and is thus able to produce a much more dense image with full HD resolution. The touch sensing technology used is something *Evoluce* is calling *Evoluce Integrated Through Screen Optics* (ITSO). The ITSO Sensing Technology, developed by Evoluce, recognizes an unlimited number of simultaneous screen inputs. Whether they come from touches or pens, you can write or draw naturally at the same time on the screen, which is pleasant to touch as well as scratch-resistant. ITSO technology is camera-based, and enables object recognition through tags and markers.

To be able to push such multi-touch tables out to the general consumer the price has to be right, and this is the point where Microsoft fails. It is clear that Microsoft does not intend for the general consumer to buy this table with a price tag of 150 000 NOK. Here Evoluce has a better price, by offering better technology for almost the half price of Microsoft, but 80 000 NOK is still too much to pay. Since the technology is under continuous development, the hardware you buy may be outdated tomorrow. In contrast the prototype I have built costs 30 000 NOK, and that includes equal or better hardware components than Microsoft.

Although Microsoft and Evoluce charge a lot of money for their solutions, it is in a way understandable. Both companies have invested a lot of money in developing their own proprietary hardware and software. If these companies could look towards the open source community instead of developing proprietary software, both the community and the companies would benefit.

By looking at the huge interest for multi-touch, it would be fair to expect that more companies will develop and sell camera-based multi-touch in the future. This is the only multi-touch technology that allows true object recognition. In the meantime, capacitative and resistive technology will continue to dominate the market for touch enabled devices.

Chapter 3

Multi-touch development environment

Besides hardware, a multi-touch solution would also need advanced software to correctly interpret what the camera sees on top of the table surface. To create a great multi-touch experience, fast software is needed. When a user touches a touch device, the person expects the device to respond instantly. The responsiveness of the device depends on the time it needs to process the user input, and presents the user with a result on the display.

The software is used to aid the process of transforming raw camera input to meaningful input for applications. To do this, the software package has to capture an image from the camera, preprocess it, recognize the objects in the image, track objects by comparing objects recognized in earlier captured images, and transport the interpretation of the image over to an application capable of understanding this information.

When the information reaches its destination application, this application has to interpret it based on its context. Many common programming languages have a multi-touch framework capable of understanding such information, and whether this information should be interpreted as a gesture or a single touch event is up to the application developer to decide.

While developing software it is important to have some limitations in mind. A regular camera will be able to capture between 30 and 120 frames per second. To be able to interact smoothly with an application, a minimum of 30 frames per second is required. This in turn means that the processing software cannot use more than 1/30 of a second from the raw input is captured till the user application has received its input. A combination of smart algorithms implemented in software, and fast hardware helps to minimize the latency and increase the responsiveness of the device.

This chapter will further elaborate on how to implement and configure a full software environment for multi-touch. Starting with the preprocessing of camera input, and all the way to recognizing objects. Figure 3.6 provides a quick overview of how the system is assembled. This architectural schematic will be further elaborated and explained in section 3.3.

3.1 Preprocessing

The first step to successfully utilize multi-touch input is to preprocess the raw image delivered from the camera to help the recognition software to make consistent and accurate recognitions. It is mainly two things that influence the input image, the camera itself and the environment the camera is operating in.

To correctly interpret the location of a touch, we have to know something about how the lens of the camera works. Since many multi-touch builds require a short distance from the camera to the screen, wide angle lenses are used. These lenses have the side effect of producing a barrel distortion on the image. See figure 3.1, for an example of such distortion. To cope with this, either of two methods are used: You can model the properties for the lens, and create a mapping function to map the table surface coordinates to the ones detected by the camera.

You can calibrate the screen by mapping physical touches on known points on the screen to the ones detected by the camera. By having these points in a matrix, you can use these values to calculate the dynamic offset for each touch point. More touch points used in the calibration equals a more accurate model of the lens distortion. For an example of such calibrations see figure 3.2.

When the distortion of the camera lens is taken care of, the other issue is the environment the multi-touch will operate in. Each image captured from the camera might not only include the contact points of fingers and objects, but also a (static) background. In most systems, a robust background-subtraction algorithm first needs to preprocess each image. This assures that static or background objects in the images are removed. Since ambient light in the environment may change frequently, adaptive models, such as Gaussian mixture models of the background has been developed to intelligently adapt to non-uniform, dynamic backgrounds.

However, with the current infrared (IR) approaches in multi-touch hardware, an adaptive background model turns out to be overkill. Due to the fact that the captured images filter out (non-infrared) light, much of the background is removed by the hardware. Given these IR images, it is often sufficient to simply capture a single static background image to remove nearly all of the ambient light. This background image is then subtracted from all subsequent frames.

When we have a correct representation of what we are interested in, we have to define a threshold for the image, so we can separate between hovering objects,

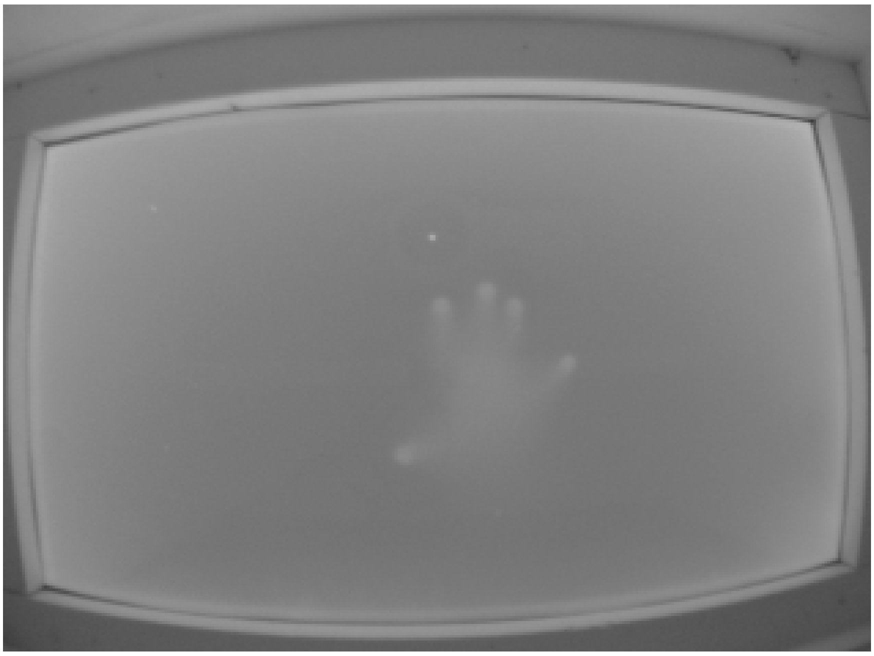


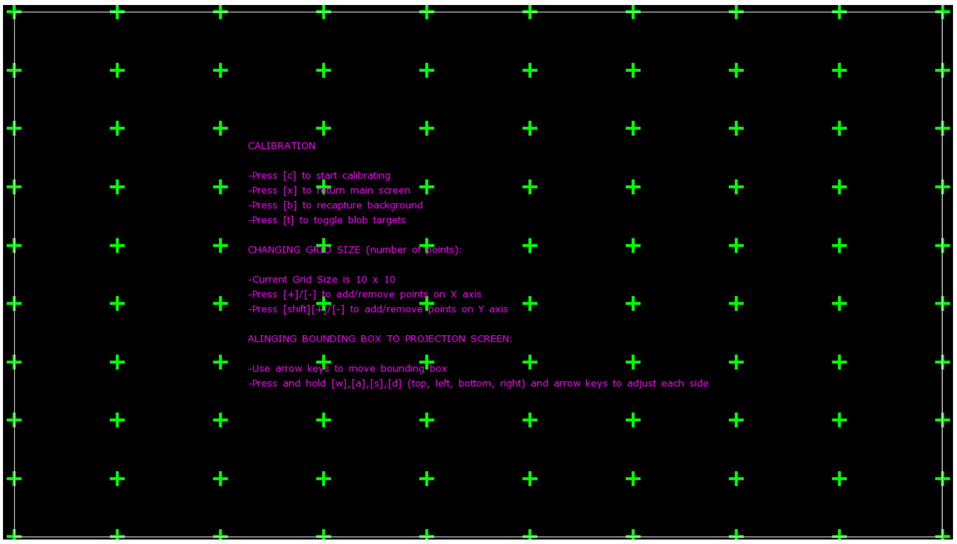
Figure 3.1: Image illustrating how the lens distort the image with barrel distortion. Image is take from the live input stream from the camera.

and actual touches. This can be done by applying a high pass filter, which only lets stronger blobs through. If the image is weak with low contrast, an amplifying filter can be applied to enhance the contrast. As a result the processed frame only shows white contours (blobs) which are the points of interest. See figure 3.3 for an example of how the image is enhanced during preprocessing. The preprocessing stage is often the most time consuming and computationally heaviest.

3.2 Tracking

Tracking is very important to multi-touch technology. It is what enables multiple fingers to perform various actions without interrupting each other.

The task of tracking consists of reliably being able to re-identify a given object for a series of video frames containing that object. After preprocessing the image frames from the camera, all that remains are the foreground objects. To be able to track the objects, they needs to first be detected in all the frames and then the data



```
CALIBRATION

-Press [c] to start calibrating
-Press [x] to return main screen
-Press [b] to recapture background
-Press [t] to toggle blob targets

CHANGING GRID SIZE (number of points):

-Current Grid Size is 10 x 10
-Press [+]/[-] to add/remove points on X axis
-Press [shift][+]/[-] to add/remove points on Y axis

ALINGING BOUNDING BOX TO PROJECTION SCREEN:

-Use arrow keys to move bounding box
-Press and hold [w],[a],[s],[d] (top, left, bottom, right) and arrow keys to adjust each side
```

Figure 3.2: Screen shot of the calibration process in the tracker software *Community Core Vision*. The user is required to calibrate each point separately.

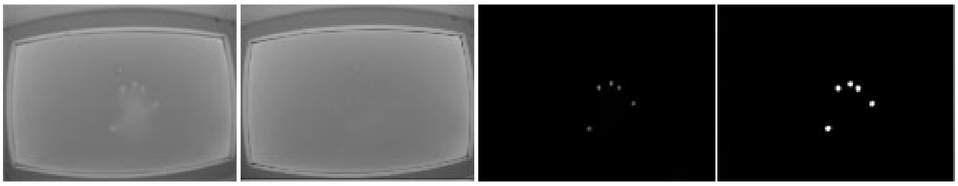


Figure 3.3: A series of images show how the image enhanced during the preprocessing. First image show the raw input from camera. Second image show the captured static background to be subtracted from the raw image. Third image show the subtracted image when a high pass filter has processed the image. Last image show the image after it has been amplified. The last image is the image fed to the tracking software.

must be associated between image frames in order to identify a recognized object. These objects are often identified with their centroids, which are points tracked from frame to frame. Given an extracted centroid, the tracking algorithm predicts where that point should be located in the next frame.

The most common model for the tracking problem is the generative model, which is the basis of popular solutions such as the Kalman and particle filters. The Kalman filter is an efficient recursive filter which estimates the state of a linear dynamic system from a series of noisy measurements. This means that only the recognized object from the previous image frame and the current image frame are required to compute the estimate for the current state (position) of the tracked object. For each state prediction, data association is performed to connect tracking predictions with object observations. Tracks that have no nearby observation are considered to have died, and observations that have no nearby tracks are considered to be a new object to be tracked.

Because the camera for multi-touch capture image frames with infrared light, clear images with little noise is obtained after preprocessing, and an approach using Kalman filter may not be necessary. In most situations it is more than sufficient to perform data association by looking for the closest matching object between two frames with a k-Nearest Neighbors approach. Because the camera captures images with a fixed frame rate one can estimate how far a person may be able to move its finger during the period of 1/(frame rate), and use this as an estimator on where the finger may be in the next frame. With k-Nearest Neighbors, a data point is compared with the set of 'k' points closest to it and they vote for the label assigned to the point. Using this approach, it is possible to reliably track blobs identified in the images through frames.

The most commonly used tracking software is *OpenCV*[24], an open source Computer Vision library based on the method explained here. This library is used in many of the most known tracking frameworks like *Touchlib*[34] and *Community Core Vision*[33], and provides straightforward manipulation of images and videos. The framework track detected blobs in real-time with high confidence. The information pertaining to the tracked blobs (position, ID, area, etc.) can then be transmitted as events which are identified when new blobs have been detected, when a blob 'dies', and when a blob has moved. A screen shot of *Community Core Vision* is shown in figure 3.4.

3.3 Communication protocols

When we have identified the blobs during tracking we need to distribute this information out to the applications in need of input. TUIO (Tangible User Interface Protocol) has become the industry standard for sending tracked blob data.

According to [15] TUIO is an open framework that defines a common protocol and API for tangible multi-touch surfaces. TUIO is a simple protocol designed specifically to meet the requirements of table-top tangible user interfaces. The TUIO protocol allows the transmission of an abstract description of interactive surfaces, including touch events and tangible object states. This protocol encodes control data from a tracker application and sends it to any client application capable of decoding the protocol.

The TUIO protocol has been implemented using OpenSound Control, an

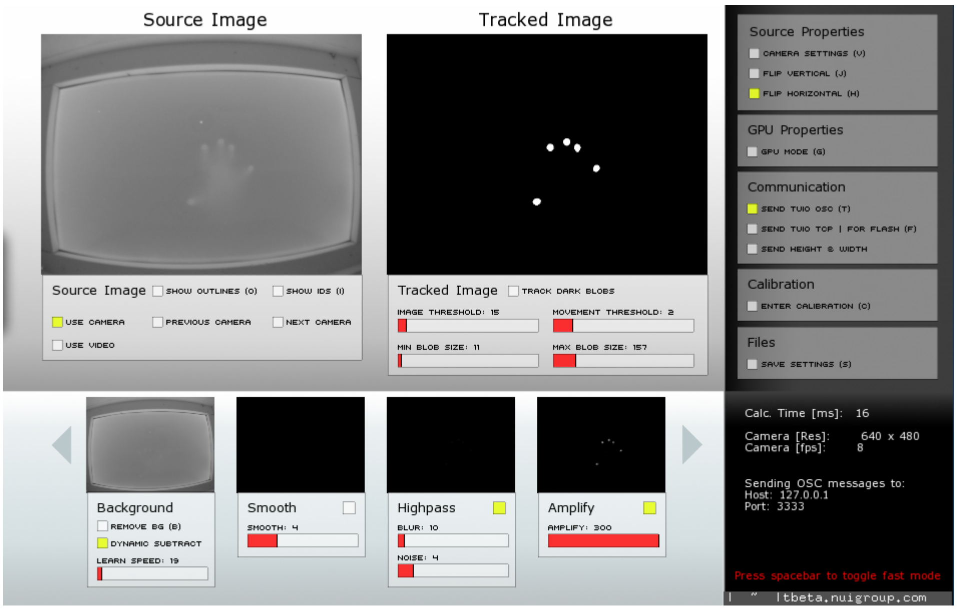


IMAGE THRESHOLD: 15
MOVEMENT THRESHOLD: 2
MIN BLOB SIZE: 11
MAX BLOB SIZE: 157
LEARN SPEED: 19
SMOOTH: 4
BLUR: 10
NOISE: 4
AMPLIFY: 300

```
Calc. Time [ms]:  16

Camera [Res]:     640 x 480
Camera [fps]:     8

Sending OSC messages to:
Host: 127.0.0.1
Port: 3333

Press spacebar to toggle fast mode
tbeta.nuigroup.com
```

Figure 3.4: Community Core Vision user interface. The raw source image is the large image displayed to the left, and the large image to the right is the image that the tracking is working on. Below there is controls for tweaking the filter settings. The GUI is very self explaining.

emerging standard for interactive environments not only limited to musical instrument control, and is therefore usable on any platform supporting this protocol. Figure 3.5 shows an illustration where this protocol is implemented.

The TUIO protocol defines two main classes of messages: *set messages* and *alive messages*. Set messages are used to communicate information about an objects state such as position, orientation, and other recognized states. Alive messages indicate the current set of objects present on the surface using a list of unique session IDs.

The message format used by TUIO is based on *profiles*. Different *profiles* have different parameters. For example to send information about a blob, we can encode this using a two dimensional surface object descriptor:

```
/tuio/2Dcur set s x y X Y m
```

In order to provide low latency communication the implementation of the TUIO protocol uses UDP transport. When using UDP the possibility exists that some packets will be lost. Therefore, the implementation of the TUIO protocol includes redundant information to correct possible lost packets, while maintaining an efficient usage of the channel. An alternative TCP connection would assure the

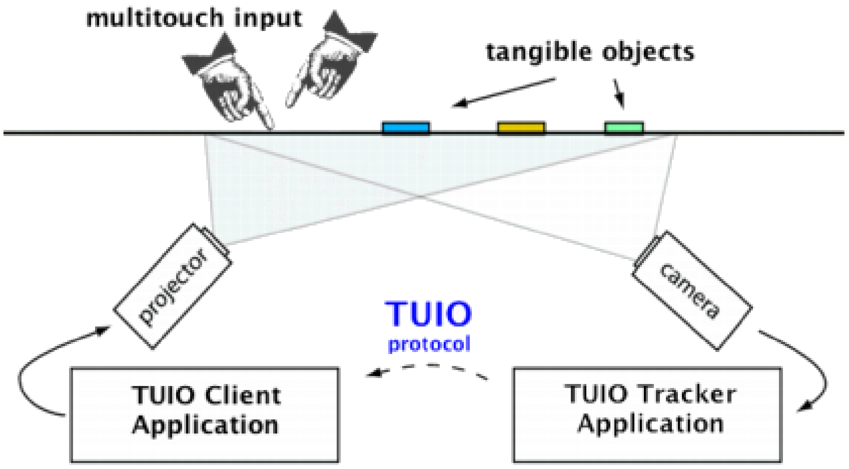


Figure 3.5: Diagram illustrating that the TUIO protocol is operating between the tracker application and the client side applications receiving touch information from the tracker. Diagram courtesy of www.tuio.org.

secure transport but at the cost of higher latency.

When using Flash it is required to convert UDP packages to TCP. This can be done by using a tool called FLOSC which acts as a proxy. When an application uses the OSC protocol, it will only be able to receive events containing properties of the detected blobs.

3.4 Software architecture and application framework

The software architecture used in camera-based multi-touch is presented in figure 3.6. Below is a short description of all the layers, how they connect, and a reference to the section where you can read more about each part.

From the bottom, the camera is picking up the infrared reflections from the person touching the multi-touch surface. These captured images are sent via the camera drivers, USB and FireWire respectively. The tracking software, here represented as Community Core Vision (CCV), works like an umbrella over OpenCV (Open Computer vision) library. Through CCV the user can adjust what type of filters to use for the preprocessing (section 3.1) of images before they get handled by the tracker (section 3.2). It is important to mention that both CCV and OpenCV is community driven open source projects, and a developer stands completely free to edit the source code for their own need.

When the tracker is done processing the input, it sends the tracked objects over

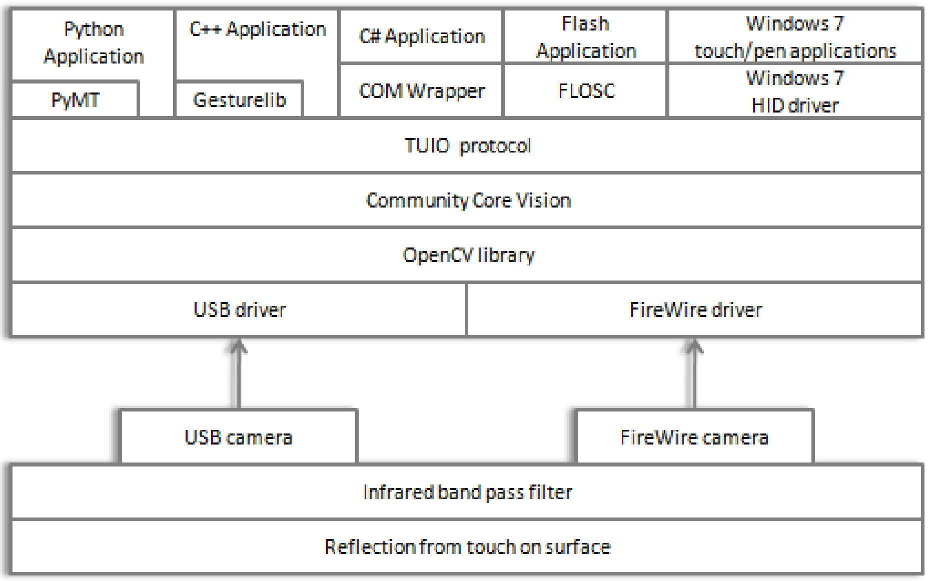


Figure 3.6: Block diagram illustrating the software architecture, and the flow between the different software and hardware components in camera-based multi-touch.

the TUIO protocol (section 3.3) using User Datagram Protocol (UDP). Applications that want to utilize multi-touch have to listen to a specific UDP port, and can choose to use the TUIO messages received.

As there are continuous development of new multi-touch frameworks in several languages, I have chosen to include only some languages in the architectural block diagram, but the basic idea is the same for all additional languages.

PyMT[20] is the community driven open source framework for Python which is capable of interpreting TUIO-messages. This package provides the application developer with the ability to create feature rich applications without concern of the underlying processing of input. PyMT is a huge package which includes everything from graphical multi-touch components to the ability to interpret gestures. It is important to note that gesture recognition is a client side job, and not a task for the tracker. What to do with all the received touches and recognized gestures is up to developer to decide.

For C++ and Flash, it works in almost the same manner as for Python, the only exception is Flash which is not capable of receiving messages over UDP. FLOSC (section 3.3) basically feeds the UDP TUIO-messages into a TCP/IP connection established between the FLOSC bridge and the Flash application.

For Windows 7 the whole setup is somewhat different. Windows 7 is not capable of receiving TUIO-messages by itself, and relies on a proprietary standard for representing multi-touch input. To overcome this issue a human interface device (HID) driver has to be made to manipulate Windows into believing it receives input in its own proprietary format.

3.5 Application development

Multi-touch technology allows users to interact with a computer in a new way. Programming applications and interactions for multi-touch devices is very different from developing classic mouse based interfaces. The major differences lie in having to handle multiple simultaneous touches/cursors. While being able to interpret multiple touches expands the possibilities for potential interactions and interfaces, it also adds an additional, nontrivial layer of complexity to the programming and computational aspect of development.

It quickly becomes clear that interpreting multiple simultaneous touches becomes much more complex than handling a single pointer from for example a mouse. The context of the user interface can change with each individual touch, and subsequent event handlers must be aware of all the interactions happening before making a decision about how to deal with a touch event.

When using a multi-touch device, the type of interaction methods depends on the application. Not many operating systems today support multi-touch natively, and therefore one is required to implement most of the touch interpretation in the program itself. This seems to be changing in newer versions of the most popular operating systems like Windows and Mac OS. For Windows users, Windows 7 was released in October 2009, and contain native multi-touch support. Although the set of programs in Windows 7 utilizing multi-touch still is sparse, you get an idea of what the near future may look like. Now you can use multi-touch in the browser, image editing with paint and the picture viewer.

Ultimately, the possibilities for new interactions and ways to interpret multiple touches are limited only by the creativity and imagination of the programmer. This allows for some very natural and intuitive interactions to be made, but it also means the logic and algorithmic complexity becomes much greater and harder to cope with.

3.6 Tracking with Community Core Vision

Now that we have investigated the theory behind how camera-based multi-touch works, it is time to see how the software is connected on a detailed level. This section is for developers that are interested in multi-touch, either as a contributor to the open source code frameworks, or as a developer interested in utilizing these frameworks to create new and innovative applications. Either way, it is important to know the basic source code, and how this code is connected.

More specific, this section will elucidate, from a developers perspective, all the steps necessary to convert raw camera input, into actions carried out in a multi-touch application. The source code used as example in this section will be taken from the software I used in my prototype described in chapter 5. For preprocessing and tracking the software is *Community Core Vision* utilizing the *Open Computer Vision* library. For representing tracked objects, *Tangible User Interface Objects* will be used. Due to a large code base in CCV, the source code is simplified. Some code has been replaced by ## *explanation* ## to minimize the amount of code written in the thesis. The concept of the code should still be easy interpreted.

To quickly get an overview of how the different components are connected, code sample 3.1 contain a modified main loop from CCV. How many times this main loop is able to run each second decides the processing capacity of the complete tracking process. First the pixels from the camera are acquired, and then converted to a frame CCV is capable to operate on. To recognize the blobs presented in the image, the frame has to be preprocessed with filters as explained in section 3.1. This filtering is done by applying three filters in CCV, the background subtraction (code sample 3.2), the high pass filter (code sample 3.3) and the amplification filter (code sample 3.4).

```
//Simplified main loop for CCV
mainloop(){

  //get pixels from camera
  getCameraPixels()

  //Grab frame from CPU
  grabCameraFrameToCPU()

  //Remove background, apply highPassFilter and ↩
     → amplifyFilter
  filter->applyFilters( processedImg );

  //Find the contours or objects in the filtered input ↩
     → image
  contourFinder.findContours(processedImg,  ( ↩
     → MIN_BLOB_SIZE * 2) + 1, ((camWidth * camHeight) * ↩
     → .4) * (MAX_BLOB_SIZE * .001), maxBlobs, false);

  //Track the found countour (blobs) to see if the match ↩
     → earlier observations
  tracker.track(&contourFinder);

  //Send the observed Blobs to the clients
  myTUIO.sendTUIO(&getBlobs());

}
```

Listing 3.1: Simplified main loop from the CCV source code

```
void CPUImageFilter::sub (cvBackground, cvImageInput ) {
  //Subtract the stored static image from the new inputImage
  cvSub( cvBackground, cvImageInput, cvImageInput );
}
```

Listing 3.2: Source code to subtract the stored static image from the new inputImage in CCV

```
void CPUImageFilter::highpass ( float blur1, float blur2 ) {

  //Blur Original Image
  if(blur1 > 0)
  cvSmooth( cvImage, cvImageTemp, CV_BLUR , (blur1 * 2) + 1);

  //Original Image - Blur Image = Highpass Image
  cvSub( cvImage, cvImageTemp, cvImageTemp );

  //Blur Highpass to remove noise
  if(blur2 > 0)
  cvSmooth( cvImageTemp, cvImageTemp, CV_BLUR , (blur2 * 2) + 1);

    swapTemp();
  flagImageChanged();
}
```

Listing 3.3: Apply highpass filter in CCV

```
void CPUImageFilter::amplify ( CPUImageFilter& mom, float level ) {

  float scalef = level / 128.0f;

  cvMul( mom.getCvImage(), mom.getCvImage(), cvImageTemp, scalef );
  swapTemp();
  flagImageChanged();
}
```

Listing 3.4: Apply amplify filter in CCV

When the preprocessing is done, CCV starts finding contours in the filtered image. This algorithm return only those objects that have a size between the *MIN_BLOB_SIZE* and the *MAX_BLOB_SIZE* defined in the CCV settings.

The recognized contours from each frame propagate to the tracking mechanism to identify relations with earlier recognized contours. A simplified version of this method can be seen in code sample 3.5. This method loops through all the detected contours, and try to find the nearest neighbor to this point among earlier tracked points. For a point to be identified as a neighbor, it has to be within a maximum distance equaling a preset value. If the contour being examined does not have a neighbor, the point is treated as a new touch on the screen. If the contour does have a neighbor the point is treated as a movement from the neighbor to the new contour. If there still exist earlier tracked blobs in the *trackedBlobs*-list when all the new contours have been checked, these *trackedBlobs* are treated as if the finger at this point is removed.

```

void BlobTracker::track(ContourFinder* newBlobs, ↩
  → ContourFinderList* trackedBlobs){

  //For all the blobs currently beeing tracked
  for(int i=0; i<trackedBlobs.size(); i++)
  {
    //Finds nearest neighbor to the newest point detected
    int winner = trackKnn(newBlobs, &(trackedBlobs[i]), ↩
      → 3, 0);

    if(winner==-1) //track has died, mark it for deletion
      ## Mark track for deletion trackedBlobs[i].id ##
    else //still alive, have to update
      ## Update the track associated with trackedBlobs[i ↩
        → ].id ##

    ## If newBlobs is not associated with an existing ↩
      → track, a new track is created. (You have touched ↩
      →  the table, not part of a finger motion) ##

    ## If the blob is associated with an existing track. ↩
      → (You have moved your finger on the table )##

    ## If a previous traced tracks has none of the new blobs in the list near, then the track is terminated (You have removed one of your fingers) ##

    //Call the event handler to notify what has happend. What acutally has happend is encoded in the newBlobs[i]
    TouchEvents.messenge = newBlobs[id]
}
```

Listing 3.5: Simplified source code from tracking routine in CCV

In the end of the tracking method a *TouchEvent message* is sent to inform the state of all the tracked contours. The class *TouchEvent* implements listeners for *TouchDown*, *TouchUp*, *TouchMoved* and *TouchHeld* as can be seen in code sample 3.6.

```
class TouchEvents{

  Blob messenger;

  // All these events is relayed to the TUIO class, so the correct blobs is beening sent to the client software.

  void TouchDown(const void* sender, Blob& eventArgs){
    TouchDown(eventArgs);
  }
  void TouchUp(const void* sender, Blob& eventArgs){
    TouchUp(eventArgs);
  }
  void TouchMoved(const void* sender, Blob& eventArgs){
    TouchMoved(eventArgs);
  }
  void TouchHeld(const void* sender, Blob& eventArgs){
    TouchHeld(eventArgs);
  }

}
```

Listing 3.6: Simplifed verson of the event handler in CCV. This class sends the touches to the TUIO class to be sent to the client software for interpretation

When one of the listeners in *TouchEvent* is triggered, a TUIO message will be sent to the client. It will in practice be sent a continuous stream of TUIO events to the client. The method for packing and sending TUIO in CCV can be seen in code sample 3.7. For each registered blob this method prepares a set message, and in the end sends all the set messages together with an alive message over UDP to the clients listening.

```
map<int, Blob>::iterator this_blob;
for(this_blob = blobs->begin(); this_blob != blobs->end() ; this_blob++)
{
  //Set Message
  ofxOscMessage set;
  set.setAddress("/tuio/2Dcur");
  set.addStringArg("set");
  set.addIntArg(this_blob->second.id); //id
  set.addFloatArg(this_blob->second.centroid.x);  // x
  set.addFloatArg(this_blob->second.centroid.y); // y
  set.addFloatArg(this_blob->second.D.x); //dX
  set.addFloatArg(this_blob->second.D.y); //dY
  set.addFloatArg(this_blob->second.maccel); //m
  if(bHeightWidth){
    set.addFloatArg(this_blob->second.boundingRect.width) ; // wd
    set.addFloatArg(this_blob->second.boundingRect.height );// ht
  }
  b.addMessage( set ); //add message to bundle
}

//Send alive message of all alive IDs
ofxOscMessage alive;
alive.setAddress("/tuio/2Dcur");
alive.addStringArg("alive");

std::map<int, Blob>::iterator this_blobID;
for(this_blobID = blobs->begin(); this_blobID != blobs->end(); this_blobID++)
{
  alive.addIntArg(this_blobID->second.id); //Get list of ALL active IDs
}
//Send fseq message
ofxOscMessage fseq;
fseq.setAddress( "/tuio/2Dcur" );
fseq.addStringArg( "fseq" );
fseq.addIntArg(frameseq);

b.addMessage( alive ); //add message to bundle
b.addMessage( fseq ); //add message to bundle
```

```

TUIOSocket.sendBundle( b ); //send bundle
```

Listing 3.7: Source code for packing and sending TUIO messages in the CCV tracker

3.7 Application development with PyMT

The touch events sent as TUIO by CCV will in this section be used by the client side multi-touch framework *PyMt*. This section present both how *TUIO* is received by PyMT, and small application tutorial to get started developing applications.

PyMT is capable of receiving any message over the *OSC* protocol, but I will here focus on *TUIO*. For the simplicity it will be perfectly safe to treat OSC as just a regular network socket connection capable of receiving text. Code sample 3.8 show how the mechanism for depacking a received */tuio/2Dcur TUIO* message in PyMT is implemented. In this code sample the initialization process is omitted because it is not relevant for understanding the processing of *TUIO*.

```
class Tuio2dObjTouch(Touch):
    '''A 2dObj tuio touch. /tuio/2Dcur  Multiple profiles ↩
       → are available:

        * xy
        * ixyaXYAmr
        * ixyaXYAmrh
    '''
    def __init__(self, device, id, args):
        super(Tuio2dObjTouch, self).__init__(device, id, ↩
            → args)

    def depack(self, args):
        if len(args) < 5:
            self.sx, self.sy = args[0:2]
            self.profile = ('pos', )
        elif len(args) == 9:
            self.fid, self.sx, self.sy, self.a, self.X, ↩
                → self.Y, self.A, self.m, self.r = args ↩
                → [0:9]
            self.profile = ('markerid', 'pos', 'angle', ' ↩
                → mov', 'rot', 'rotacc')
        else:
            self.fid, self.sx, self.sy, self.a, self.X, ↩
                → self.Y, self.A, self.m, self.r, width, ↩
                → height = args[0:11]
            self.profile = ('markerid', 'pos', 'angle', 'mov', 'rot', 'rotacc',
                           'shape')
            if self.shape is None:
                self.shape = TouchShapeRect()
                self.shape.width = width
                self.shape.height = height
        self.sy = 1 - self.sy
        super(Tuio2dObjTouch, self).depack(args)
```

Listing 3.8: Source code for receiving TUIO messages in PyMT

The */tuio/2Dcur TUIO* object has as shown in the code sample three possible interpretations depending on how much information the tracker sends. At most a */tuio/2Dcur* object can include an id, position, angle, movement, rotation and shape information.

For each *TUIO* object received over OSC, the *update()* method is called. See code sample 3.9. The *TUIO* can either be a *alive message*, or a *set message*. A *set message* contain all the current active touch points registered by the tracker, and indicates either a new touch, or a movement of an existing touch. PyMT updates its internal representations of all the touches. An *alive message* contain the ID on all currently alive touches. If the *alive message* received does not include some of the internal represented touches, these are removed, and now considered as deleted touches (user removed finger from the touch surface).

```
def _update(self, dispatch_fn, value):
    oscpath, args, types = value
    command = args[0]

    # verify commands
    if command not in ['alive', 'set']:
        return

    # move or create a new touch
    if command == 'set':
        id = args[1]
        if id not in self.touches[oscpath]:
            # new touch
            touch = TuioTouchProvider.__handlers__[oscpath](self.device, id, args[2:])
            self.touches[oscpath][id] = touch
            dispatch_fn('down', touch)
        else:
            # update a current touch
            touch = self.touches[oscpath][id]
            touch.move(args[2:])
            dispatch_fn('move', touch)

    # alive event, check for deleted touch
    if command == 'alive':
        alives = args[1:]
        to_delete = []
        for id in self.touches[oscpath]:
            if not id in alives:
                # touch up
                touch = self.touches[oscpath][id]
                if not touch in to_delete:
                    to_delete.append(touch)

        for touch in to_delete:
            dispatch_fn('up', touch)
            del self.touches[oscpath][touch.id]
```

Listing 3.9: The update method is called every time a new TUIO is received

We now have the basic *TUIO* interpretation ready, and these are presented as touches with coordinates and other properties in Python. To use this data we have to create an application. For simplicity, a trivial example application will be examined. This example do not utilize the true power of multi-touch, but work as an example of how to use the received *TUIO*.

The example is adapted from the PyMT webpage[20]. The application code is listed in 3.10. This application creates a button and a label on the screen. Every time the button is pressed, a new random number between 1 and 100 is displayed in the label. Because this code example is simple, it is also kind of self explaining. The most interesting part to note is the definition of a *on_press* listener for the button. The button widget is capable of *on_press* (fired when the button is pressed down),*on_release* (fired when the button is released) events. These events correspond to the messages received as *TUIO*.

```
import random
from pymt import *

w = MTWindow()

b = MTButton(label='Hello, World!', pos=(40, 40), size=(200, 200))
w.add_widget(b)

l = MTLabel(label='#', font_size=20, pos=(270, 40))
w.add_widget(l)

```

```
@b.event
def on_press(touch):
    l.label = str(random.randrange(0, 100))

runTouchApp()
```

Listing 3.10: Number generator application implemented in PyMT

Other interesting widgets you can use in PyMt applications is for example the *scatter* widget. This widget support in addition to *on_press* and *on_release* a event called *on_transform*. This event exploits PyMts built in gesture library to interpret multiple touches, then it will be possible to rotate, scale, move or zoom objects.

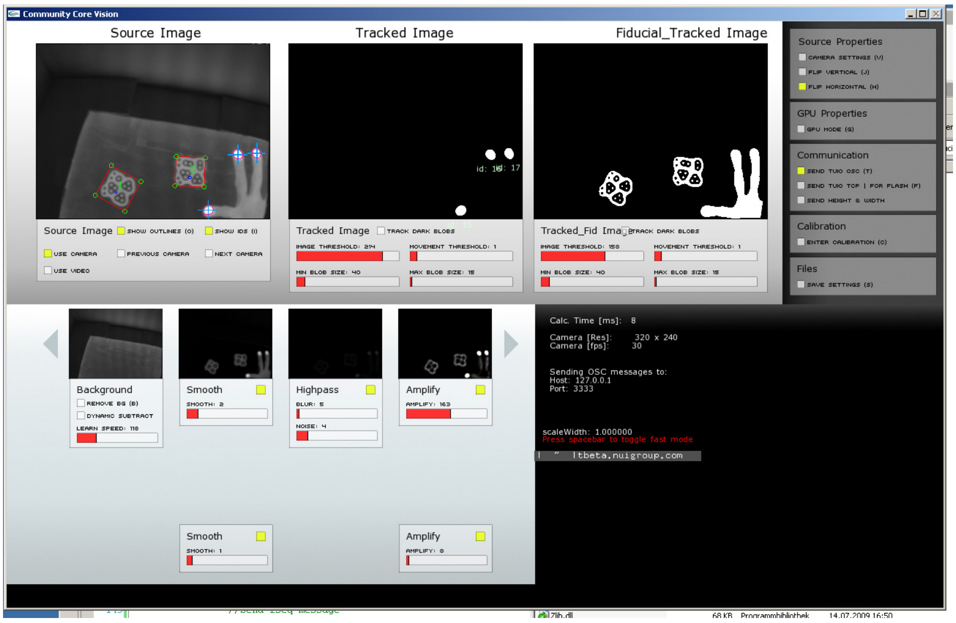
For more information on how to develop multi-touch applications in python, the homepage of PyMT[20] is a good place to start.

3.8 Object recognition

Since this master's thesis is an extension of the multi-touch project I wrote during fall 2009, it is natural that the future improvements stated there are addressed here. Object recognition was one of these future features. Object recognition has now been implemented into the prototype as an experiment, and a showcase of new exiting technology. Although the focus of this project has largely been gesture recognition, the prototype could not be called complete without object recognition.

To narrow the project down, and avoid focusing too much on implementation of object recognition, this project has mainly integrated technology made by other projects into this project. Since reacTIVision [14] already has made a fantastic system for fiducial tracking, I focused on integrating their software into my table instead of implementing my own system. reacTIVision is an open source project which has made a huge contribution to the field of object recognition. Using the word *integration* does not mean that it was an easy job, as it takes a lot of configuration and tweaking to actually make object recognition work properly.

Tracking fiducials instead of shapes opens up a much broader area of use, which mean similar shapes can get different IDs. According to [14] the fiducial tracking system works like this: reacTIVision tracks specially designed fiducial markers in a real time video stream. The source image frame is first converted to a black and white image with an adaptive thresholding algorithm. Then this image is segmented into a tree of alternating black and white regions (region adjacency graph). This graph is then searched for unique left heavy depth sequences, which have been encoded into the fiducial symbol. Finally the found tree sequences are matched to a dictionary to retrieve an unique ID number. The fiducial design allows the efficient calculation of the marker's center point as well as its orientation. OSC messages implementing the TUIO protocol encode the fiducials' presence, location, orientation and identity and transmit this data to the client applications.



```
Calc. Time [ms]:  8

Camera [Res]:     320 x 240
Camera [fps]:     30

Sending OSC messages to:
Host: 127.0.0.1
Port: 3333

scaleWidth: 1.000000
Press spacebar to toggle fast mode
tbeta.nuigroup.com
```

Figure 3.7: Screenshot of a special version of Community Core Vision with the ability to track fiducials.

The actual object tracking is done by CCV, which has been adapted by Stefan Schlupek [6] with a library from reacTIVision. See figure 3.7 for an illustration of this interface compared to figure 3.4. This new add on to CCV makes it possible to adjust filter settings separately based on whether you want to track fiducials or fingers. CCV sends a similar TUIO message like the ones explained in section 3.3. Since the message format used by TUIO is based on *profiles*, object/fiducial have its own *profile* with different parameters. For example to send information about a tracked object, we can encode this using a two dimensional surface object descriptor [15]:

```
/tuio/2Dobj set s i x y a X Y A m r
```

Here the important parameters are X, Y and A. Where X and Y tells the position of the object, and A tells the rotation angle of the object. Read section 3.3 for more information. What types of fiducials CCV is able to track must be encoded in a file called *tree.xml*. This file include an identifier, and a numeric sequence representing all the fiducials the program can recognize. Such sequence can be:

```
w0122222212212121111
```

The file used in my setup is precoded with all the fiducials reacTIVision is capable of recognizing. As an example of different fiducials, see figure 3.8.

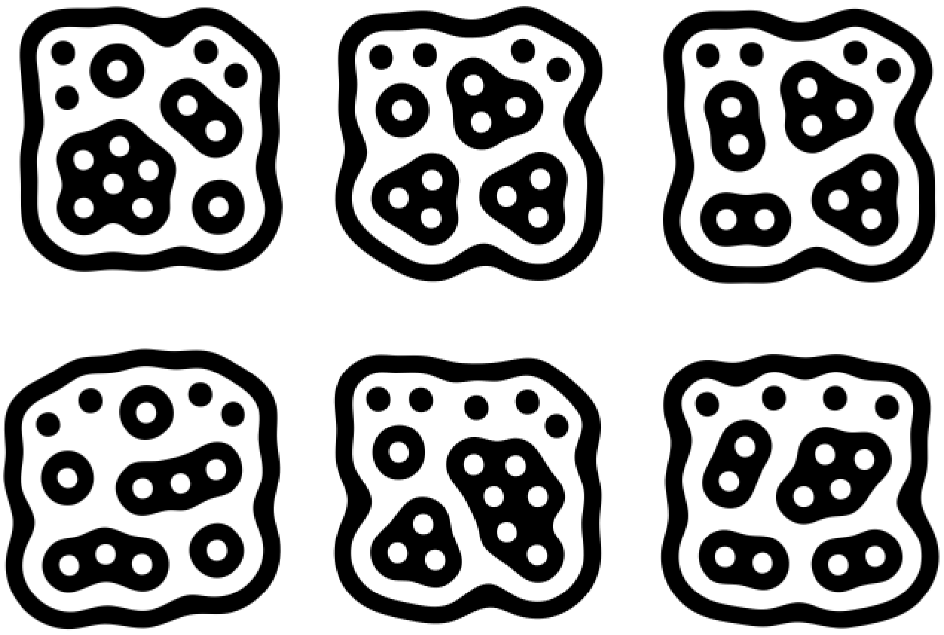


Figure 3.8: Illustration of six different fiducials the table is able to recognize. Image courtesy of reacTIVision.

Obtaining good results for regular finger tracking may be called easy compared to object recognition, where a much better image quality is required from the input camera. It is even preferable to use more than one camera. Since this project is based on only one camera at the moment, this prototype will work best if only finger tracking or object tracking (fiducial tracking) is enabled.

Although fiducial tracking have not been the main focus, it is important to acknowledge the power of object recognition, and it certainly an important path to follow into the future of multi-touch technology.

3.9 Windows 7 integration

To make the software environment as complete as possible, this project has been integrated with native Windows 7 touch. Windows 7 is the first operating system by Microsoft that feature multi-touch support. To be able to use this feature, a conversion between TUIO-commands and native Windows touch events have to be made. A user named Nesher [21] at codeplex.com, has created a human interface device(HID) driver which can do this. Multi-Touch Vista is a user input management layer that handles input from various devices (touchlib, multiple mice, TUIO etc.) and normalizes it against the scale and rotation of the target window.

This driver run as a background service, and continuously translate incoming TUIO-messages to commands Windows 7 understands. By using this driver, it is also possible to emulate Microsoft Surface input, which mean we can utilize the power of running applications originally designed for Microsoft Surface.

Chapter 4

Gesture recognition

An important part combined with building and testing a multi-touch table have been to test different gesture recognition algorithms for multi-touch. To make the system as complete as possible, the implemented gesture recognition methods has been fully integrated into the open source project PyMT [20]. In the end of this project the code will be submitted back to the code repository for PyMT, and possibly be included in future releases. This chapter will give an overview of what systems exists for gesture recognition, and give a thorough introduction to the basics of gesture recognition. This introduction will include theory and implementation details for the gesture recognition implemented. In the end of the chapter a result and discussion section will provide results for testing the efficiency and accuracy of each of the gesture recognition methods.

All of the methods presented in this chapter is either implemented by the author, or already included in the PyMT framework. More specifically, six methods are implemented by the author, and one is already included. This chapter will also include a simple introduction to the theory behind Hidden Markov Models for gesture recognition, although this method has not been implemented. It may sound strange to implement something that already exist in other applications, but many existing solutions for gesture recognition are closed systems and hard to integrate into a programming framework.

By improving an application with gesture recognition, people will be able to interact more naturally with applications, and less training in how to use the program will be required. As the mental model of the designer will be more aligned with the mental model of the user through self explanatory interfaces.

The main goal of gesture recognition is to create a system which can identify specific human gestures and use them to convey information, or for device control. In order to ensure accurate gesture recognition and an intuitive interface, a number of constraints are applied to the model. The multi-touch setup should provide the capability for more than one user to interact with it, either independently or in a cooperative mode. According to [7]it is important that we do not fall into the temptation of making a WIMP interface with multi-touch. We should rather rethink the idea, and use the advantages gestures and multi-touch gives us. Only then will we be able to utilize the true potential of multi-touch enabled devices, such as the table built in this project.

In camera-based multi-touch, gesture recognition is a part of the client side implementation of the software. It is up to each language specific framework to decide how this is done. According to [23] the most common way to implement gestures is through use of different models like Neural Networks, region based, direction based, and Hidden Markov Models. It is therefore natural that this chapter include a description of all these methods.

Before we dive into the details of the different gesture recognition methods, it is necessary with an introduction to gesture recognition. According to [22] there primary exist two groups of gestures, direct and symbolic gestures.

Direct gestures are the oldest group of computer related gestures, and has existed since the first operating systems to include support for a mouse. Direct gesture is as the name suggest a gesture that directly manipulate an object. Moving a window around on the screen with the mouse is for example thought of as a direct gesture. Introduced with multi-touch some modern gestures where invented, although no one actually know who invented them. This include zoom and rotation gestures as illustrated in figure 4.1. By measuring the distance between two fingers, pinch to zoom is often the preferred tool for changing size on objects using multi-touch. This provide great feedback, and is a much more intuitive way of resizing objects than dragging in the corner of an object.

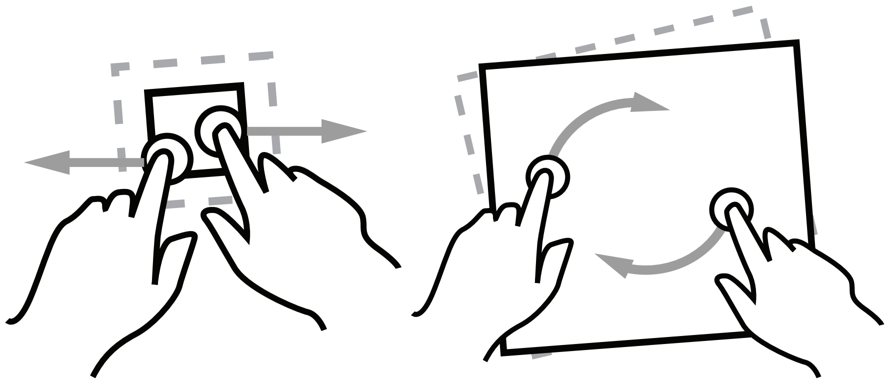


Figure 4.1: Example of direct gesture. Zoom and rotation. Illustrations provided by Gestureworks (www.gestureworks.com).

In contrast to direct gestures, symbolic gesture do not necessarily have any relation with an object. Symbolic gestures are pattern based, and demands much more complex algorithms and processing to recognize. The symbolic gesture pattern is generated by a sequence of touch-points drawn by a user, and based on this trace the recognition method work its magic. If the shape the user draw gets identified, an associated command is executed.

Symbolic gestures is often used to simplify tasks, such that you do not have to go deep into menus to find hidden options. An example of this could for instance be saving a document. Just draw an *S*-formed shape, and the document is saved. A good gesture recognizer will be able to recognize many different shapes, including text, numbers, triangles, squares and many more. See figure 4.2 for examples.

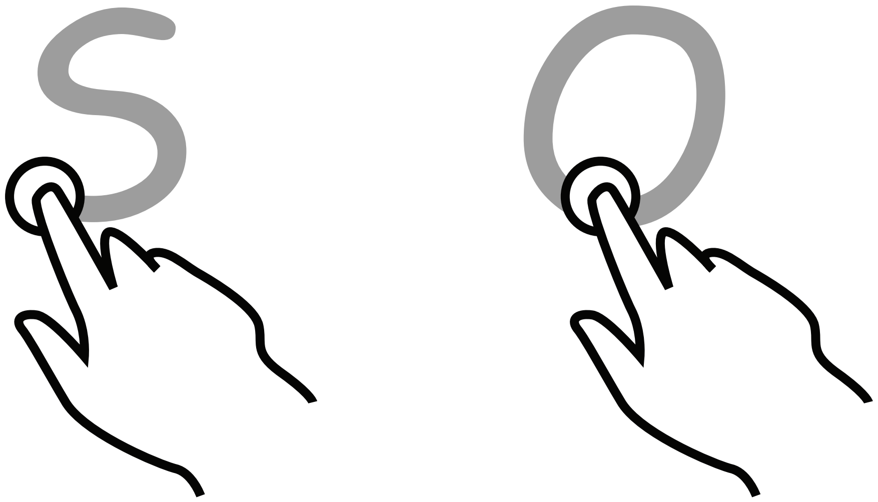


Figure 4.2: Example of symbolic gesture. Drawing of the letter *S* and a circle. Illustrations adapted from Gestureworks (www.gestureworks.com).

In this project the focus has been on symbolic gestures, and further references to a *gesture* is to be interpreted as a symbolic gesture. The focus is on symbolic gestures as these are less explored in the multi-touch domain than direct gestures. Another reason for this choice, is that it is much easier to implement direct gestures. PyMT already have a great implementation of this.

While implementing I have focused to make my changes to the PyMT framework as transparent as possible. Old applications utilizing the gesture recognition in PyMT implemented before my changes will still work flawlessly, they will even benefit from the new gesture recognition system without even having to change one line of code. New applications will have the option to choose a user selectable algorithm, but leaving this option blank will default to using a hybrid solution composed of many recognition algorithms.

The next section will present some existing gesture recognizers, and why I do not use them in favor of implementing something new.

4.1 Existing gesture recognizers

Today there exist quite a few gesture recognizers, but the majority of these recognizers are implemented to operate in a single pointing device environment. They are not able to recognize multiple gestures in parallel, and have therefore little utility in a multi user environment. Although this may be a problem, many existing recognizers have the fundamentals in place, so they are still interesting to have a look at.

iGesture [32] is one of the interesting frameworks currently not supporting multi-touch. iGesture supports the application developer who would like to add new gesture recognition functionality to their application as well as the designer of new gesture recognition algorithms. iGesture is a Java-based gesture recognition framework focusing on extensibility by providing an integrated solution which includes the iGesture recognition framework. This may in turn enable iGesture to be extended with multi-touch support. The iGesture framework can be configured to use any recognition algorithm, or customized gesture sets can be defined.

Among the gesture recognizers that actually work in a multi-touch environment is AME Patterns libraries, an incomplete gesture recognition framework. According to [28] is AME Patterns library a new C++ pattern recognition library, currently focused on real-time gesture recognition. It uses concept-based programming to express gesture recognition algorithms in a generic fashion. The library has recently(2009) been released under the GNU General Public License as a part of AMELiA [29], an open source library collection. It implements both a traditional Hidden Markov Model for gesture recognition, as well as some reduced-parameter models that provide reduced training requirements (only 1-2 examples) and improved run-time performance while maintaining good recognition results.

Both AME patterns library and iGesture are great libraries, but currently they do not offer any bindings to other programming languages than C++ or Java. Instead of writing bindings I implemented gesture recognition algorithms my self, according to the internal structure in PyMT. Much to keep the original structure the way the original project starters intended, and to contribute to this already fantastic and complete framework. Neither AME patterns library or iGesture have seen active development the past year.

4.2 Recognition methods

This section will explore some of the most known and used methods in gesture recognition. Seven methods have been implemented, and one is only described on a theoretical level. The implemented code feature two methods including direction based algorithms, one method using a region based approach, one using dot product, two methods involving artificial neural network and finally one hybrid method combining the best from some of the other methods. A method using Hidden Markov Models is described in theory.

To make the system as complete as possible, the implemented gesture recognition methods have been fully integrated into the open source project PyMT [20].

4.2.1 Input preparation

Common for all the gesture recognition methods is preparation of the input data. This has to be done because computers do not have the same gesture recognition ability as humans do. To people gestures would look almost the same regardless of where and how they were drawn. A computer relies only on coordinates, and a small gesture in the corner will look completely different from a big gesture drawn in the middle of the screen. Some people write small, some big, and in combination with either fast motions or really careful drawing it becomes evident that some kind of normalization is necessary. The input preparation code is inspired by the C code examples by Oleg Dopertchouk [8].

To make the recognition as accurate as possible the input need to be normalized and converted into a common format for storage. This will enable the possibility to later compare the gestures to new gestures. Since the chosen framework is PyMT, I have based the normalization part on the existing code, rather than recoding the whole procedure. The normalization process is based on common knowledge, and includes coordinate scaling, uniformly distributing the coordinates and gesture centering around point (0,0). The whole process can be seen in figure 4.3, and is further detailed below.

To make the process as fast as possible, it is important to keep it simple. The raw input from CCV is a stream of touch points with an associated ID sent over the TUIO protocol. The PyMT framework saves these points, and when a sequence of reoccurring IDs are identified, gesture recognition is initiated. Points with the same ID is grouped in a sequence, which from now on will be called a gesture stroke, or only stroke. It is this stroke that has to be normalized before it is used in the recognition phase.

During training of the gesture recognizer these normalized strokes are saved to permanent storage with an associated name. These names does not serve as an ID, so it will be possible to have different variations of in example a 'circle'. The name along with the score for the best match is what a user will see when testing gestures on a fully trained system.

The first step in the normalization process is gesture stroke scaling. This is done in a fast an efficient way by finding the outer bounds of respectively the x and y dimension of the stroke coordinates. Then assign the scaled coordinates according to this formula 4.1. When the scaling is complete the gesture stroke will look like the second image in figure 4.3. See how the axes have changed from the first image.

$$(x_s, y_s) = (\frac{x - x_{min}}{x_{max}}, \frac{y - y_{min}}{y_{max}}) \tag{4.1}$$

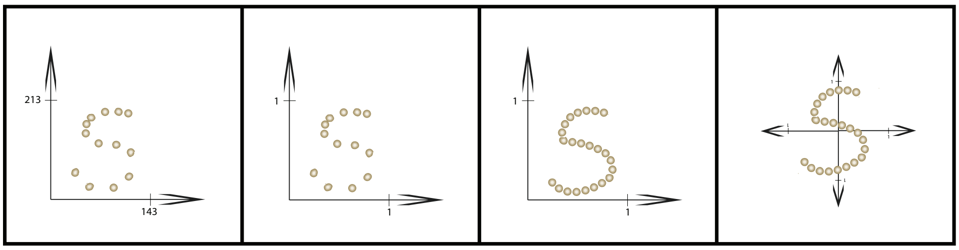


Figure 4.3: Illustrations of the four steps needed to prepare input for gesture recognition. The first image represent the raw input data received from the CCV. In the second image the gesture stroke has been scaled, and now all the points lies between 0 and 1. In the third image the gesture is normalized to include a predefined number of points with equal distance between them. The fourth image show the gesture stroke completely normalized and centered around origo.

In the second step we have to remove the variations caused by different peoples method of drawing gestures. A way to do this is to uniformly distribute the points in the gesture stroke. To be able to do this the gesture points we need to know the total length of the gesture. To measure the length, Pythagoras theorem is used. The length of all the individual line segments between each consecutive point in the gesture stroke is added together. For easier comparison to later gestures we should chose a fixed number of points to represent each gesture stroke. As long as all the gestures in the gesture database is represented with the same number of points, it does not matter what this number is. If you chose a small number of points the gesture recognition will execute fast, but you will lose some of the accuracy provided by a large number of points. 32 should be a good compromise between speed and accuracy, but this can be changed by a parameter later on if satisfactory results are not achieved.

When total length of the gesture stroke is calculated, the spacing between points can be obtained by dividing the total length by the number of points you have chosen to represent the gesture stroke with. In this case 32. To find the new points the old gesture is traversed. For each 1/31 of the total length, a point is added to the new gesture. Regardless of how many points the gesture stroke included, this 32 point long stroke will then be the new uniformly distributed gesture. The result of this can bee seen in the third image in figure 4.3. Compared to the second image, the stroke in image three contains a less number of points.

Finally the gesture stroke has to be centered around (0,0). To do this the arithmetic-mean of the x-coordinates and the y-coordinates are calculated separately. The mean will tell where the middle of the gesture can be found. Each stroke point gets subtracted with the arithmetic-mean for x and y respectively, and the result will be that the middle is moved to co-locate with the (0,0). The result of this can be seen in the last image in figure 4.3.

The normalized gesture stroke is now made, and can be used in the recognition process.

4.2.2 Dot product based recognition

The first method explained here is the recognition method included in PyMT [20]. Unlike all the other methods implemented in the gesture recognizer, I have not made any significant changes to this method. This method bases gesture recognition on the principle behind dot product. According to [8], by calculating the dot product between two gestures, it is possible to estimate how alike they are. A dot product result closer to 1.0 means close match, and a low or negative number mean that the gestures are completely different. This is possible because the gestures strokes prepared during input preparation is in theory a huge normalized vector.

In this case we use 32 points to represent a gesture, so the dot product will calculate the result of two $2 * 32$ dimensional vectors. The dot product is calculated using the standard formula for dot product(formula 4.2), and the result provide us with the correlation between the vectors x and y.

$$x \cdot y = x_1 * x_2 + y_1 * y_2 + ... \tag{4.2}$$

This is by far the fastest and easiest gesture recognition method used in this project, and it can recognize letters and digits. This method is not universal, it will often have a problem separating circles and squares, but this is the price for simplicity and speed.

To make this method more robust, a simple rotation invariant mechanism is implemented. It will use the angle between the two first points in the gesture stroke to decide the rotation of the gesture. This is by no means a permanent solution, a small error in the start of the gesture might make it unrecognizable. To make the testing of all the algorithms fair, rotation invariance has been turned off during testing because not all of the algorithms have this feature implemented.

4.2.3 Direction based recognition using Levenshtein and context sensitive distance

The direction based algorithm is based on the direction the points in a gesture stroke progress. Possible directions a stroke can progress is divided into 8 different zones like an 8-connected path which can be seen in figure 4.4. For each point in the gesture stroke the algorithm starts by calculating the angle between itself and the following point. The angle is used to determine which one of the directions that are closest to the vector formed by these two points. The output from this is a chain of values describing all the shifts in direction during the gesture stroke. An example sequence is shown in figure 4.3, with the matching path illustrated in figure 4.5. The pseudo code for building the direction list is listed in 4.1.

```
def calculateDirectionList(gesture):
    #build list of directions of each step in the gesture
    lastPoint = None
    for point in gesture:
        directionList = []
        if my_lastPoint == None:
            lastPoint = point
        else:
            directionList.append(calculatePointDirection(lastPoint, ↩
                ↪ point))
    return directionList

def calculatePointDirection(fromPoint, toPoint):
    x = toPoint.x -  fromPoint.x
    y = toPoint.y -  fromPoint.y

    angle = math.degrees(math.atan2(x,y))

    if angle < 0:
        angle = 360-abs(angle)

    direction = int(round(angle/45.0))
    if direction == 8: direction = 0

    return direction
```

Listing 4.1: Pseudo code for building the direction list

When you have obtained the direction sequence for the gestures you want to compare, there are different possibilities for how to proceed. During this project two direction based recognition algorithms has been implemented, one using a well known method of comparing strings, and one implemented using a self invented context sensitive method.

The well known method for comparing strings is ”edit distance”, and in this case Levenshtein distance [16] is used. Levenshtein distance is defined as the minimum

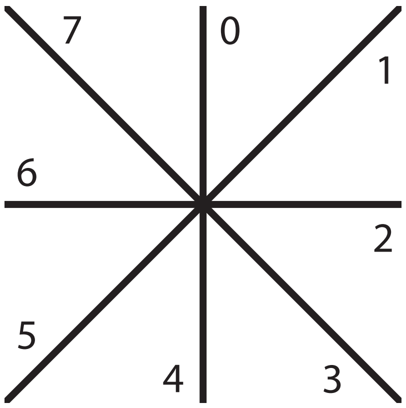


Figure 4.4: Illustration of the different directions a gesture stroke can progress.

number of edits needed to transform one string into another, with the allowable edit operations being insertion, deletion, or substitution of a single character. Levenshtein distance is chosen because of its simplicity, and the ability of comparing strings of unequal length. The pseudo code for Levenshtein method is shown in listing 4.2.

```
def levenshteinDistance(directionList1, directionList2):
   listLength1 = len(directionList1)
   listLength2 = len(directionList2)

   #Matrix of the same size as the two direction lists to compare
   matrix = [0...listLength1][0...listLength2]

   #Fill inn the initial direction index in the matrix
   for i in range(0,listLength1 + 1):
      matrix[i][0] = i

   for j in range(0,listLength2 + 1):
      matrix[0][j] = j

   #Start in the top left corner, and start filling out the matrix
   for j in range(1,listLength1 + 1):
      for i in range(1,listLength2 + 1 ):
         if directionList1[i-1] == directionList2[j-1]:
            matrix[i][j] = matrix[i-1][j-1]
         else:
            matrix[i][j] = min( (matrix[i-1][j]+1),      #Deletion
                                (matrix[i][j-1]+1),      #Insertion
                                (matrix[i-1][j-1]+1))    #Substitution

   #The number in the down right corner is now the minimum edit distance
   return matrix[listLength1][listLength2]
```

Listing 4.2: Pseudo code for Levenshtein distance algorithm

The self invented solution takes in to consideration the actual context, and how these direction numbers relate to each other. The difference between direction 3 and 4 is much less than the difference between 3 and 7. Instead of calculating how many insertions, deletions and substitutions, this method calculates the distance between two gestures in a much more fine grained manner. The pseudo code for the context sensitive direction method is shown in listing 4.3.

```
def contexSensitiveDirectionScore(directionList1, directionList2):
    listLength1 = len(directionList1)
    listLength2 = len(directionList2)

    #Check if lists are of equal length
    if listLength1 != listLength2:
        return "Error"

    #Initiate distance counter
    totalDistance = 0

    #For each index in the two lists add the difference in direction
    for i in range(listLength1):
        totalDistance += abs(directionList1[i]-directionList2[i])

    #Return the total distance between the direction lists
    return totalDistance
```

Listing 4.3: Pseudo code for the context sensitive distance algorithm

By showing two simple examples, it is easy to see that there is huge difference in the score these two methods will give for matching gestures. The two examples show a more or less straight line drawn from left to right. By comparing the gestures in example 4.3 and 4.4 we get almost the same score using both Levenshtein distance and the self invented method. For clarity example 4.3 and 4.4 are illustrated in figure 4.5 and 4.6 respectively. Using Levenshtein will give a match with the score 87.5%, while the self invented method will give a score of 95.8%. There is nothing wrong with this result, as minor differences in score will always be the case.

$$[2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2] \tag{4.3}$$

$$[1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2] \tag{4.4}$$

It is first when we try to compare two similar gesture stroke shown in example 4.5 (figure 4.7) and example 4.6 (figure 4.8) we will see a huge difference between the algorithms. In this case my algorithm gives a score of 89.3% while the method using Levenshtein gives a score of only 25%.

$$[1, 2, 2, 2, 3, 2, 1, 2, 2, 2, 3, 2] \tag{4.5}$$

$$[2, 3, 2, 1, 2, 2, 2, 3, 2, 1, 2, 3] \tag{4.6}$$

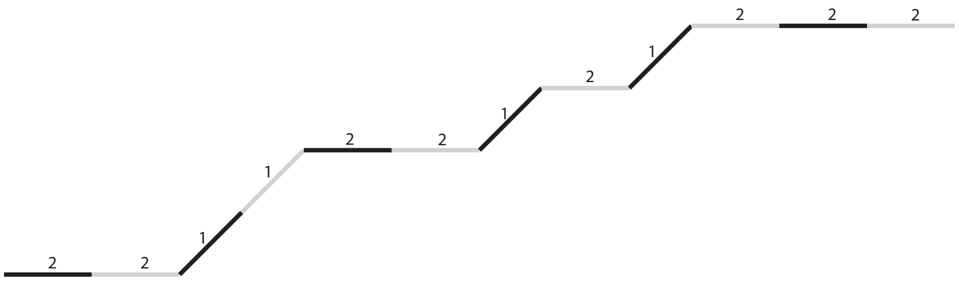


Figure 4.5: Illustration of the different directions a gesture stroke can progress.

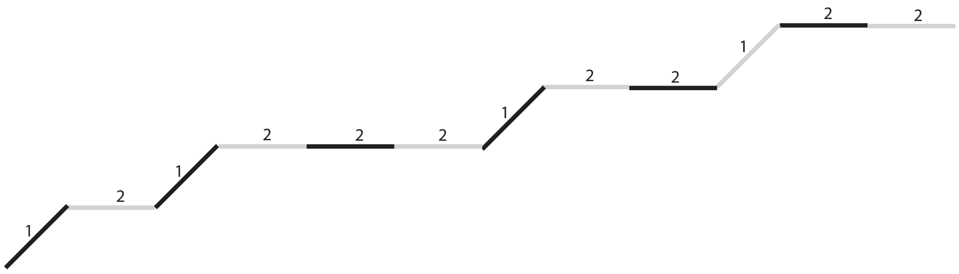


Figure 4.6: Illustration of the different directions a gesture stroke can progress.

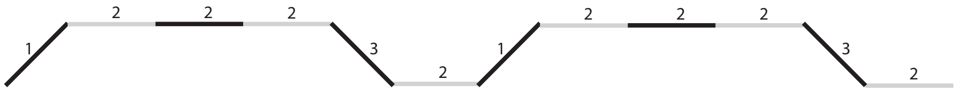


Figure 4.7: Illustration of the different directions a gesture stroke can progress.

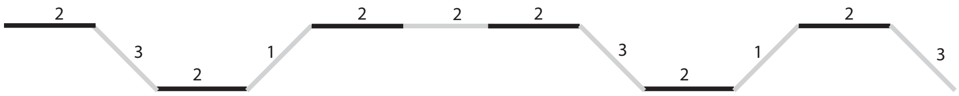


Figure 4.8: Illustration of the different directions a gesture stroke can progress.

Although the two examples presented here are simple, this will be the case for many different gestures strokes. Even though you try to be accurate when you draw a gesture, a bit *wobbly* noise is really hard to get rid of, and some of the directions gets interpreted as an another direction than that you intended.

To make the direction based method more robust, a rotation invariant mode has also been implemented. If the rotation invariant parameter is set to *True*, the gesture stroke is checked for a match in eight different angles by rotating the gesture around. Since we already have the direction sequence, the only thing we have to do to rotate the stroke is increasing each direction number by one, and apply 8 modulo to wrap the number around. The gesture is also checked backwards to try all possible ways a user can draw the symbol. In this mode the algorithm will have to run 16 times, but since the main computation time lays in calculating the direction list, this does not have a huge impact on the computation time. The algorithm is still able to recognize gestures in real time, without the user noticing any delay.

To further enhance the direction based approach other algorithms for edit distance can be implemented. It is also possible to use other types of algorithms to compare two strings like for example some kind of artificial neural network approach explained in section 4.2.6.

4.2.4 Region based recognition using Levenshtein distance

The region based approach relies on a different concept than the direction based approach. Inspired by the concept behind Xstroke [36] for Linux, the algorithm splits the gesture points into different regions instead of using their relative direction as a base. In this case it is a grid of 3x3 regions as seen in figure 4.9. The algorithm creates a sequence of which regions the gesture stroke crosses. To eliminate noise, the algorithm only records if the stroke crosses regions 1,3,5,7 and 9. Using a 3x3 grid is by no mean the best solution, but it makes the calculation fast enough for real time gesture recognition. See the pseudo code in listing 4.4 for an simplified overview of how the region list is built.

```
def _calculate_region_list(gesture):
    #Use this type of region mapping:
    #|7|8|9|
    #|4|5|6|
    #|1|2|3|

    regionList = []

    #build list of regions the points visit
    for point in gesture:

        #Calculate what region the point lays
        region = point.region()
        #The full region calculation code is removed for simplicity

        #Only append region 1,3,5,7 and 9. Do not add same region
        #if the last one is the same.
        if(region==1 or region==3 or region==4 or region==7 or region ↩
            → ==9):
            if(len(regionList)!=0):
                if(regionList[-1] != region):
                    regionList.append(region)
            else:
                regionList.append(region)

    return regionList
```

Listing 4.4: Pseudo code for how the region list is built

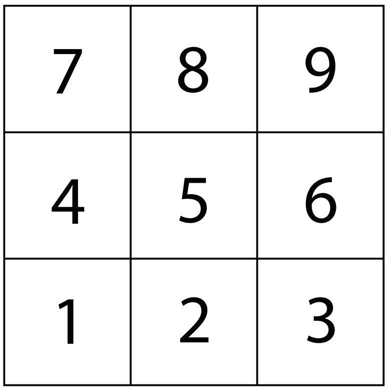


Figure 4.9: Illustration of the different regions a gesture stroke point can be inside.

As for the direction based algorithm, Levenshtein distance is used to calculate the minimum numbers of edits needed to transform one gesture string into another. The gesture in the database with the highest matching score above a given threshold is chosen to be the believed gesture.

Since the Levenshtein algorithm does not need the gesture strokes to be of equal length, the gesture stroke is shortened by removing duplicate region points in a row. This means that a gesture stroke like [3,5,5,7,7,7,7,9,3,3,3] will be shortened to [3,5,7,9,3]. This reduce the computation time, and simultaneously remove noise introduced by people that are drawing gestures in a very variable speed.

As an example the gesture symbol forming an *N* will generate the sequence [1,7,5,3,9] as seen in figure 4.10.

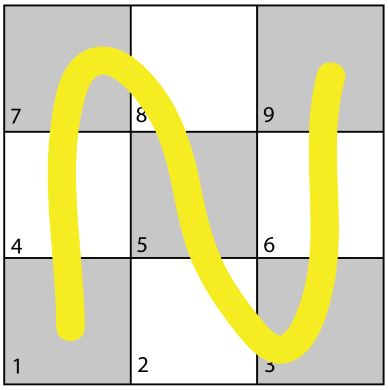


Figure 4.10: Illustration of the different regions a gesture stroke simulating a N will cross to generate the sequence [1,7,5,3,9].

4.2.5 Hidden Markov Model based recognition

According to [10], Hidden Markov Model(HMM) is one of the most important techniques within the field of symbol recognition. As mentioned in the introduction, a short theoretical introduction to HMM for gesture recognition will here be presented. HMM is a statistical model where the distributed initial points work well and the output distributions are automatically learned by the training process. It is capable of modeling spatio-temporal time series where the same gesture can differ in shape and duration. As [3] highlight, for a HMM method to be successful, the major challenge is finding a feature set that captures the essence of the gesture and yet is resilient to variability in scale, orientation, and sketching style. HMMs are especially known for their application in temporal pattern recognition such as speech[27], handwriting, gesture recognition.

As with the other gesture recognition methods, HMM also use the normalized sequence for pixel coordinates as input. This sequence correspond to the two dimensional locations of the finger drawing a gesture over time. This temporal unfolding of the gesture symbol in terms of pixel coordinates can be viewed as a form of gesture recognition[3]. The goal of the system is to compute and learn a set of model parameters that encapsulate symbol related discriminate temporal features. These trained models are finite and can be used to generate likelihood that some future observation, a new sequence of pixels, was generated from some particular known model. The overall goal is to develop one large classifier whose job is to recognize and classify gesture symbols. According to [3], one example of a general pattern recognition system can be applied to pixel-driven symbol recognition through the following steps:

1. Observation and capturing of the input data (Community Core Vision)
2. Pre-processing of data (Input preparation)
3. Extraction and identification of features (Could be angles between consecutive points)
4. Classify gestures according to features (Build a classifier with relations between features)
5. Optional post processing procedures (Eliminate false recognitions)

This small introduction is in no way a comprehensive guide to implementing a HMM for gesture recognition, as such subject could fill its own master's thesis. But for the interested reader, a thorough guide to Hidden Markov Models can be found in the excellent original documentation *An introduction to hidden Markov models*[27] by *Lawrence R. Rabiner*.

4.2.6 Artificial neural network based recognition using direction and pixel based input

Together with Hidden Markov Models is artificial neural network(ANN) a more complex approach to gesture recognition than dot product or direction based edit distance. ANNs is often used in pattern recognition tasks [4], and has the potential to these tasks well. The power of ANN is that it is a very smart modeling technique capable of modeling extremely complex nonlinear functions. Its success depends heavily on that the ANN has been optimally tuned, which is a very difficult task. The ANN used in the gesture recognizer implemented in this project is a feed forward neural network with backpropagation. This is a supervised learning method, and training is required to make the system to recognize gestures.

To use an ANN for gesture recognition, an input format must be defined. In this project two different approaches have been implemented. One using the raw data obtained by measuring the angles between the points in the gesture stroke, and one method that converts the gesture stroke into an image representation with pixel data. Although the input format is different, the rest of the process including training of the ANN, testing gestures for similarity and tuning stays the same. This section is split into several parts, first the network topology i presented, then the two input methods is described individually. After the input methods have been described, the whole process of training and recognition is described together for both approaches.

Network topology

The ANN implemented for the purpose of this project is composed of three layers, one input, one hidden and one output. Internally in the neural network all the input nodes is connected to all the nodes in the hidden layer, and all the hidden nodes are connected to all the output nodes.

The input layer constitutes of either 31 or 64 nodes, dependent on which input approach is used. For the direction based angle input method, 31 floating point numbers is used to represent in what angles the gesture stroke progress. For the image based pixel input, 64 nodes is used to represent a binary image of the gesture stroke.

Common for both input representations are that seven nodes is used in the hidden layer, and seven nodes are used in the output layer. The output layer has to have as many nodes as there are different types of gestures in the database. Optimally only one node gets activated in the output layer when a gesture is clamped to the input nodes. Using seven nodes in the hidden layer is by no means the correct number, but this was the number that provided the best result for a system with seven different gestures in the database. It is hard to choose how many nodes to include in the hidden layer, and what is optimal varies from input to input.

Figure 4.11 present the topology of a network with 31 input nodes. The network is illustrated with the input vector with directions, and the output vector with the scores for each gesture in the database.

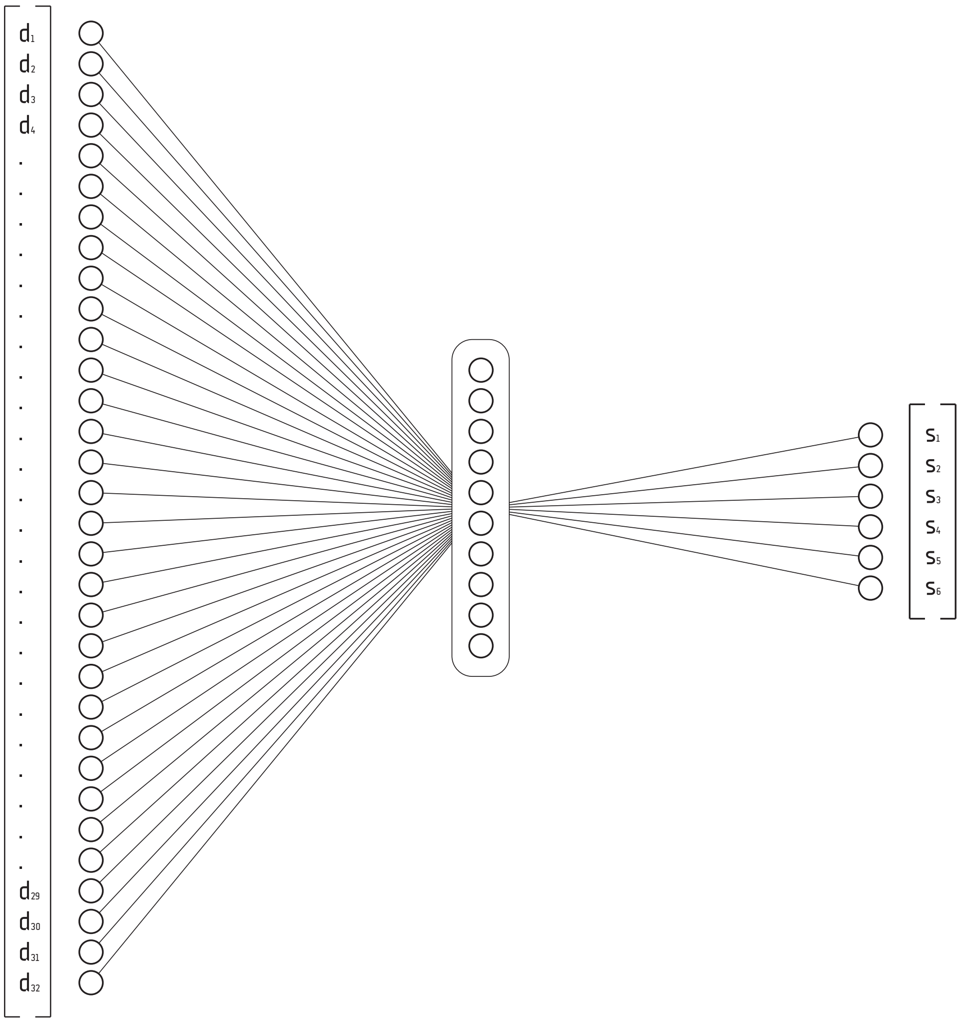


Figure 4.11: Illustration of the topology of the feed forward network used in gesture recognition. To the left is the input vector with directions, then the input nodes, then the hidden node layer. The hidden node layer has an all to all connection with all input and output nodes. To the right the score vector is illustrated.

Direction based angle input

The direction based angle input method lends some of the code from the direction based approach, and use the direction sequence explained in section 4.2.3 as a basis. A gesture stroke consist of 32 points, which together form a list of 31 angles, where the angle is calculated between each consecutive point. The raw direction list as obtained the way described in section 4.2.3 contain values between 0 and 7. To be able to use these values as input to the ANN, the values have to be normalized to lay between 0 and 1. The easiest way to do this is by dividing each number by the maximum direction value 7. This provide a list populated by 31 values in the discrete range between 0 and 1 with seven steps. To get a more fine grained input, it is also possible to calculate the pure angle between 0 and 360 degrees between each point in the stroke, and then normalize these values to get a continuous range of input values. Both these approaches have been implemented, as the latter method sounded very promising. After testing the two methods, the latter method was rejected as the discrete values provided by the first method gave better results.

The rest of this section assume that the discrete direction input is used. An example input list is shown for an optimal drawing of the gesture $\mathcal{N}$ in example 4.7. A normalized version of example 4.7 is shown in example 4.8. It is the latter example that is used as input the ANN to describe an $\mathcal{N}$.

$$[0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0] \tag{4.7}$$

$$[0,0,0,0,0,0,0,0,0,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0.43,0,0,0,0,0,0,0,0,0,0] \tag{4.8}$$

Image representation with pixel data input

The most common way to use neural network for symbol recognition is using images as input. The only problem in this thesis is that we do not have an image of the gesture, only 32 points representing the gesture stroke sequence. To be able to use pixel data as input to the neural network, the gesture stroke has to be converted into an image. For a stroke with 32 points, a 2D matrix of 32 x 32 points will be constructed to simulate an image. After a gesture has been drawn on the table, it has gone through input preparation, which mean we have to convert it to fit inside our 32x32 matrix. The 32x32 matrix has been choose because this is the only way we can be sure to reconstruct all the original point coordinates in the gesture. 32 x 32 equals 1024 pixel points, and is to large to be used as input to the ANN. Having 1024 input nodes would yield too high training time, and in addition a penalty in the recognition time.

We then have to convert the 32x32 2D-matrix to a more manageable size. This is done by merging neighbor pixels to reduce the image to half size. 16x16 is still more than desired, so the process is repeated to obtain a pixel grid of 8x8. 64 pixels is within the range of a manageable input size, so the grid is not resized further. An advantage of downsizing the matrix is also that the pixel values are now much closer together. What before were pixels spread across the image, is now a connected path depicting the symbol that is going to be recognized. Figure 4.12 illustrate how the downsizing process works on the gesture symbol $\mathcal{N}$.

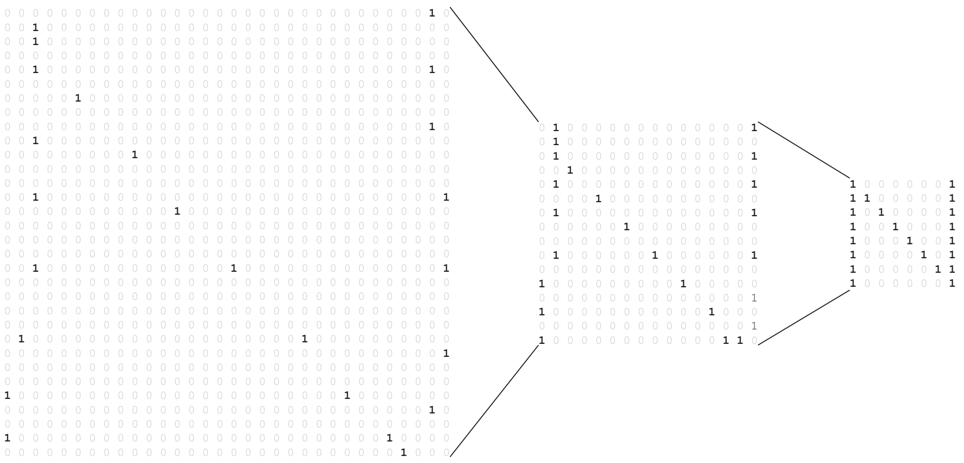


Figure 4.12: Illustration of the conversion of a gesture stroke to a pixel based image. The image is downscaled from 32x32 matrix to an 8x8 matrix as to enhance the computation speed, and reduce the number of required input nodes in the ANN.

Before the downsized 8x8 image can be used as input to the neural network, it has to be converted to a 1-dimensional array. This is done by starting in the top left corner of the image, and copying pixel values row by row into a new array. The 1-dimensional array corresponding to the gesture in figure 4.12 is illustrated in figure 4.13.

Training

The ground principle in supervised learning is that the neural network learns by *example*. Since two different input methods have been implemented, two ANNs needs to be trained in the gesture recognition system. The main steps in training an ANN can be listed as follows:

1. Initialize the weights in the network with a random value between -1.0 and 1.0

2. Retrieve gesture from training set

```
[1,0,0,0,0,0,0,1,
 1,1,0,0,0,0,0,1,
 1,0,1,0,0,0,0,1,
 1,0,0,1,0,0,0,1,
 1,0,0,0,1,0,0,1,
 1,0,0,0,0,1,0,1,
 1,0,0,0,0,0,1,1,
 1,0,0,0,0,0,0,1]
```

Figure 4.13: Illustration of the 1D-array that is used as input to the ANN during training and recognition.

3. Convert gesture to preferred input format
4. Define desired output for the corresponding input
5. Feed the network with the input data
6. Compute output
7. Compare output with desired output values, and compute the error
8. Adjust weights accordingly and repeat the process until a small enough error is obtained, or a preset number of training iterations have passed.

Although this may sound trivial, the process of training a complete system can be hard. This has to be done, and you may have to provide many different examples of the same gesture before the system starts to recognize different symbols. The learning part is very time consuming, and the number of iterations decide how accurately trained the system will be. The backpropagation part of the neural network propagates errors backwards when training the system. This is done to update the weights, and to ensure that optimal learning is obtained as fast as possible. The pseudo code for backpropagation is shown in listing 4.5.

```
For 1000 iterations or until error threshold is reached:
    For each gesture in training set:
        Calculate the output of the feed forward network
        Compare with the desired output corresponding to the symbol and compute error
        Back propagate error across each link to adjust the weights
    Compute average error and update weights
```

Listing 4.5: Pseudo code for backpropagation algorithm

Below are the steps in backpropagation listed from a mathematical point of view. The list include all the steps taken inside the inner for loop in pseudo code 4.5.

x represent the input vector, w_{kj} are the weights between nodes in two layers, and y represent the output vector. t is used to denote the desired output for an input x. For example should y_k be equal t_k for input x_k for an optimally tuned network. The math is based on a two-layer network like the one used in this implementation, and sum-of-squared error (E) is used calculate the partial error in the backpropagation routine. The output layer use a linear activation function, and the hidden layer have a logistic sigmoid activation function. η is the learning rate.

- Calculate the output of the feed forward network using forward propagation:

  Input activation: $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$

  Hidden layer activation using sigmoid function: $z_j = \tanh(a_j)$

  Output layer activation using linear activation: $y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$

- Compare with the desired output and compute error:

  Output error: $\delta_k = y_k - t_k$

- Back propagate error across each link to adjust the weights:

  Hidden layer error: $\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$

- Calculate partial error for which layer to *blame* the error:

  Partial error for input layer: $\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$

  Partial error for hidden layer: $\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$

- Update weights:

  New weight: $w^{(\tau+1)} = w^{\tau} - \eta \nabla E_n(w^{\tau})$

To achieve an optimal trained ANN, the input needs to be good quality. It is important to understand how complex the patterns to recognize are, and how chaotic the input space is. This decide the learning rate, and the number of iterations that are needed to train the network for a given number of input sets. As addressed in the result section 4.3.4, the test set data used while testing the system contain noise, which make the training of the neural network harder.

Recognition

When using the ANN for gesture recognition, the same steps as with training is used, except that there is no learning involved. The only thing that is needed is a forward pass, and then the output is presented on the output nodes. How this work can be shortly summed in three steps:

- Retrieve gesture from the test set (or the multi-touch table for *online* gesture recognition)
- Convert gesture to preferred input format
- Compute output

And more mathematically, where x are input node values, w_{kj} are the weights between nodes in two layers and y are the output node values:

- Input activation: $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$
- Hidden layer activation using sigmoid function: $z_j = \tanh(a_j)$
- Output layer values: $y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$

Optimally if a gesture is perfectly drawn, the node associated with the current gesture will result in the value 1.0, and all the other output nodes will be 0.0. The output of each node can be seen as the percentage of how similar the drawn gesture is to the gesture represented by the node. The output node with the highest value, is hopefully the node that represent the gesture you tried to draw.

4.2.7 Hybrid recognition method

The hybrid method is an improvised method, with the goal of combining the best properties from the dot product method, the direction based methods and the region based method. To do this, the hybrid algorithm runs each of the mentioned algorithms, and calculates a combined score for the tested gesture. As is evident, this method is not a self-contained algorithm, and requires the four other algorithms to be implemented prior to use of this method.

During recognition the input gesture is compared to each of the different gesture symbols in the database for similarity. In this case, that means 7 comparisons. When doing this for each of the mentioned algorithms we get $7 * 4$ scores. The scores per symbol is then added together, and we again have 7 scores. The gesture symbol with the highest combined score is chosen as the recognized gesture, and the user is presented with the result. Using this gives an advantage of robustness, as a small error in one of the algorithms gets washed out by the other three.

The reason for not including the neural network approach is to have a good method that do not require time consuming training of ANNs. The method can be used right out of the box as long as the gesture database have been loaded with some perfect matches for each gesture in the gesture set. The sore received from ANN based methods is also not directly comparable to the other scores received from the methods included in the hybrid method.

4.3 Testing

To be able to assess the quality of the gesture recognition algorithms implemented, a test has been performed. While reading this section, it is important to note that this is not an usability test, but rather a pure performance test executed in its natural environment. In addition to testing the gesture recognition algorithms themselves, this test also serves as a proof that the multi-touch table behaves correctly. This includes indirectly testing of all the underlaying hardware and software that enables the user to draw symbols on the table surface.

In addition to the pure testing routine presented in this section, the implemented code does also feature *online* gesture recognition that enables real-time gesture recognition. This implementation is not explicitly tested, but has the same performance as the automatic tests presented here.

This section presents what type of gestures is included in the test, how the test is performed, who participated and the test environment.

4.3.1 Symbolic gesture set

When choosing gesture symbols to be included in the gesture set, there are some things to have in mind. Usually when testing performance, standard tests exist, which serve as an indicator of how good the system is implemented. Unfortunately, such standard tests do not exist for symbolic gesture recognition, so I had to create my own.

The point of this test is to test the robustness of the system, so the gesture set should include both easy separable gestures, as well as gestures that are hard to distinguish for a computer. There is mainly three types of symbols that exist: Open symbols, closed symbols and crossing symbols. The gesture set include examples from all these groups. As open symbols the number 5, the letter S and the letter N were chosen. The 5 and S look alike, and tests the system if it can handle similarity. The open symbolic gesture are shown in figure 4.14

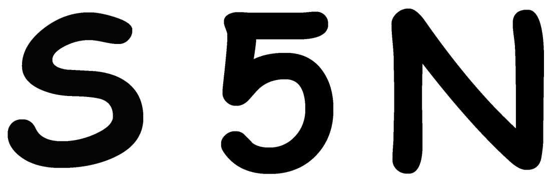


Figure 4.14: The open gestures in the symbolic gesture set that is used to test the performance of the gesture recognition system.

Another group of gestures is closed gestures. From this group a circle and a square have been chosen. These two gestures are also in a way similar, and are only separated by the fact that circles are rounded squares. For an illustration of the closed gestures see illustration in figure 4.15

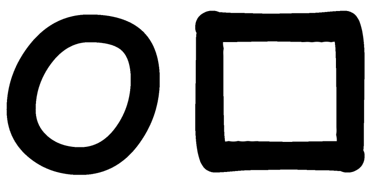


Figure 4.15: The closed gestures in the symbolic gesture set that is used to test the performance of the gesture recognition system.

The last group of gestures included is crossing gestures. From this group the number 8 and a *delete* symbol has been included. Figure 4.16 illustrates the crossing gestures included in the symbolic gesture set.

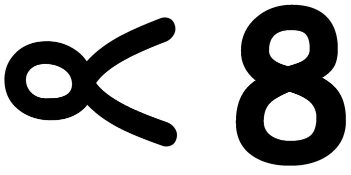


Figure 4.16: The crossing gestures in the symbolic gesture set that is used to test the performance of the gesture recognition system.

Together these seven gestures constitute a healthy mix of similar and unsimilar gestures for testing the accuracy, robustness and speed of the system.

4.3.2 Test subjects

To create a complete test set as possible, twelve people have participated. All the participants are students in their mid-20's(11 males, 1 female) with some minor differences in academic background. Eight study computer science, two economics, one social science and one design. All are interested in technology, and only two of the test subjects did not have a smart-phone with touch capabilities.

Noone had previous experience using symbolic gestures as input when using a computer or other device.

To make the test as authentic as possible, these 12 people were asked to draw five of each symbol presented in section 4.3.1. The multi-touch table was used as an input device, which none of the participants had used before. The symbolic gestures provided by these people were used to train and test the gesture recognition system.

4.3.3 Test environment

The test environment constitutes of the multi-touch prototype built during this project. The gesture recognition software runs on the multi-touch prototype computer presented in section 5.1.3. This computer is modern and fast, and produces results that can be expected to be similar to other modern computing systems today.

4.3.4 Test set

To be able to test the system, a test set is needed. The twelve people presented in 4.3.2 were asked to draw seven different symbols on the multi touch table. Each person were asked to draw each gesture five times, in total 35 gestures per person. With twelve people 35 times 12, a total of 420 gestures were supplied to run the tests. To enable as intuitive interaction as possible, the test subjects were not allowed to practice drawing the gestures before drawing their contribution to the test set. This may result in poor performance and bad recognition rate, but reflects how untrained people are able to intuitively use the multi-touch table without training.

The only guideline the test subjects got were me showing them a piece of paper depicting the gesture they were supposed to draw. The test set generation were automated by the system, and the program guided the test-subject through the process of drawing gestures. Figure 4.17 show a screen shot of the program telling the test subject to draw the symbol $\mathcal{N}$ the third time. The whole normalized version of the test sets used can be seen in Appendix C.

As an important side note, some of the people that helped me create the test set wanted to "test" my recognition system, and drew some ugly gestures that were harder to recognize. I have not removed these gestures from the test set, and think it is important to test the system for actual usage. By experience, during testing people

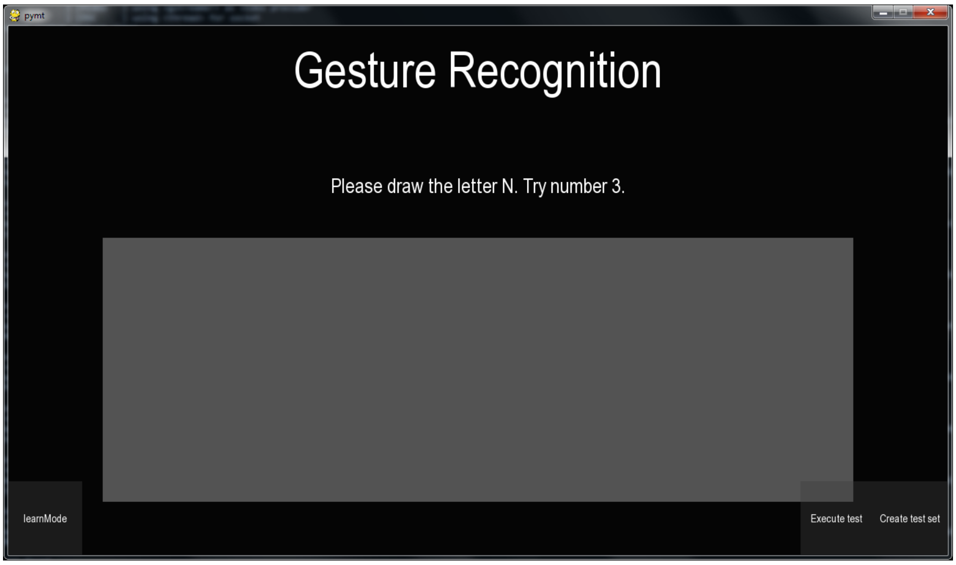


Figure 4.17: Screen shot of the graphical user interface for the gesture recognizer.

tended to draw fast and inaccurate when interacting with the system. Especially when they try the *online* gesture recognition system with immediate feedback.

4.3.5 Test method

After the complete test set has been created, the testing of the gesture recognition algorithms can begin. For the system to recognize gestures, it has to be trained. For the dot product-, direction-, region-based methods this mean to load a set of perfectly drawn gestures into the gesture database. These will be used to compare the gestures in the test set for similarity, and based on those results chose which gesture the input is most alike.

To enable the image based, and direction based artificial neural network methods to recognize gestures, two different ANNs have to be trained. For training the network the two first gestures of each symbol per person in the test set will be used. Up to five minutes is required to train the two networks. For information on how the concrete training process works, please read section 4.2.6.

After all the training is done, the system can start to execute the testing of gestures in the test set. For each gesture tested, all the implemented algorithms try to recognize the gesture. The execution speed and recognition result is saved, and later used to present the total recognition rate for each gesture symbol per algorithm. To make the test fair, rotation invariant is turned off because not all the methods have this feature implemented yet.

To make the testing process as efficient as possible, the whole test system is automated. The only thing the user have to do is to supply a test set, and press a button initiate the process. First the system automatically train ANNs, load perfect gesture to the database and then loop through all gestures in the test set. When all the gestures in the test set have been tested, the code automatically calculates the number of gestures tested, gestures recognized, recognition rate and speed. For convenient use in this document, the code output the results per algorithm directly into LaTeXformat tables.

To create an as full and realistic image as possible, the calculated execution times include all the necessary calculations. This include building a direction list for the direction based methods, and calculating region traversal list for the region based method. An exception to this is the training of the neural network, so only recognition time is measured for this method.

The following section present the test results for each gesture recognition method.

4.4 Results

This section presents the raw results obtained by testing all the gesture algorithms implemented. To be able to compare the performance of each algorithm, it is easier to have the whole image. The discussion will therefor come in the next section after all the results have been presented. Each table contain the results for one gesture recognition method.

Gesture	**Tested**	**Recognized**	**Recognition rate**	**Execution time**
S	60	60	100.0%	3.4 ms
5	60	23	38.3%	3.4 ms
N	60	60	100.0%	3.5 ms
Circle	60	58	96.7%	3.4 ms
Square	60	19	31.7%	3.5 ms
Delete	60	60	100.0%	3.4 ms
8	60	59	98.3%	3.6 ms

Table 4.1: Test results for the dot product based gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	28	46.7%	10.4 ms
5	60	58	96.7%	10.2 ms
N	60	60	100.0%	10.9 ms
Circle	60	50	83.3%	10.4 ms
Square	60	59	98.3%	10.5 ms
Delete	60	59	98.3%	10.3 ms
8	60	56	93.3%	10.1 ms

Table 4.2: Test results for the Levenshtein direction based gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	19	31.7%	2.3 ms
5	60	58	96.7%	2.3 ms
N	60	60	100.0%	2.1 ms
Circle	60	43	71.7%	2.2 ms
Square	60	60	100.0%	2.2 ms
Delete	60	60	100.0%	2.2 ms
8	60	54	90.0%	2.3 ms

Table 4.3: Test results for the context sensitive direction based gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	0	0.0%	11.9 ms
5	60	19	31.7%	12.5 ms
N	60	51	85.0%	12.8 ms
Circle	60	46	76.7%	12.4 ms
Square	60	60	100.0%	12.1 ms
Delete	60	60	100.0%	11.8 ms
8	60	50	83.3%	12.3 ms

Table 4.4: Test results for the region based gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	31	51.7%	0.5 ms
5	60	0	0.0%	0.5 ms
N	60	57	98.3%	0.4 ms
Circle	60	0	0.0%	0.5 ms
Square	60	58	96.7%	0.4 ms
Delete	60	0	0.0%	0.5 ms
8	60	56	93.3%	0.5 ms

Table 4.5: Test results for the direction based artificial neural network gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	0	0.0%	1.6 ms
5	60	59	98.3%	1.8 ms
N	60	60	100.0%	1.7 ms
Circle	60	60	100.0%	1.7 ms
Square	60	60	100.0%	1.8 ms
Delete	60	60	100.0%	1.8 ms
8	60	60	100.0%	1.7 ms

Table 4.6: Test results for the image based artificial neural network gesture recognition method.

Gesture	Tested	Recognized	Recognition rate	Execution time
S	60	32	53.3%	25.0 ms
5	60	59	98.3%	24.5 ms
N	60	60	100.0%	25.3 ms
Circle	60	60	100.0%	25.0 ms
Square	60	59	98.3%	24.8 ms
Delete	60	60	100.0%	24.5 ms
8	60	60	100.0%	24.5 ms

Table 4.7: Test results for the hybrid gesture recognition method.

4.5 Discussion

During testing mixed results have been obtained. Some recognition methods work better than anticipated, and other do not provide acceptable results. For a method to be useful, it has to be robust and fast. As evident by the results, it is possible to separate the algorithms into four different groups based on performance. The results show two recognition methods with very good performance, three with good performance, one with acceptable performance and one with bad performance. This section will present performance of the algorithms. It is important to note that none of the algorithms have been further enhanced or tweaked after they were tested with the test set. The reason for not doing this is to present as honest results as possible. This way, there is reason to believe that if twelve new people were to create a new test set, these results would be about the same as these presented here.

Before the execution speed of each algorithm is commented, it is important to define what is considered fast. The optimal performance would be real-time. Real-time is not a strictly defined time frame, and what to expect of the algorithms is therefore hard to define. Since we are dealing with feedback to the user, a good basis for what execution time to expect could be taken from the film industry or the gaming industry. A movie is filmed in 24 frames per second(FPS), and a game is considered smooth if the frame rate is above 30 FPS. By using a minimum of 30 FPS as a basis, an algorithm should finish recognition in $1000/30 = 33$ milliseconds or less to be perceived as real-time.

Among the the two algorithms classified as having very good performance, the Hybrid method stands out. The Hybrid method shows the most robust results with a average recognition rate of 93%, only degraded by the symbol *S*. Since the Hybrid method is a combination of four other recognition algorithms, there is no hybrid-specific tuning necessary for it to work properly. The most notifiable advantage of the hybrid method over each of the individual algorithms it consist of is that results get better because it reduces the consequence of flaws in each individual algorithm, and increases influence of its strengths. Some of the algorithms are great at separating squares and circles, whereas some interpret these as the same gesture. The reason for the mediocre performance for the symbol *S*, is that three of the algorithms included in the Hybrid method have poor performance regarding recognition of the symbol *S*. All the results for the hybrid method can be seen in table 4.7. On the downside, the Hybrid approach is not the fastest algorithm, since it executes four algorithms instead of one. With a speed of around 25 ms it is still within the limits I have defined as real-time.

If the Hybrid method is redesigned, as well as built from scratch, the execution time will be improved. Tasks like creating a direction list for the two direction based methods are now done twice, and could easily be reduced to one time if the hybrid method was to be implemented as a completely stand alone algorithm. Another way to increase the speed of all algorithms implemented is to use another programming language for performance critical code. Python is not a particularly fast programming language, and could strongly benefit from some C-code. Parts of the code could be implemented in in-line C, or for example with the aid of a mathematical library written in C. As this is a comparison of different algorithms, it can be argued that it is more interesting to look at the difference in speed between the algorithms, rather than the actual speed.

The other method classified as very good is the ANN approach using image based input. This is the most advanced method implemented in this project, and is the method with the greatest potential among the tested algorithms. Due to the random nature of ANNs, this is also the method that is the most difficult to optimally tune for gesture recognition. This is especially true when dealing with noisy input. As explained in section 4.3.4, some people intentionally tried to make their gesture contributions hard to recognize, only to "test" the robustness of the implemented system. The most significant effect of this, is that the gesture symbol *S* and *5* both look very similar to the system. For this reason, the tested ANN system is not capable of separating these symbols. The system will recognize all the *S* symbols as the symbol *5*. The results for the image based ANN can be seen in table 4.6. The reason for not being able to recognize *S* may be a accounted to the phenomenon over-learning, but I did not find any other parameter set that performed better than these results. I was able to make the system recognize about 70% of the *S*s in the test set, but this resulted in huge degradation to the recognition rate of the other symbols. This is a matter of choice, and I would rather chose to remove *S* or *5* from the gesture set than accepting weaker results. Relatively to the non-ANN based algorithms, this method is super fast. With an average of below 2 ms to recognize a gesture, this method can easily be called real-time. Despite the fast recognition time, it is important to remember that this algorithm require time consuming training of up to 5 minutes, although this is required only once when the system is initiated with a new gesture symbol set.

Two of the methods classified as *good performance* algorithms is both based on the same direction based principle, and is natural to directly compare. Initially when implementing these methods I believed the context sensitive method would perform much better than the Levenshtein method. As evident in table 4.3 and 4.2, the performance in regard to recognition rate is similar. Both methods have consistently high recognition rates on all gestures with the exception of *S* and *cirlce*. As with many of the other algorithms these two methods also suffer from the problem of separating the similar gestures *S/5* and *circle/square*. What separates these methods is the execution speed, with the context sensitive method being 4-5 times faster. Both methods is well within what would be natural to call real-time. These two methods have the potential to, with more tweaking, to compete with the Hybrid method.

The third method classified as a *good performance* algorithm is the dot product method. This is the only method capable of recognizing the gesture symbol *S* 100%. But this comes with the price of failing in the recognition of the symbol *5* and *square*. The dot product method is very fragile and limited by design, and only small variations in gesture symbols could degrade the performance. This method has its strength as a part of the Hybrid method more than standalone, which is the price to pay for simplicity. The results for the dot product method can be seen in table 4.1. Despite the below-average performance compared against the two best methods, the simplicity of the dot product method makes it fast. With an average execution time of 3.5 ms, the method is among the fastest methods tested.

The region based recognition method has been classified as having acceptable performance. The method has a major weakness, that there is a limited number of symbols it is capable of recognizing. As with the dot product method, the region based method is limited by design. Similar symbols like *S* and *5* would look exactly the same, and produce the same region list. The reason some *5* symbols is recognized is because of inaccurately drawn gestures which create distinct differences between *S* and *5*. Simplicity almost always comes with a backside. Although the region based method does not work very well on its own, it has its strength in contributing to the Hybrid method. The results for the region based method can be seen in table 4.4. To enable the region method to recognize a larger set of gestures, a larger grid could be used to calculate the region list. If 16 or more regions is used, the accuracy will increase, but not without cost. It is much more expensive in terms of calculation time to use a larger grid to calculate the region list, and as shown by the results, the region based method is already among the slowest methods implemented using only a grid of nine regions.

Only one method has been classified as bad. The direction based ANN method provide such bad results that it can potentially be seen as a non-working method. As evident by the results for the direction based ANN implementation in table 4.5, the method need further improvements to be of any utility in a gesture recognizer. I have not been able to find the reason for these bad results, but it is probably due to a mix of several problems. Increasing the number of iterations has generally a positive proportionality relation to the performance of the ANN. However, in certain cases, further increasing the number of iterations has an adverse effect of introducing more number of wrong recognitions. This partially can be attributed to the high value of learning rate parameter as the network approaches its optimal limits and further weight updates result in bypassing the optimal state. With further iterations the network will try to swing back to the desired state and back again continuously, with a good chance of missing the optimal state at the final iteration.

This phenomenon is known as over learning. The size of the input states is also another direct factor influencing the performance. It is natural that the more number of input symbol set the network is required to be trained for the more it is susceptible for error. Usually the complex and large sized gesture symbol sets require a large topology network with more number of iterations. Learning rate parameter variation also affects the network performance for a given limit of iterations. The less the value of this parameter, the lower the value with which the network updates its weights. This intuitively implies that it will be less likely to face the over learning difficulty discussed above since it will be updating its links slowly and in a more refined manner. Unfortunately, this would also imply a higher number of iterations is required to reach its optimal state. Thus, a trade-off is needed in order to optimize the overall network performance. The optimal value decided upon for the learning parameter is 0.5, but as evident this may be wrong.

4.6 Conlcusion and further work

With seven gesture recognition methods implemented, this system show what the multi-touch prototype is capable of. Since this thesis has broad focus, it has been important to test concepts, and not build complete solutions. These results show that there are huge possibilities in gesture recognition for multi-touch. With some effort this solution can be expanded to incorporate features like complete rotation invariant gesture recognition, multi-stroke gestures and not at least multi-modal interaction. The computing power of modern computers increase every day, and gesture recognition methods that before were slow, can now be executed in real-time.

Although gesture recognition is a great addition to your programming toolbox, there is also a backside. There are still some obvious obstacles to gesture recognition. We still have to manage noisy patterns that may be hard to interpret. No two persons are alike, which also is reflected in their pattern drawing. This makes the gesture drawing inconsistent, and what level of tolerance and variability that should be accepted must be considered. We need mechanisms to distinguish intentional gestures from unintentional. For example, when a person is preforming a direct gesture moving a picture around. If the picture unintentionally creates the pattern of a gesture, what should then be done? A challenge will also be to separate what is multiple single gestures, and what is a combined gesture, when many people use the table in a collaborative manner.

From a practical view it is important to consider what gestures to include into the gesture set. The general recognition rate would increase if the gesture set consist of easy separable gestures. We should avoid gestures like *S* and *5*. From a intelligent user interface perspective, it would be at least as important to find natural gestures people remember instead of forcing people to remember new patterns. What is natural for some, may be unnatural for someone else. It is important to look at cultural variations, especially if the developed program utilizing gesture recognition are to be used by people all over the world.

Chapter 5

Prototype

During this project, a prototype demonstrating the capabilities of multi-touch has been further enhanced based on the prototype build started August 2009. Since this project focuses on camera-based multi-touch, it is natural that this is the technology used to build the prototype. An important consideration when choosing technology was: what type of environment the prototype will operate in; who is going to use it; and what type of applications it will be running. Hopefully this prototype can be a part of future projects, either as an testbed for new applications or as a testbed for new multi-touch technologies.

The technology choice was made last semester, so this project has focused on enhancing the DSI method as much as possible. A table was chosen as the form factor, to make the prototype as flexible as possible. If you want to test a new camera-based method, the prototype can be easily adapted to this technology. The components used in this table, can also be rearranged to form a touch wall, if needed. But considering the size of such a construction, it is very unpractical for a prototype that will most likely be moved around as a showcase.

The prototype has gone through some major improvements during this master's thesis, both on the software and hardware part. The table now has wheels, handles, an electric adjustable projector mount, and many more function and design improvements. On the software side, the table now has in addition to normal finger tracking, the ability to do both gesture and object recognition. More on this later in this chapter.

An important goal of this build has been open source and low cost, so that as many as possible will be able to recreate a table like this on their own, without exhausting their budget. As mentioned in section 2.3, commercially available solutions based on camera-based methods do exist, but these solutions suffers from an extremely high price, as well as proprietary hardware and software.

This chapter presents the chosen technology, and argues for this choice. The specific parts needed to build the prototype will be presented, and any modifications needed will be described. The specific software used in the prototype will be examined. In the end I will explain some of the challenges I met during the build, and see how this prototype can be further improved. All information regarding the build, and components have been updated according to the current status of the prototype.

5.1 Chosen technology and components

I have chosen to use the principles of Diffused Surface Illumination(DSI) when building my prototype. This is the youngest method of all the camera-based multi-touch methods, and has some very interesting properties. By using this method I can utilize the advantages such as object recognition, and detection of hovering in addition to regular finger tracking. DSI is very similar to rear DI, but in contrast to DI, DSI provide an even illuminated surface for easier image processing.

Because DSI is the youngest of all the camera-based technologies, this was the technology I was able to find the least information about. Not many people have actually tested this method, or at least not built a full prototype of it. Research done on the topic of diffused surface illumination is sparse, and it has been hard to find evidence that it is better than similar technologies. Therefore it was interesting to try out this method, and see if this method was as promising in practice as it is in theory. For a overview of how the components are put together regarding to DSI, see figure 5.1.

The prototype consists of a projector, a camera, a computer, a projection surface, parts for capturing touches and a box to put everything inside. As the table is built with cost in mind, components are chosen after the principle of minimizing cost/value.

For an overview of how the components are organized inside the table see figure 5.2 and 5.1.

5.1.1 Box

The box is built using wooden boards, and screws. I am not a carpenter, so the design of the box has not been an important issue building this prototype. The box is built with modularity in mind, so if a new touch technology is desired, you can replace the top of the table separately, leaving the projector, camera and computer unchanged inside the box. This applies specifically to the method FTIR. To use this technology, the only thing that has to be done is to replace the Plexiglas EndLighten sheet with a regular acrylic sheet. I already have this sheet, so the swap can be done in a couple of minutes.

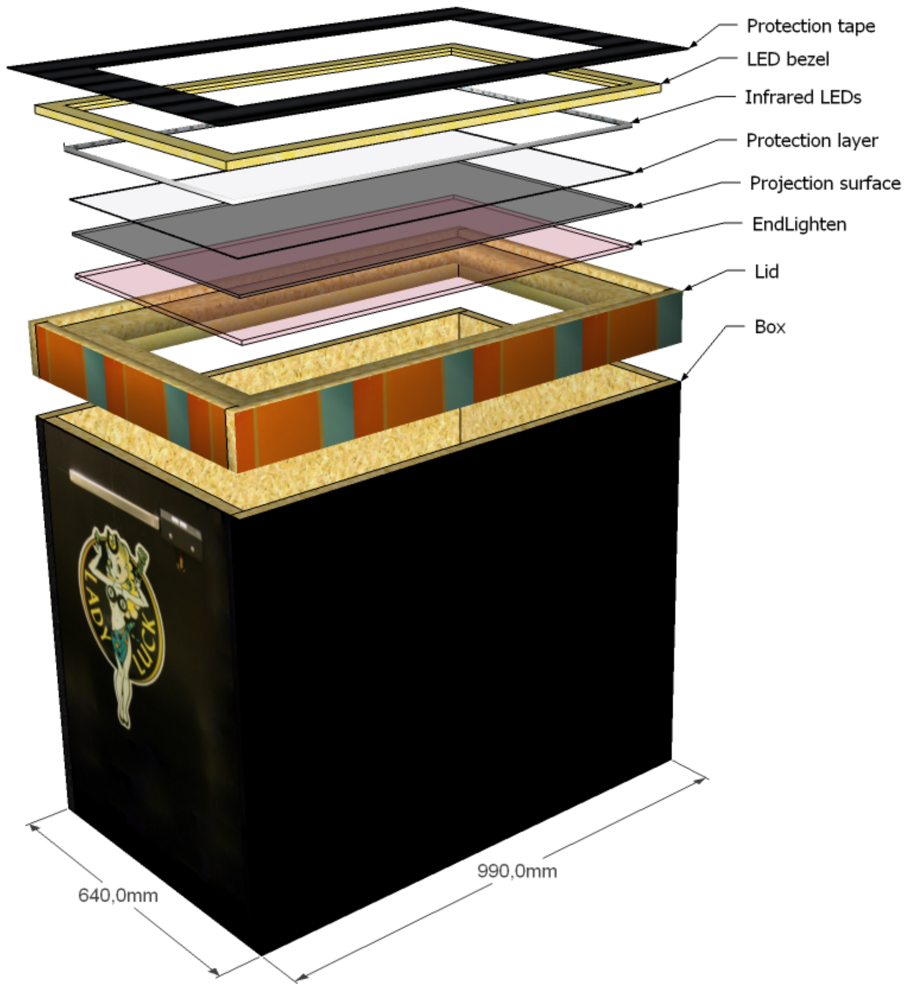


Figure 5.1: 3D illustration of the built table prototype showing how the layers in the lid are aligned according to the specifications of the DSI technology.

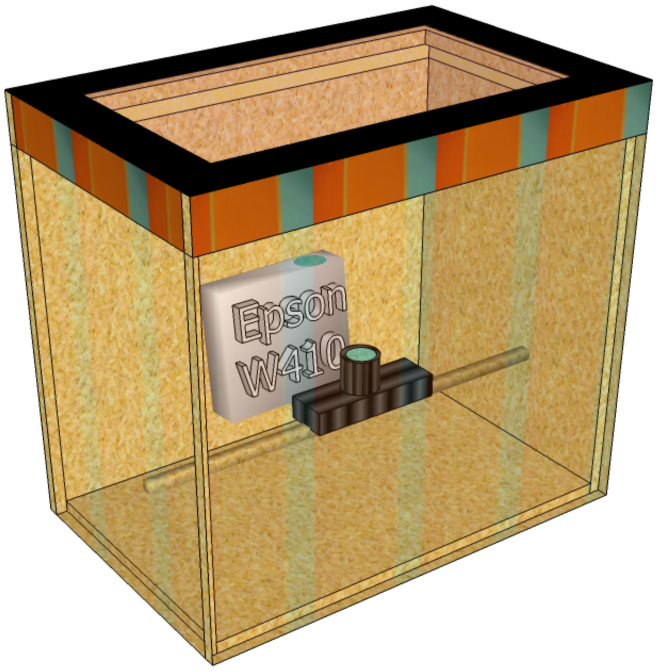


Figure 5.2: The illustration shows how the projector and camera are aligned inside the multi-touch table.

The box is made for collaborative use while standing, and therefore, it is 90cm high, which is a standard working height. See figure 5.3 for an overview of the structure, and outer dimensions of the box, and section 5.1.5 for a detailed explanation of the parts inside the top module.

Since the fall project, the table have been modified to make it more appealing to look at, more functional and practical. The table now have wheels with a lock function and handles, to make transportation easier. To make it easier to replace parts, the front door now feature hinges. These hinges also enhances the demo effect of the table, as it is now much easier to open the table to show people the different parts inside. In respect to function, the table now features an electrical adjustable projector mount which makes fine tuning the projector angle much easier.

The table have also been painted in black to obtain a more neutral look, but I have chosen to keep the top module painted in orange, yellow, brown and blue. This color choice is made to keep the table looking playful like a prototype should, and encourage more experimentation in the future.

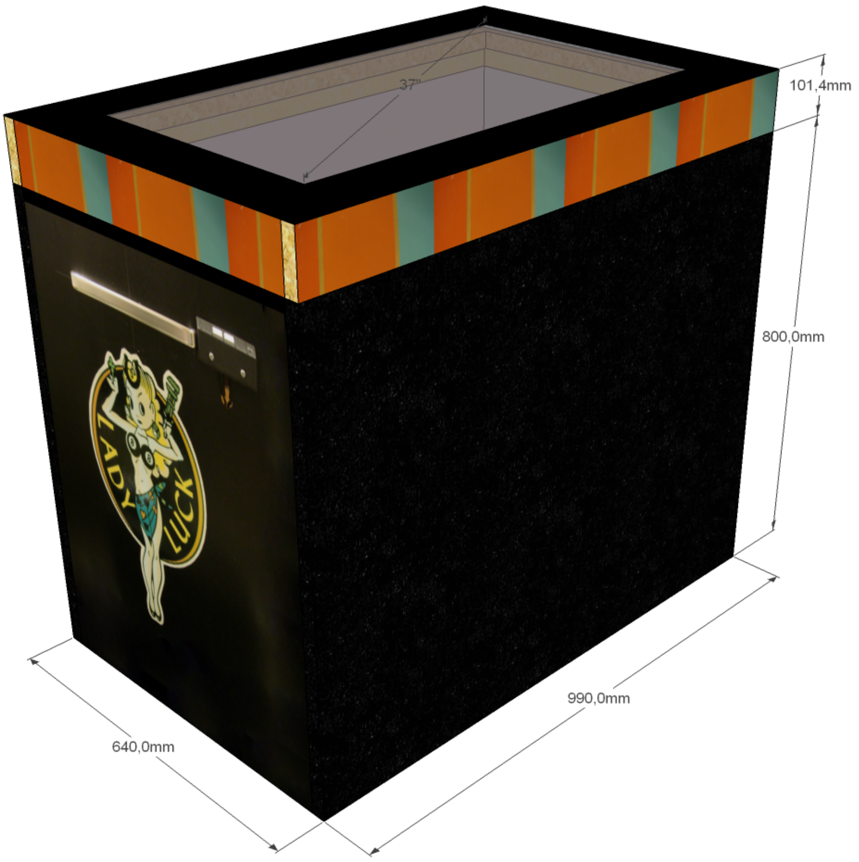


Figure 5.3: The outer dimensions of the finished prototype.

5.1.2 Camera

The camera used is a Sony PlayStation 3 ™(PS3) camera (figure 5.4), known for its low price and high frame rate. Frame rate is very important when working with multi-touch, as this decides the responsiveness of the table. A higher frame rate equals more images to process each second. The PS3 camera is able to process up to 120 frames per second when using a resolution of 320x240, and maximum 75 frames per second when using a resolution of 640x480.

The camera comes with a standard lens and a filter for enhancing the visible light specter. This build requires this filter to be removed, and instead replaced with an infrared band pass filter of the same wave length as the infrared LEDs used. This filter ables the camera to only see the infrared specter, and filter out noise and unwanted ambient light from the environment where the table is placed. To be able to put the camera closer to the table surface, a wide angle lens is used instead of the stock lens. The replacement lens also enable you to manually adjust the focus on the camera depending on the distance from the table surface.

Figure 5.4: Sony PlayStation 3 camera. Image courtesy of Sony.

5.1.3 Computer

A computer for such heavy processing of images dictates that more power is better. The most important component for images processing is the central processing unit (CPU), and the speed of the memory. The computer utilizes an Intel Core i5 750 processor, 4GB of fast memory, and a powerful graphics card. A powerful graphics card is really not needed, but enables development of graphics intensive applications.

The CPU has four cores, and operates on a clocking frequency of 2.66 GHz, which mean we can dedicate a singe core to process images. By dedicating a core to image processing we obtain stable and predictable speeds without degrading the performance of the rest of the system. We can then run heavy applications like games in parallel with image processing.

5.1.4 Projector

When choosing a projector, there are a lot of things to consider, and to be aware of. The first thing to think of is the size of the table, or to be more precise, the distance between the projector lens and the projection surface. The table will not be very user friendly if it is two meters tall. To shorten the projection distance, a short throw projector is needed. This is a special type of projector which is fitted with a lens capable of shooting a large image from a relatively short distance.

For the camera-based touch method to work optimally it is preferable that the projector emits as little light as possible in the infrared specter, but at the same time you want the projector be as bright as possible. The brighter the projector, the more ambient light from the environment the table will tolerate, while still preserving the contrast.

Projectors come in all kinds of resolutions and screen formats, and there is no definite answer on what projector one should choose. Usually greater resolutions means better image, but the price is often intolerable for the combination of very high resolution and short throwing distance. Most projectors support different formats like 4:3, 16:9 and 16:10, so this is often something you do not have to consider when buying a projector.

Figure 5.5: Epson W410 projector. Image courtesy of Epson.

The projector chosen for this build is Epson W410 (figure 5.5), which is one of the few projectors that meet the requirements stated above, while being affordable. This is a ultra short throw projector, which is capable of shooting an image of 37” from only 40 cm away, which is perfect for this setup. The projector is capable of

4:3, 16:9 and 16:10 aspect ratio, and 720p (1280x720) resolution, which provides a range of possible setups.

5.1.5 DSI parts

To construct a table using the DSI technique described in 2.2 some special components are needed. The core component in using this method is the Plexiglas EndLighten. There is not much freedom in choosing supplier, as there are only one manufactures who produces these sheets of glass.

The parts are combined in the way showed in figure 5.6. On the top there is a thin clear sheet of Plexiglas measuring 1.5mm in thickens for protection against scratches, which can be replaced. By protecting the other more expensive layers, the table can withstand rougher use.

The middle layer is the projection surface which the projector project onto. This layer is 5mm. This work like the principle in rear projection TVs, and provide a clear image on the opposite side.

The bottom layer is the EndLighten, and as you can see in figure 5.6 it is surrounded with LEDs. A LED-ribbon goes around the sheet, and light up the EndLighten from all sides providing a nice even light all over the sheet surface. This LED-ribbon consist of a chain of 850nm infrared LEDs with a spacing of 11mm between the LEDs.

5.2 Software

On the software side, some major improvements has been done. The table have now the ability to run object and gesture recognition as described in section 3.8 and chapter 4. The table still uses Windows 7 as the runtime platform. This is the only software component that needs a license, and it is closed source. Despite that, I have chosen this OS because I want to keep the possibility to run Microsoft Surface applications, which the table now have been updated to manage. The table now support full Windows 7 integration through the Human Interface Devices(HID) driver explained in section 3.9.

Before I can start tracking blobs with tracking software, I need image input from the camera. Since the camera is designed for Sony PlayStation 3, there are no drivers published from Sony for Windows, Mac OS or Linux. A member of the forum NUIgroup[6] has created an open source driver for Windows, so I am using this driver in the setup. This driver is now updated, which make the whole software package much more stable.

As before the the table use Community Core Vision(CCV), although a bit modified version, to handle the input from the camera. The modified version contain

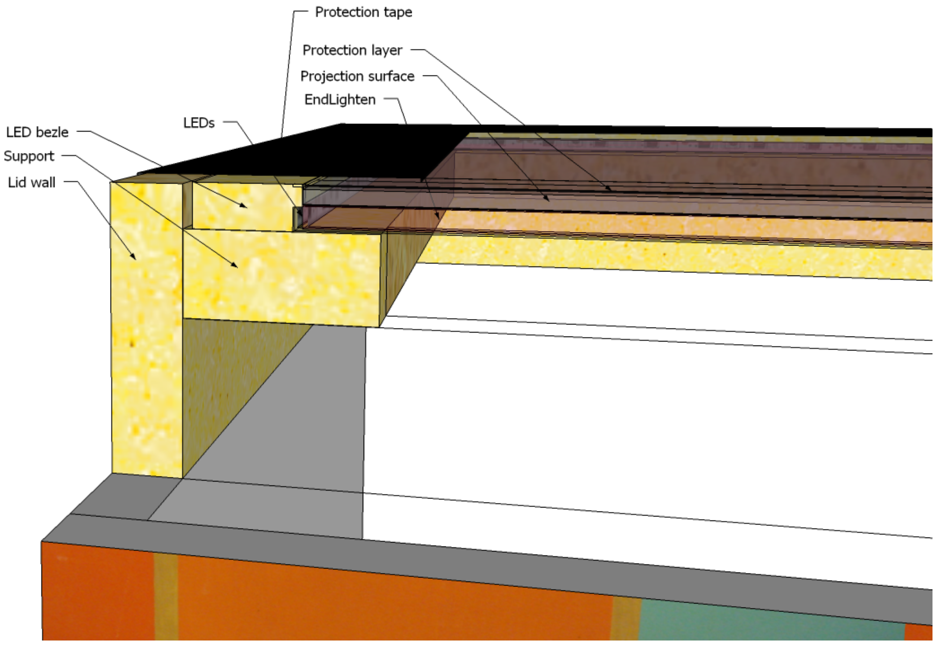


Figure 5.6: Close up image of the cut through profile of the table. Image shows how the different DSI components are stacked, and how the construction supporting the Plexiglas is built.

fiducial recognition to further enhance the complete multi-touch experience. CCV is still the best choice because this open source solution has the most active community, and is the easiest software package to customize. To read more about CCV please read section 3.6.

The CCV tracker use the TUIO-protocol(section 3.3) to transport information about touches and recognized objects to applications. In this particular prototype PyMT has been chosen as the main demo multi-touch framework, and all demos have been implemented with this Python framework. This framework has also been enhanced by me with a much stronger gesture recognition, which is one of the core features of the prototype.

As a bonus, the whole software setup is now much more stable.

5.3 Results

I am very pleased with what I have accomplished to build during the fall project and master's thesis, especially considering the time constraints. The table has a total price tag of about 30 000 NOK, which is reasonable price compared to other commercially available solutions. The table is tracking fingers with high accuracy, and objects with reasonable accuracy. The response time is fast. I wanted the table to be as precise as possible, so I have chosen to use the camera with the 640x480 resolution. Regarding the frame rate, limitations exist both in the computer and the projector. There is no point in processing images much faster than the refresh rate of the projector, as we will not be able to see any changes before the image updates anyway. The computer also has limited processing power, with a maximum of about 66 frames per second depending of the number of simultaneous fingers or objects that are present in each image frame.

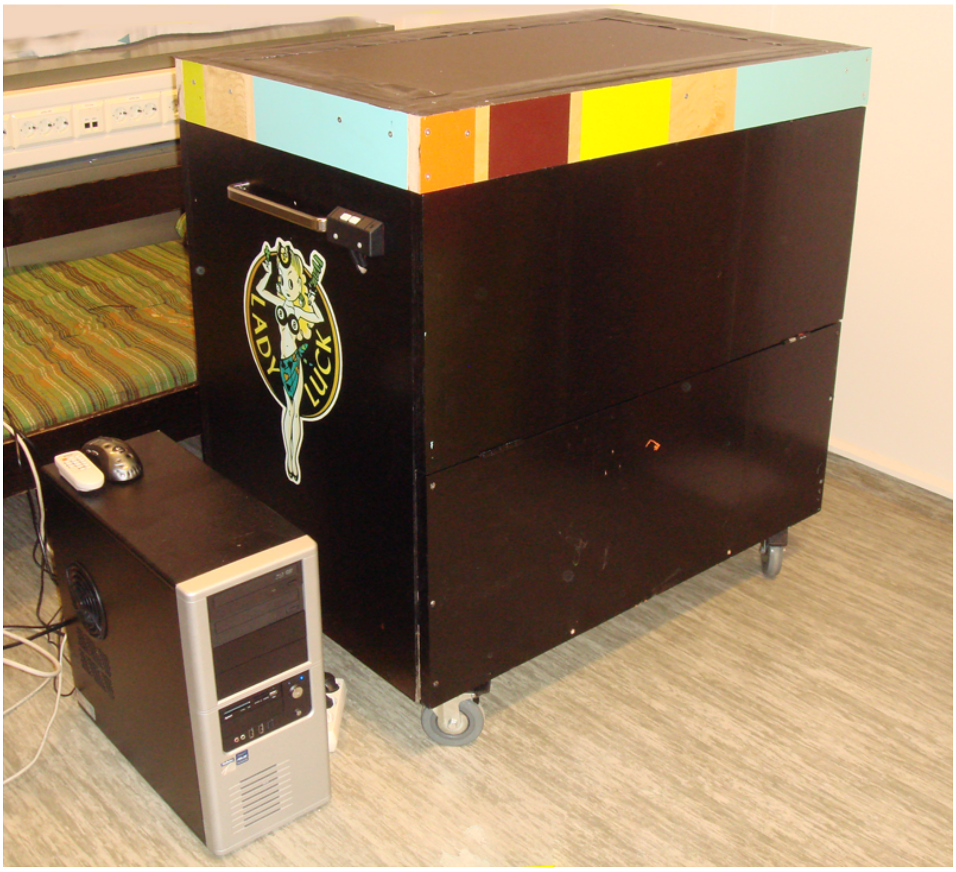


Figure 5.7: Photo taken of the finished prototype. The computer is at the moment outside the box, but will on a later stage be put inside the box.

The table is built with a screen size optimized for the 16:9 aspect ratio, and a 37" screen surface. This makes the table large enough for many people to interact together in cooperative manner. The resolution is set to 1280x720 which is in the top range of resolutions for such tables. Figure 5.7 shows an image of the completed table with the computer outside.

Although the table works satisfactory, there is still room for improvement. The table should ideally be more tolerant of changing light conditions, but with a limited budget it is hard to make everything perfect. To increase the camera resolution, it would also be good to use more than one camera. Two cameras would especially benefit object recognition. Using more than one camera would also enable the possibility to spread object and finger tracking load over more CPU cores.

The table has gone through gesture recognition testing (section 4.3), which also include testing of all the underlaying hardware and software that makes it work. In addition I have received a lot of feedback from other students testing and playing with my prototype. There was exclusively positive feedback, and people are usually amazed of what has been accomplished. This in turn supports the fact that there is great interest for multi-touch as an interaction method, and new and exciting ways to use computer systems will emerge.

Since the build of this prototype was started in August 2009, other companies have released commercial solutions. The market is tightening, and the competition is getting harder. This is truly an evidence that multi-touch technology is maturing, and we are certainly going to see more of this in the future.

5.4 Challenges

During the project some challenges and problems arise, and had to be solved. During the start of the prototype build in August 2009, the largest problem was limited time, and to receive all the components fast enough. It took a great deal of time to get the order of the components through the bureaucracy of my university. Some of the components, like the camera, LEDs and all the Plexiglas sheets had to be ordered from abroad, which meant delays. Payment, shipping, custom declaration and other unknown factors often take more time than calculated.

When all the components arrived, it was down to my carpenter skills to build a suiting box to house all the components. For obvious reasons, such as me being a young computer science student without much woodwork experience, my carpentry skills are not above average. Despite this, the table turned out okay, and nobody got hurt in the process of me operating power tools.

The brackets for mounting the projector did not work the way I had intended, so to position the projector correctly, custom mounting brackets had to be ordered. This was never delivered during the fall project, so the projector was temporarily held in place by tape, and some books. Although this did not present any performance degradation, it still made the solution look unelegant.

When I received the Plexiglas EndLighten, the edges were a bit rough, so they had to be polished to increase how much light it was possible to send into the sheet. After polishing the edges they become much clearer, and the light distribution was improved.

During the spring, more waiting was a problem, due to the missing projector mount. After finally getting it, it did not fit the way I wanted. But this time I took action myself, and modified the mount to fit. During this modification I had to disable some features like motorized projector adjustment, which now only works if a smaller image on the table surface is acceptable.

The toughest challenge has been to get the camera to behave correctly. This may sound like a simple task, but considering constantly changing light conditions, and that the camera need to both function for finger tracking and object tracking, it has not been easy. I have used countless hours to make this work reasonably well with all components working together.

These challenges presented here may not sound very problematic, but under these limited time and budget constraints they do have an impact. The table is a bit rough around the edges, but in the end the table was finished. After all, it is important to remember that this is a prototype and not a final version.

5.5 Further improvements

Even though the table is fully functional now, it is room for improvements and additional features are very welcome. Besides making the software and hardware more stable, there are three kind of improvements that could be done.

Design improvements , such as making the table visually more appealing, like making it lighter and thinner. This can be done by rounding the corners and painting it, or it can be done by a total renewal of the design. The table is now made of heavy wood, so the only way to make the table physically lighter is by building it in a lighter material. To make it thinner, some drastic changes has to be made. Because the projector has a minimum throwing distance, the only way to make the table thinner is by introducing some mirrors into the mix. The projector has to be placed horizontally inside the box, so that the projector beam gets relayed via the mirror before hitting the projection surface. This way, the projector does not need a very tall table.

Functional improvements , such as adding new components to enhance the user experience. Although the projector mount that is fitted inside the box now does the job, it could be made even better. The electrical adjustable projector mounting system is not used to its full potential the way it is built now. Secondly, some fans could be added to remove heat from inside the table. Since the table now has wheels, it could be wise to add these fan holes underneath the table. To further enhance the user experience, loudspeakers and a microphone could be added. This will make the table even more interactive, and even provide an interface if it is desirable to extend the feature set with multi modal interaction.

Software improvements , is the enhancement with the largest potential, where only the imagination sets the boundaries. At this point the table supports finger interaction, gesture recognition and fiducial tracking, but the software enabling these features could be made even more streamlined. Additional features not even thought of yet may also be enabled by software in the future.

Chapter 6

Conclusion

This master's thesis has been a continuation of a multi-touch interaction project I finished fall 2009. In the previous project, a build of a multi-touch table prototype was started, which in this thesis has been further enhanced, both in respect to design, function, hardware and software. The technology chosen for the multi-touch table is the technology I found most promising during the initial research in the fall project; a technology called Diffused Surface Illumination. This is a camera-based multi-touch method that offers many features, including true object recognition, which no other technology in the market is capable of providing.

In addition to further enhance the multi-touch table prototype, I have in this thesis shifted the focus more over to the software part of multi-touch, including gesture recognition and object recognition. A complete gesture recognition system has been implemented, and different gesture recognition algorithms have been tested in a multi-touch environment. With seven gesture recognition methods implemented, this system show what the multi-touch prototype is capable of. Since this thesis has such a broad focus, it has been important to simply test concepts, and not build complete solutions. A pure performance test of the gesture recognizer has been executed in its natural multi-touch environment. In addition to testing the gesture recognition algorithms them self, this test also served as a proof that the multi-touch table behave correctly. This include indirectly testing of all the underlaying hardware and software that enables the user to draw symbols on the table surface. The good results from this test show that there are huge possibilities in gesture recognition for multi-touch, and only the imagination sets the limit. With some effort the gesture recognizer can be expanded to incorporate features like complete rotation invariant gesture recognition and multi-stroke gestures.

The table has a total price tag of about 30 000 NOK, which is reasonable price compared to other commercially available solutions like *Microsoft Surface* which cost 150 000 NOK and *Evoluce One* with a price tag of 80 000 NOK. This proves that it is possible to make a cheap and open solution. The price of computer hardware continues to drop, and enables more and more people to take part in the new way of computer interaction. Using open source software makes it possible use software on my table, and in the same time contribute back to the community with new ideas and code. Since the gesture recognition methods implemented during this project has been fully integrated into the open source project PyMT [20], I will contribute this code back to the community. After this master's thesis is finished, I will clean my implemented code, remove the gesture recognition methods with poor performance, combine the best gesture recognition methods, further enhance them and submit the code back to the code repository for PyMT, and hopefully it will be included in future releases.

Although no formal tests other than the indirect tests during gesture recognition testing has been done, I have received a lot of feedback from other students testing and playing with my multi-touch table prototype. Especially from the twelve people that helped me create a test set for the gesture recognizer. There was exclusively positive feedback, and people are usually amazed of what has been accomplished, and they really liked the natural way of interacting with the computer. One of the test subjects even achieved a state of ecstasy, and kept talking for hours about how bright the future for human computer interaction was going to be. This in turn supports the fact that there is huge interest for multi-touch as an interaction method, and new exciting ways to use computer systems will emerge.

Human-computer interaction is very interesting, and will become more in focus in the years to come. It is possible to look at multi-touch as just the beginning of a more interactive world. Logical steps further will be more focus on multi modal interaction, which will make computer interaction even more natural. As faster computer hardware get available, speech recognition improves and virtual reality becomes closer to reality. We will then be able to produce software in completely new dimensions.

A good example of how far we have come in the research can be seen in *Project Natal*[19] by Microsoft, which will be commercially available during 2010. This is a camera-based 3D vision system that you attach to your gaming console (Xbox 360), which makes the gaming system able to "see" you. The system offers full body tracking of multiple people, and in addition fully capable speech recognition, allowing humans to interact with the gaming console software in new levels.

In the end I think this thesis has accomplished to enlighten some interesting topics, and I look forward to see the world starting to use this new amazing technology.

Appendices

Appendix A

Multi-touch technologies

This appendix introduces the most known and used touch technologies with focus on multi-touch. Three primary types of touch technologies exist: resistive touch, capacitative touch and optical camera-based touch, which are all based on different principles. Resistive and capacitative touch will be examined in this chapter, while optical camera-based multi-touch will be the focus in appendix B. In the end of the chapter there is also a small overview of other touch technologies currently not able to sense multiple touches. The development of new technologies, and new techniques are improving every day, so it may not be long before some of these technologies will be capable of multi-touch as well.

A.1 Resistive multi-touch

Resistive touch technology has been the most common technology used in consumer electronics up until now. Resistive technology is cheap, and thus readily available for high volume applications. Resistive screens require a degree of force when used, and perhaps technically are not touch screens. Resistive technology is often used in everything from handheld devices to cash registers in stores.

As described in [9] resistance based touch surfaces generally consist of two conductive layers coated with a resistive material and separated by an insulating layer, usually made of silicon dots. See figure A.1. When contact is made to the surface of the screen, the two sheets are pressed together, registering the precise location of the touch.

The registering of the touch happens because the conductive layers are connected, and establishes an electric current. This current is measured both horizontally and vertically by the controller in order to determine the exact position of a touch.

Traditional resistive screens cannot sense an approaching finger or multiple

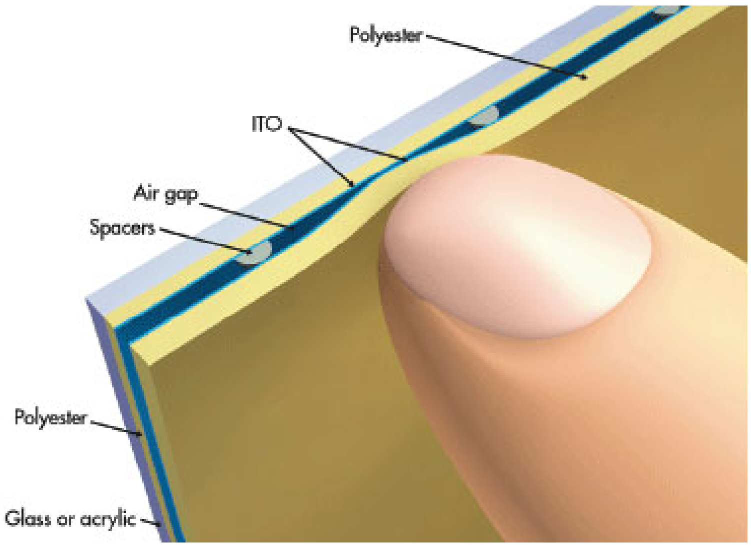


Figure A.1: Standard resistive screen technology. Image courtesy of www.electronicdesign.com.

fingers. Especially multi-touch is now in high demand, and a device without this feature is hard to sell. This is a shame, because resistive screens can provide very accurate input from contact with nearly any object like a finger, stylus, pen and palm.

Recent research by Stantum[12] and Touchco[13] on resistive multi-touch looks very promising. These are presented below.

Stantum has developed a new production technique much like the traditional technique, where the touch panel is a transparent overlay that allows touch sensing. It is placed on top of the display monitor. The touch panel is made up by different layers. Two thin and transparent (glass or plastic) overlays are covered with conductive material. This material is patterned in rows on one side and columns on the other, transforming the layers into a matrix of conductive tracks. The two layers are assembled superposed, the conductive sides facing each other and separated by a spacing material (transparent dots, air, etc.). See figure A.2.

When one or multiple touches occur on such touch-panel, the top layer slightly bends, thus creating contact between the two layers right below the touches. The controller chip detects the electrical contacts and determines the exact location of the touches.

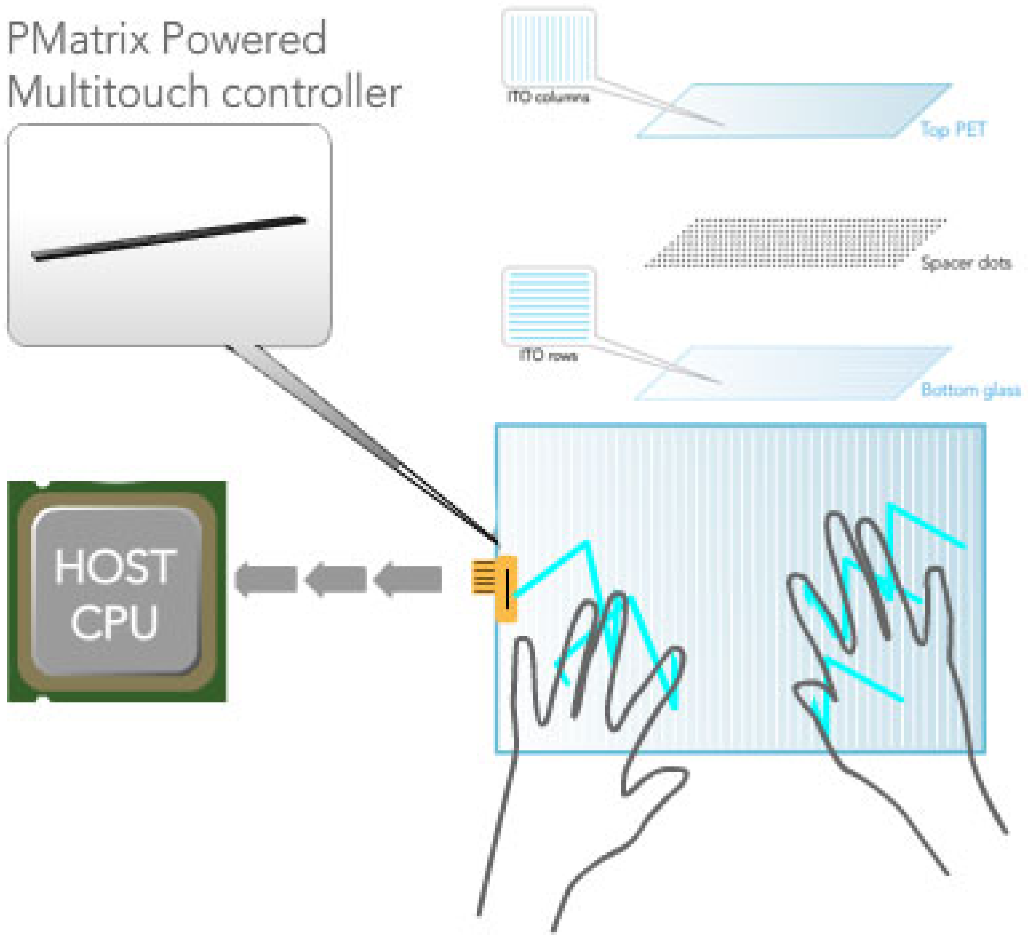


Figure A.2: Multi-touch resistive screen by Stantum. Original image courtesy of www.stantum.com (Image has been modified for this thesis.)

Touchco is developing a new technology they have named Interpolating Force-Sensitive Resistance (ISFR). Touchco's patented IFSR system is a disruptive new technology that provides high-resolution multi-touch tracking using an innovative new configuration of low-cost electronics. As described in [13] the system utilizes force sensitive resistors which become more conductive as force is applied. FSR ink is bumpy at a microscopic level so that when two ink-coated surfaces are pushed together, the surface area in contact increases (See figure A.3). When this happens, electricity flows from one layer to another. FSR sensors operate by sandwiching layers of FSR ink between sheets of plastic printed with conductive wires. When pressure is applied to a point on an FSR sensor, current flows from powered wires in one layer, through the FSR ink, to wires in the other sheet. Measuring how much electric current flows through the system tells you how much force is being applied.

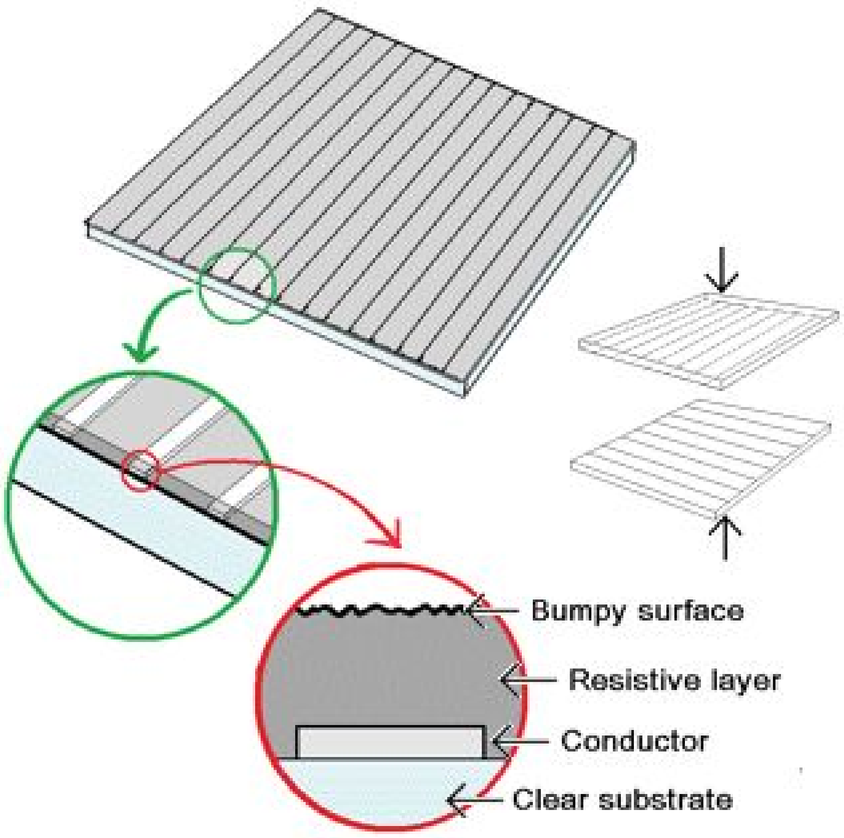


Figure A.3: Microscopic Force-Sensitive Resistance ink. Image courtesy of www.touchco.com.

The article [13] further elaborates that in an IFSR, one layer has evenly spaced vertical wires and the other has evenly spaced horizontal lines. One vertical wire in the first layer is powered up while the others are switched off. Each of the horizontal wires on the other layer are measured in turn. Then the next vertical wire is powered up and the horizontal wires are measured again. This is repeated for each of the vertical wires. The set of measurements from each combination of wires yields a ”pressure image” of the forces applied to the device, with data points at every wire intersection on the surface. The whole process is repeated many times per second. See figure A.4 for an image of how this surface is built.

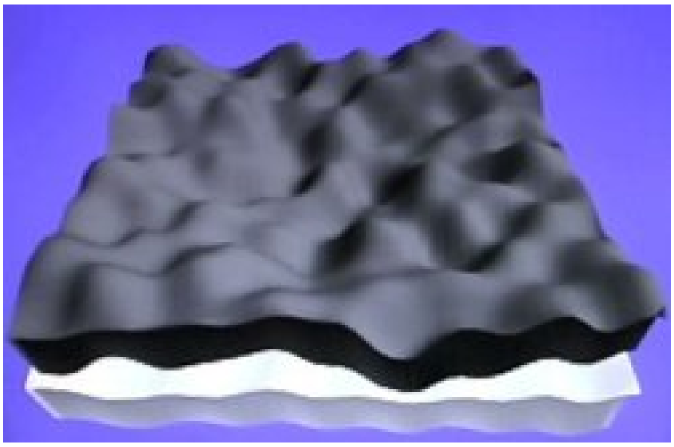


Figure A.4: Layers in Interpolating Force-Sensitive Resistance system. Image courtesy of www.touchco.com.

Although neither of these to multi-touch systems are commercially available yet, we will in the future see more resistive screens capable of multi-touch.

A.2 Capacitive multi-touch

Capacitive touch in contrast to resistive touch does not require the user to apply force when using a touch screen. As described in [26] capacitive sensors are non contact devices capable of high-resolution measurement of the position and/or change of position of any conductive target. Capacitive sensors use the electrical property of ”capacitance” to make measurements. Capacitance is a property that exists between any two conductive surfaces within some reasonable proximity. Changes in the distance between the surfaces changes the capacitance. It is this change of capacitance that capacitive sensors use to indicate changes in position of a target.

A capacitive touch screen consists of an insulator such as glass, coated with a transparent conductor such as indium tin oxide (ITO). As the human body is also a conductor, touching the surface of the screen results in a distortion of the local electrostatic field, measurable as a change in capacitance. Usually capacitive (multi-) touch surfaces can be divided into two classes, *Surface Capacitance* and *Projected Capacitance* .

A.2.1 Surface capacitance

As described in [25] surface-capacitive screens use a plain ITO layer with a metalized border pattern. Se figure A.5 for an illustration. Requiring no sophisticated ITO pattern, the electric field is approximately linear across the ITO. When a finger touches the screen, it bleeds charge from the panel, and sensing comes from the four corners. Surface-capacitive screens cannot discern more than one touch at a time, so this technique is not very interesting when working with multi-touch applications.

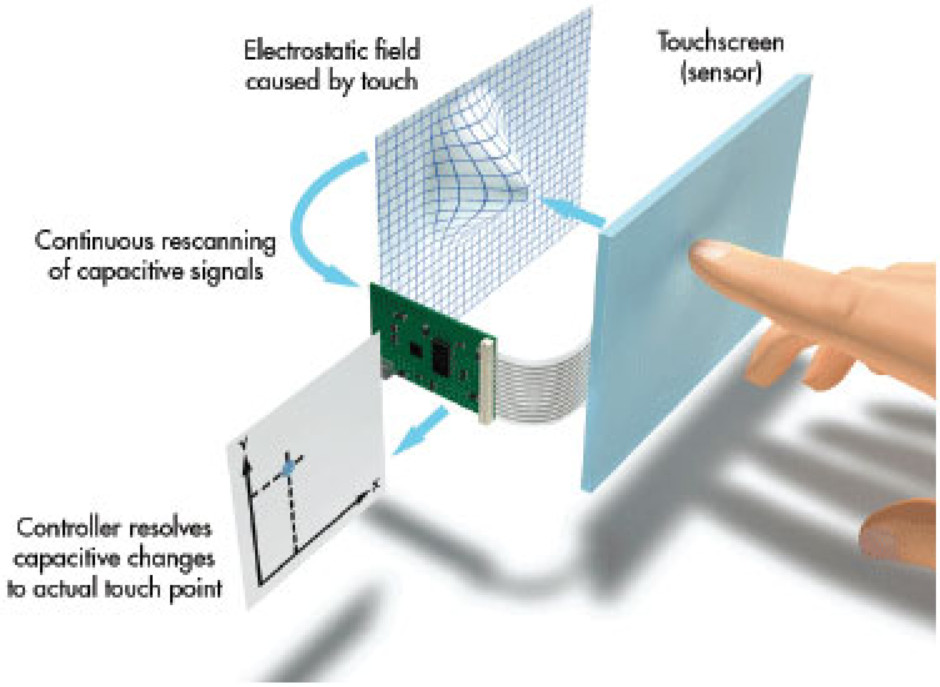

Electrostatic field caused by touch
Touchscreen (sensor)
Continuous rescanning of capacitive signals
Controller resolves capacitive changes to actual touch point
Y
X

Figure A.5: Illustration of how surface capacitance works. Image courtesy of www.electronicdesign.com.

A.2.2 Projected capacitance

Projected capacitive technology is a capacitive technology which permits more accurate and flexible operation, by etching one or more conductive layers forming multiple horizontal and vertical electrodes, which derive drive from a sensing chip. This chip can offload data to a processor. When touched, capacitance forms between the finger and the sensor grid and the touch location can be computed based on the measured electrical characteristics of the grid layer. See figure A.6. The capacitance change at every individual point on the grid can be measured to accurately determine the touch location. The use of a grid permits a higher resolution than resistive technology and also allows multi-touch operation. The greater resolution of projected capacitive touch allows operation without direct contact, such that the conducting layers can operate under screen protectors, or behind glass. Incidentally, this is the technology used in newer handheld devices such as Apples iPhone and other devices supporting multi-touch.

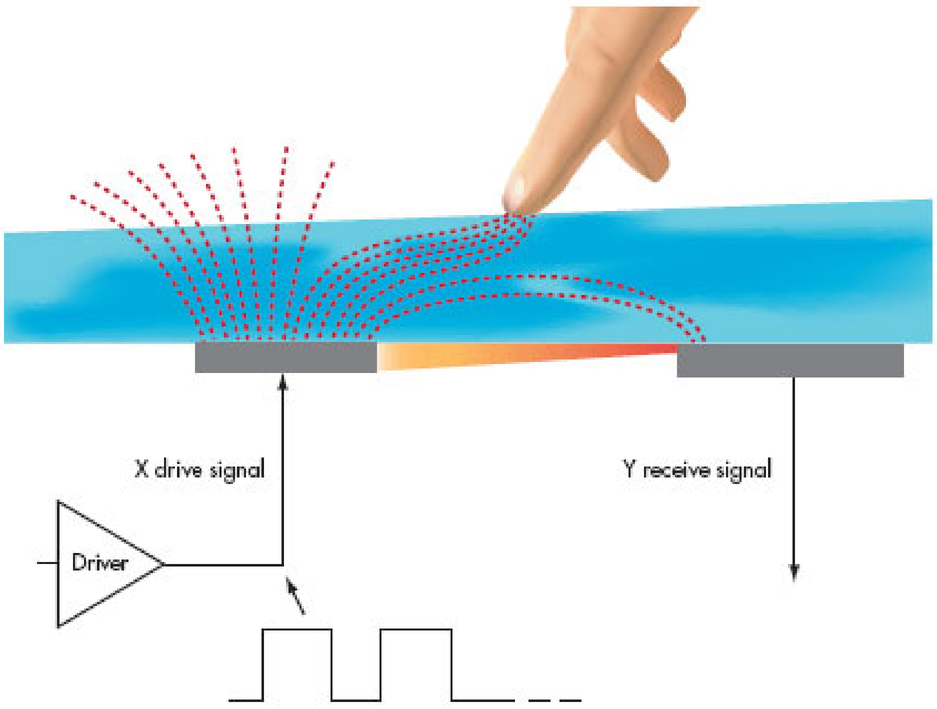


Figure A.6: Illustration of how projected capacitance work. Image courtesy of www.electronicdesign.com.

A.3 Other touch technologies

It is interesting just to mention other touch technologies for comparison and to get a as complete reference as possible. The technologies presented here is at this moment not capable of multi-touch, and is therefore not in the primary scope of this project. We have five well known touch technologies in addition to optical camera-based, resistive and capacitative touch methods. These touch technologies are presented exactly as they appear in [35].

A.3.1 Surface acoustic wave

"Surface acoustic wave (SAW) sumit technology uses ultrasonic waves that pass over the touchscreen panel. When the panel is touched, a portion of the wave is absorbed. This change in the ultrasonic waves registers the position of the touch event and sends this information to the controller for processing. Surface wave touch screen panels can be damaged by outside elements. Contaminants on the surface can also interfere with the functionality of the touchscreen."

A.3.2 Strain gauge

"In a strain gauge configuration, also called force panel technology, the screen is spring-mounted on the four corners and strain gauges are used to determine deflection when the screen is touched. This technology has been around since the 1960s but new advances by Vissumo and F-Origin have made the solution commercially viable. It can also measure the Z-axis and the force of a person's touch. Such screens are typically used in exposed public systems such as ticket machines due to their resistance to vandalism."

A.3.3 Acoustic pulse recognition

"This system, introduced by Tyco International's Elo division in 2006, uses more than two piezoelectric transducers located at some positions of the screen to turn the mechanical energy of a touch (vibration) into an electronic signal. The screen hardware then uses an algorithm to determine the location of the touch based on the transducer signals. This process is similar to triangulation used in GPS. The touchscreen itself is made of ordinary glass, giving it good durability and optical clarity. It is usually able to function with scratches and dust on the screen with good accuracy. The technology is also well suited to displays that are physically larger. As with the Dispersive Signal Technology system, after the initial touch, a motionless finger cannot be detected. However, for the same reason, the touch recognition is not disrupted by any resting objects."

A.3.4 Infrared photosensor

“Conventional infrared touch systems use an array of infrared (IR) light-emitting diodes (LEDs) on two adjacent bezel edges of a display, with photo-sensors placed on the two opposite bezel edges to analyze the system and determine a touch event. The LED and photo-sensor pairs create a grid of light beams across the display. An object (such as a finger or pen) that touches the screen interrupts the light beams, causing a measured decrease in light at the corresponding photo-sensors. The measured photo-sensor outputs can be used to locate a touch-point coordinate.”

A.3.5 Dispersive signal technology

“Introduced in 2002 by 3M, this system uses sensors to detect the mechanical energy in the glass that occurs due to a touch. Complex algorithms then interpret this information and provide the actual location of the touch.[10] The technology claims to be unaffected by dust and other outside elements, including scratches. Since there is no need for additional elements on screen, it also claims to provide excellent optical clarity. Also, since mechanical vibrations are used to detect a touch event, any object can be used to generate these events, including fingers and stylus. A downside is that after the initial touch, the system cannot detect a motionless finger.”

Appendix B

Camera-based multi-touch technologies

This appendix lists and explains the most common camera-based multi-touch technologies known today.

B.1 Frustrated Total Internal Reflection

Frustrated Total Internal Reflection (FTIR) is according to [31] considered as the start of the optical based multi-touch techniques. FTIR was first described in [11] as a low cost solution for building multi-touch capable screens. FTIR is based on the well known optical phenomena of Total Internal Reflection(TIR), which is an effect that occurs when light strikes a boundary between two mediums with different optical density. If the light travels from the optically more dense material, and the angle of the light is large enough, the light rays will be reflected inside the denser material so that no light escapes the material. The specific angle at which this occurs depends on the refractive indexes of both materials, and is known as the critical angle, which can be calculated mathematically using Snell's law.

By taking advantage of the TIR effect between glass and air, we will be able to flood the inside of the glass with trapped light. According to the original paper [11], acrylic is among the best materials to use building a screen based on FTIR. When the user touches the acrylic, the light escapes and is reflected at the finger's point of contact due to its higher refractive index. As illustrated in figure B.1, the frustrated light will then be directed downward, and a camera will be able to capture these points of light.

When fingers are dry, they will create a bad coupling between the acrylic and the finger, and a poor frustration of light inside the acrylic will occur. To further enhance FTIR it is recommended that a compliant layer is used. This layer can be made of silicone rubber, and acts as a link between your finger and the acrylic to create a greater coupling when pressed. In turn stronger blobs will be produced.

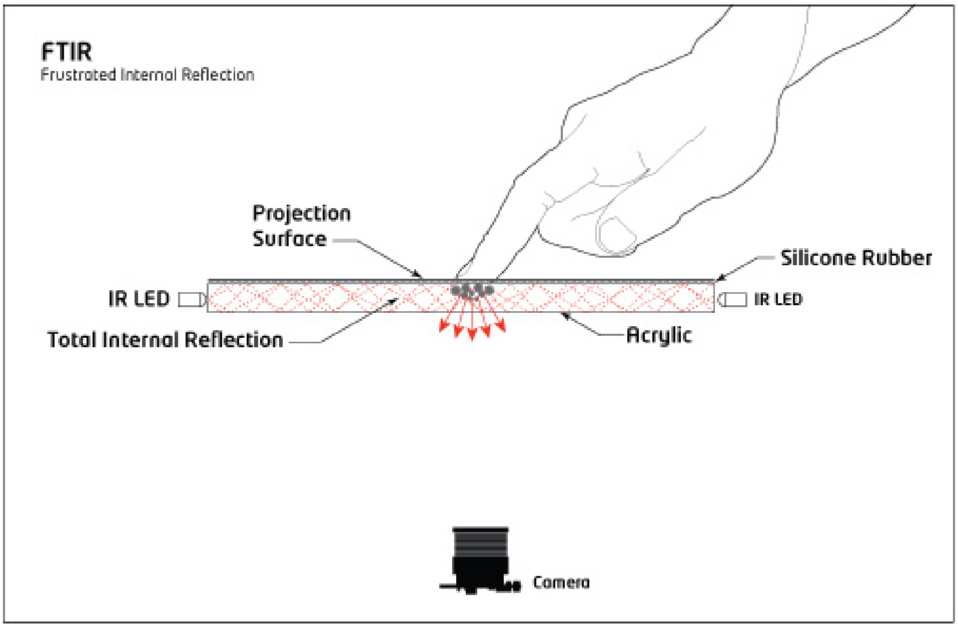


Figure B.1: Frustrated Total Internal Reflection. Original image courtesy of Tim Roth (Image has been modified for this thesis).

Common glass is unsuitable due to its poor optical transmittance, however clearer glass formulations (e.g. ”water white”) can be used successfully. This is a more expensive glass, but such glass is structurally stiffer, and is not as easily scratched as acrylic.

Advantages of FTIR [1]

1. An enclosed box is not required
2. Blobs have strong contrast
3. Allows for varying blob pressure
4. With a compliant surface, it can be used with something as small as a pen tip

[1] List of advantages and disadvantages based on [7]

Disadvantages of FTIR [1]

1. Setup calls for some type of LED frame (soldering required)
2. Requires a compliant surface (silicone rubber) for proper use - no glass surface
3. Cannot recognize objects or fiducial

B.2 Diffused Illumination

As for all the optical solutions mentioned in this document, Diffuse Illumination (DI) also use infrared light, and an infrared sensitive camera to detect blobs. In contrast to FTIR no special glass is needed, because the method relies on shadows or reflections instead of frustrated light. There exist two methods for DI, Front DI and Rear DI. Although Rear DI is the most commonly adopted method, I will explain both of them here.

Front Diffused Illumination Front DI is a very simple technique, and do not require any infrared light source, or a special camera fitted with band pass filter. Front DI use visible light, which often come from the surroundings like windows, and other light sources placed in the room. This light shines at the screen which is fitted with a diffuser. When an object touches the surface, a shadow is created in the position of the object, and a camera underneath the surface senses this shadow. See figure B.2 for an example.

Rear Diffused Illumination In a Rear DI setup the infrared light source is placed behind the projection surface pointing upward. This will illuminate the surface from the non-interactive side, and by placing fingers (or objects) on top of the surface, they will reflect infrared light down towards the camera. See figure B.3 for how this setup works. Depending on the size and configuration of the table, it can be hard to achieve a uniform distribution of light across the surface.

By having a diffuser on the top or the bottom of the surface, we will be able to project an image from a projector. This diffuser can double as a projection surface for the projector, and as a diffuser for the light coming from the light sources behind the surface. Depending of the properties of this diffuser in regard to how much IR light is transmitted and diffused, objects close to or touching the surface will create bright spots in the image captured by the camera.

In contrast to FTIR, Rear DI allows tracking and identification of objects as well as fingers, and even hovering objects will be possible to ”see”. Objects can be identified using their shape or fiducial printed on their bottom surfaces.

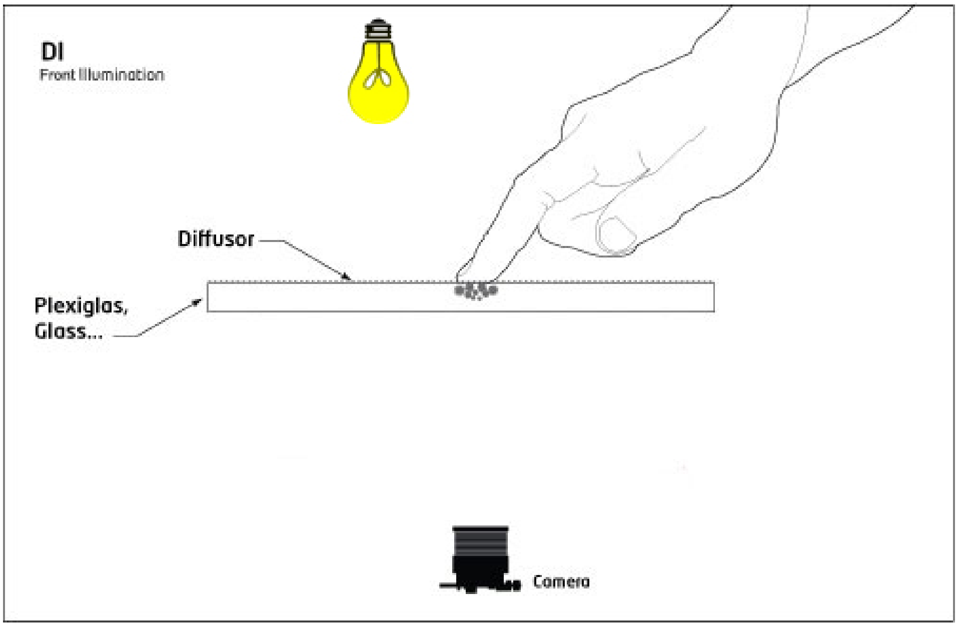


Figure B.2: Front Diffused Illumination. Original image courtesy of Tim Roth (Image has been modified for this thesis).

Advantages of Front DI [2]

1. No need for a compliant surface, just an diffuser/projection surface on top/bottom
2. Can use any transparent material like glass (not just acrylic)
3. No LED frame required
4. No soldering (you can buy the IR-Illuminators ready to go)
5. Can track fingers and hovering
6. An enclosed box is not required
7. Simplest setup

Disadvantages of Front DI [2]

1. Cannot track objects and fiducials

[2]List of advantages and disadvantages based on [7]

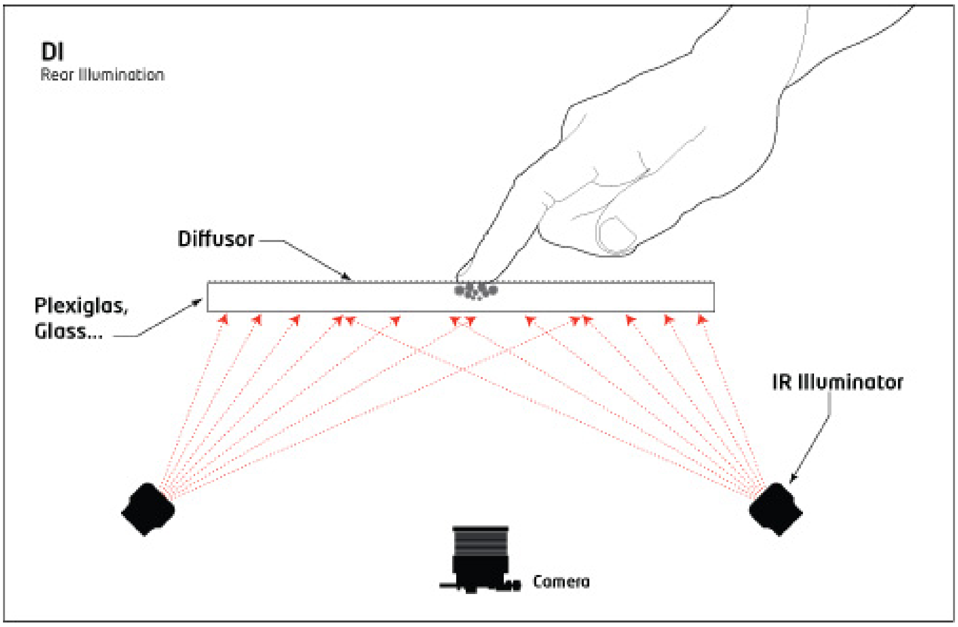


Figure B.3: Rear Diffused Illumination. Original image courtesy of Tim Roth (Image has been modified for this thesis).

2. Difficult to get even illumination
3. Greater chance of *false blobs*
4. Not as reliable (Heavily dependent on ambient light in the environment)
5. People standing around the table, blocking the ambient light.

Advantages of Rear DI [3]

1. No need for a compliant surface, just an diffuser/projection surface on top/bottom
2. Can use any transparent material like glass (not just acrylic)
3. No LED frame required
4. No soldering (you can buy the IR-Illuminators ready to go)
5. Can track objects, fingers, fiducials, hovering

[3] List of advantages and disadvantages based on [7]

Disadvantages of Rear DI [3]

1. Difficult to get even illumination
2. Blobs have lower contrast (harder to pick up by software)
3. Greater chance of *false blobs*
4. Enclosed box is required

B.3 Diffused Surface Illumination

Diffused surface illumination (DSI) was invented by *Tim Roth* as a response to the problem with uniform distribution of light across the surface using standard DI methods. In [30] *Tim Roth* presents the DSI technique in detail. Instead of using Illuminators as an IR Source, a special acrylic glass is used to distribute the light evenly across the surface.

This special glass is called PLEXIGLAS EndLighten, and contains thousands of micro mirrors. When you shine IR light into the edges of this material, the light gets redirected and spread to the surface of the glass. This technique will behave in the same way as DI when a finger touches the surface. Depending of the properties of the diffuser in regard to how much IR light is transmitted and diffused, objects close to or touching the surface will create bright spots in the image captured by the camera. Se figure B.4 for an illustration of the concept.

This method is kind of similar to building a FTIR-setup, except that you replace the regular acrylic glass with EndLighten.

Advantages of DSI [4]

1. No compliant surface (silicone)
2. No problems getting an even distribution of light, easy to set up
3. An FTIR set-up can be converted to DSI easily
4. No need to build a special case for the set-up (like other DI set-ups)
5. Fiducial tracking is possible
6. Is pressure sensitive
7. No hotspots

[4] List of advantages and disadvantages based on [7] and [30]

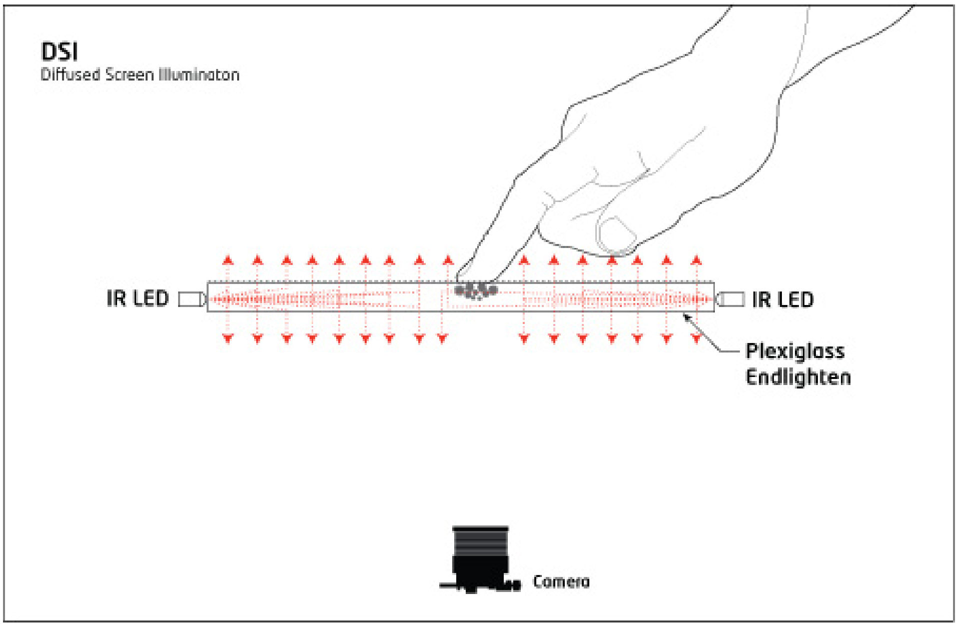


Figure B.4: Diffused Surface Illumination. Original image courtesy of Tim Roth (Image has been modified for this thesis).

Disadvantages of DSI [4]

1. Endlighten Acrylic costs more than regular acrylic (offset by the fact that no IR illuminators are needed)

2. Less contrast compared to normal DI set-ups as the surface material also redirects the IR toward the camera

3. Potentially more problems with ambient IR because of less contrast

4. Possible size restrictions because of the softness of the surface material

B.4 Laser Light Plane

Laser Light Plane (LLP) is a technology that utilizes a different technique for the light source than FTIR and DI. As the name suggests, this technique use one or more infrared lasers to create a plane of light. This light plane resides on top of the screen surface, and does not depend on what type of material the surface is made of. As for the other optical based techniques, this technique also use an infrared sensitive camera underneath the screen surface to capture the blobs. See figure B.5 for a visual picture of this technique.

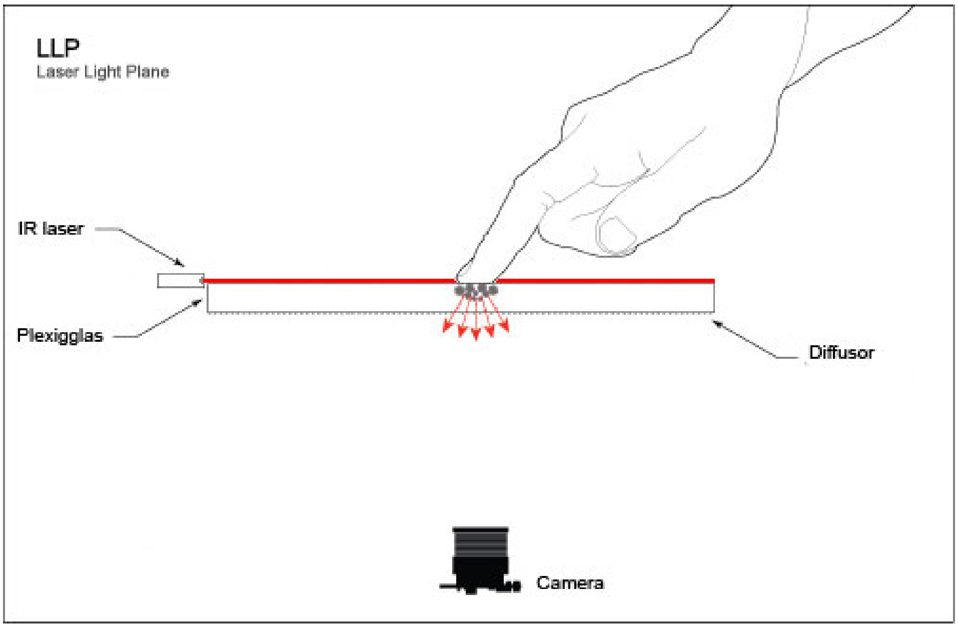


Figure B.5: Laser Light Plane. Original image courtesy of Tim Roth (Image has been modified for this thesis).

LLP requires only one laser, but it is recommended to use at least two lasers to avoid occlusion. Builds that require many simultaneous touches often use four lasers, one in each corner. Normally lasers create a straight line of light, so to make the lasers project a plane, you can put a line generator lens on it to spread the light out in a plane.

The light plane is projected on top of the surface, and when a finger is touching the screen, it will cross the laser light plane, generating a diffuse reflection that can be picked up by the camera as a blob. Pressure is not needed to cross the laser beam, so it is a true zero-force setup, which mean that even the lightest touch will produce a nice blob. One of the disadvantages of LLP is that it does not support recognition of objects, as the light comes from the sides instead of from underneath the surface.

Advantages of LLP [5]

1. No compliant surface (silicone)

[5]List of advantages and disadvantages based on [7]

2. Can use any transparent material like glass (not just acrylic)
3. No LED frame required
4. An enclosed box is not required
5. Simple setup
6. Could be slightly cheaper than other techniques

Disadvantages of LLP [5]

1. Cannot track traditional objects and fiducials
2. Not truly pressure sensitive (since light intensity does not change with pressure)
3. Can cause occlusion if only using 1 or 2 lasers where light hitting one finger blocks another finger from receiving light.

B.5 LED Light Plane

LED-LP is a technique that combines some ideas from LLP and some from FTIR. As described in [7] LED-LP is setup the same way as an FTIR setup except that the thick acrylic that the infrared light travels through is removed and the light travels over the touch surface in a LLP similar fashion. Figure B.6 show this concept. The infrared LEDs are placed around the touch surface, with all sides preferably being surrounded to get a more even distribution of light. There are baffles on the top and bottom of the LEDs to hinder light spill downward through the surface, and only keep the light rays going fairly straightforward. Since the light coming from the LEDs is conical instead of a flat laser plane, the light will light up objects placed above the touch surface.

This method was created by users at *Natural user interface group*. The goal was to create a cheap alternative to LLP and FTIR for people that are interested in multi-touch but do not have the funding. LED-LP is usually only recommended when working with a LCD screen. There are better methods such as Rear DI when using a projector.

Advantages of LED LP [6]

1. No compliant surface (silicone)

[6]List of advantages and disadvantages based on [7]

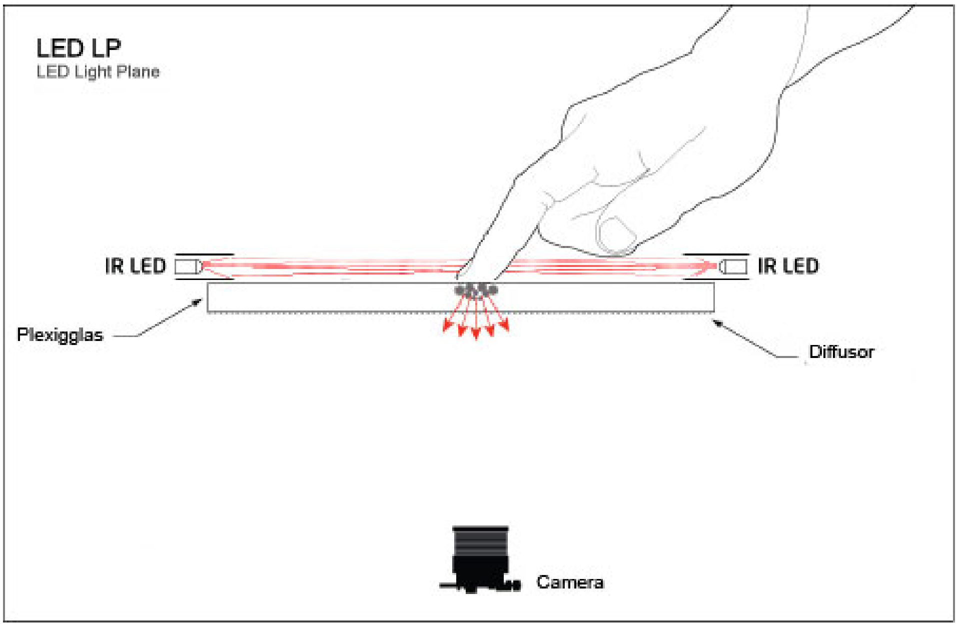


Figure B.6: LED Light Plane. Original image courtesy of Tim Roth (Image has been modified for this thesis).

2. Can use any transparent material like glass (not just acrylic)
3. An enclosed box is not required
4. Could be slightly cheaper than other techniques

Disadvantages of LED LP [6]

1. Hovering might be detected
2. Does not track objects or fiducials
3. LED frame (soldering) required
4. Only narrow-beam LEDs can be used - no ribbons

Appendix C

Test sets

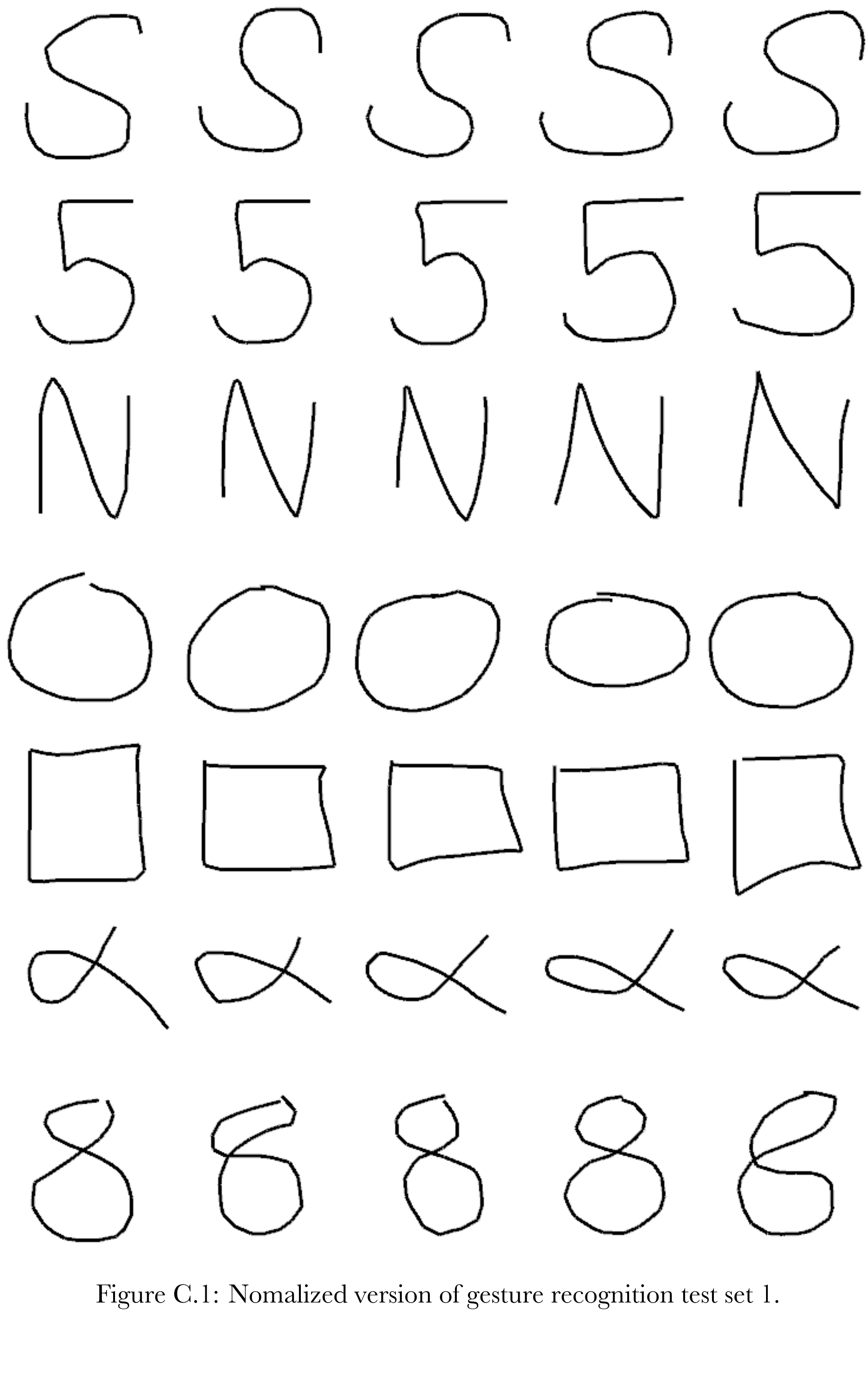


Figure C.1: Nomalized version of gesture recognition test set 1.

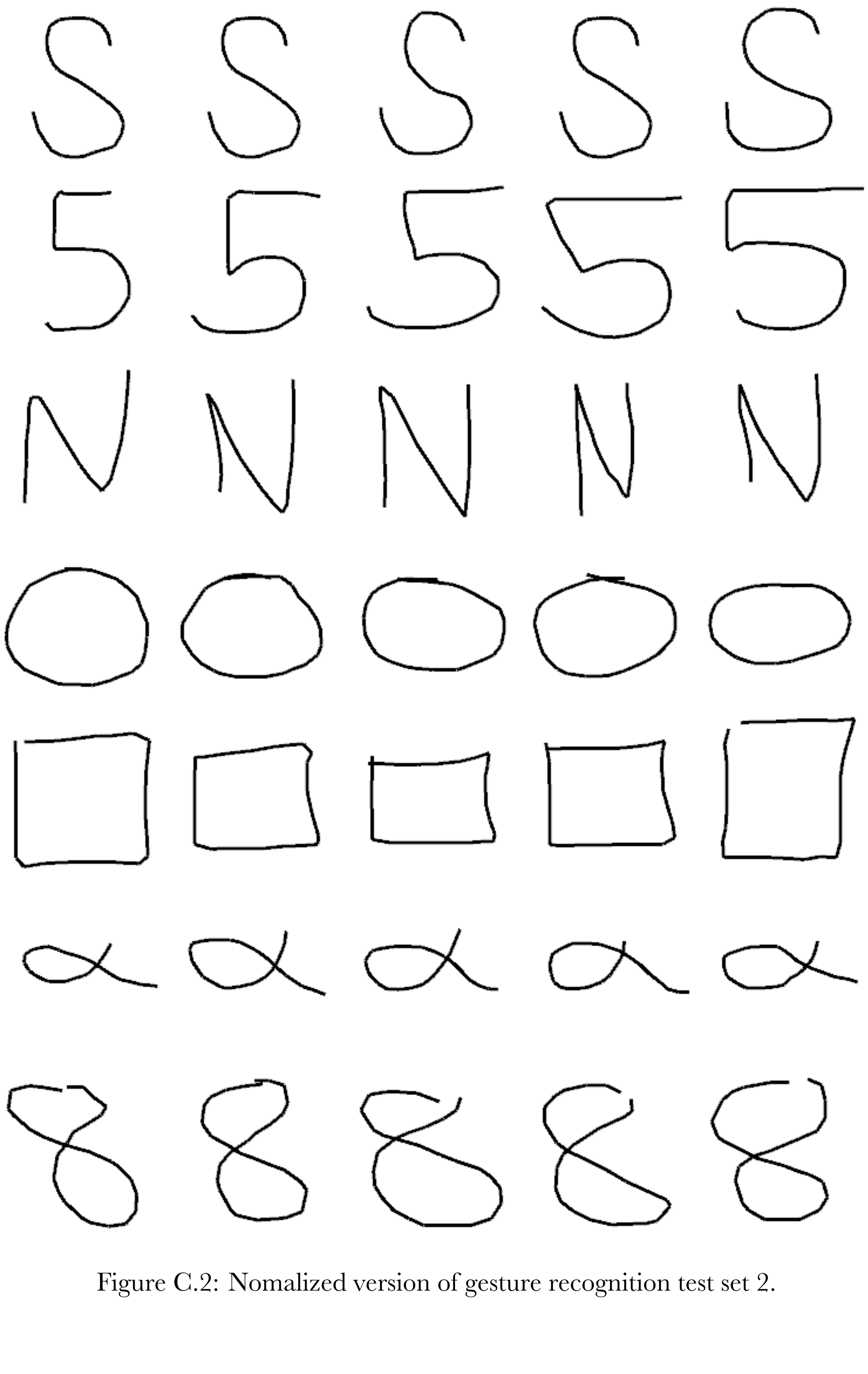


Figure C.2: Nomalized version of gesture recognition test set 2.

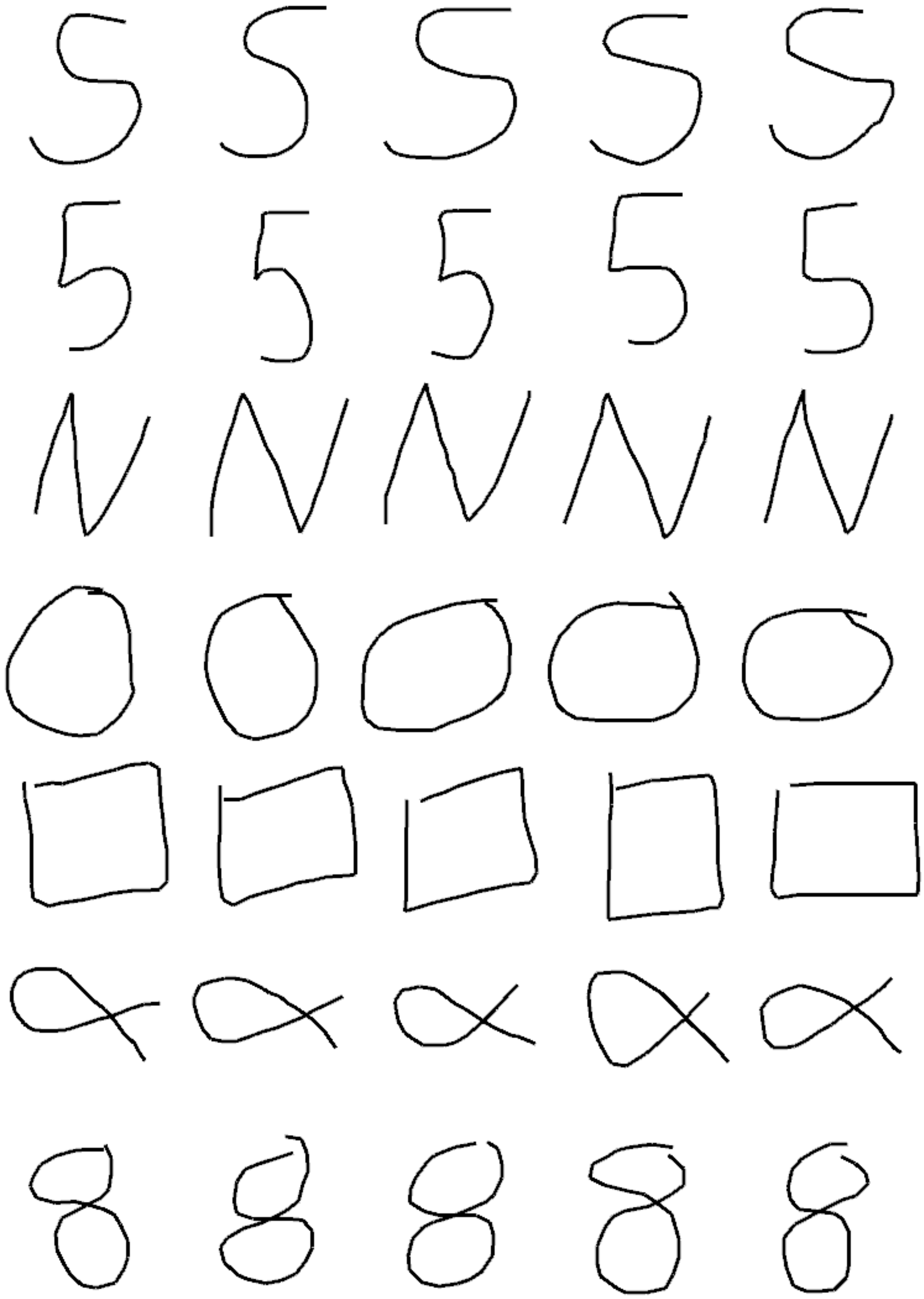


Figure C.3: Nomalized version of gesture recognition test set 3.

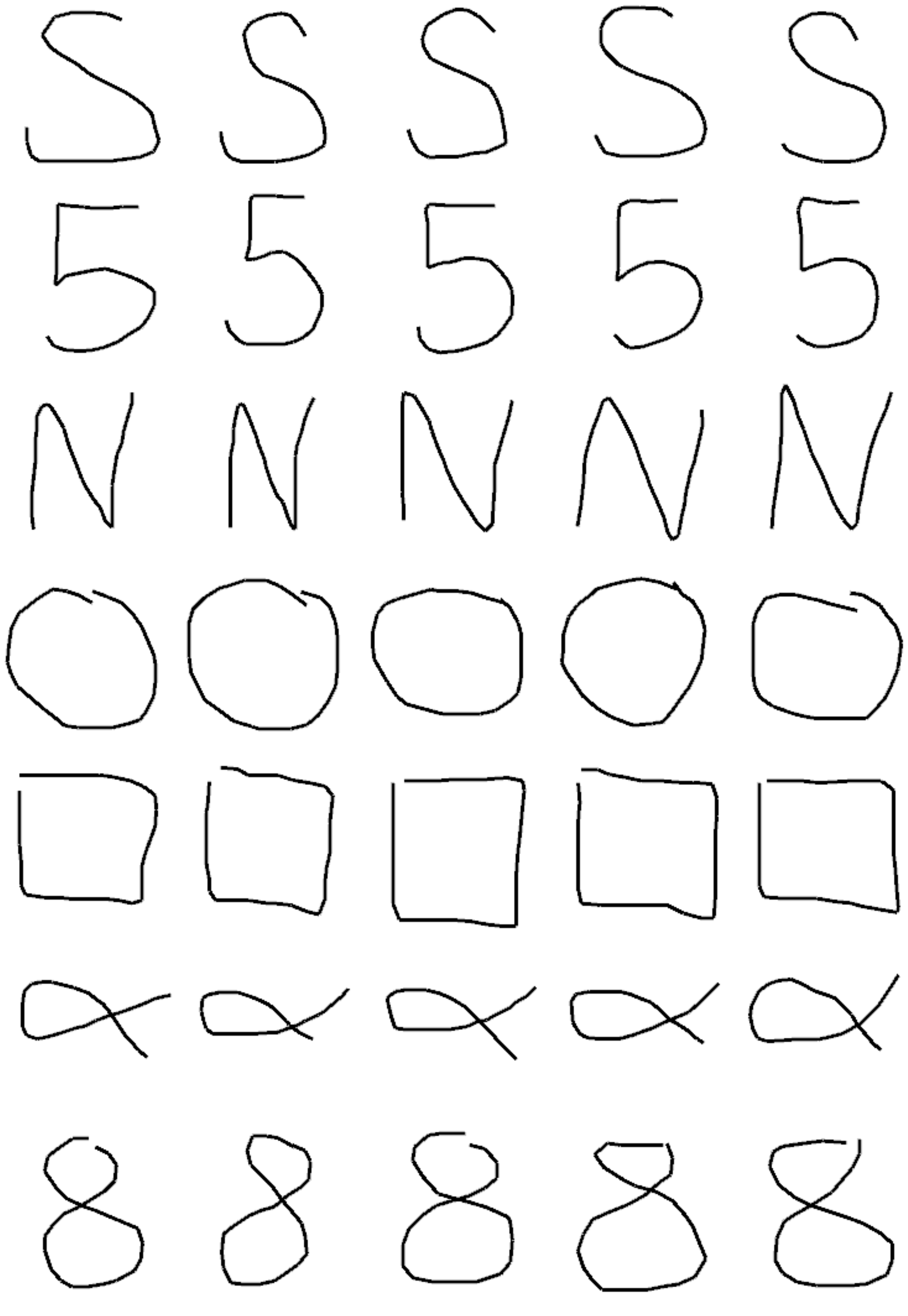


Figure C.4: Nomalized version of gesture recognition test set 4.

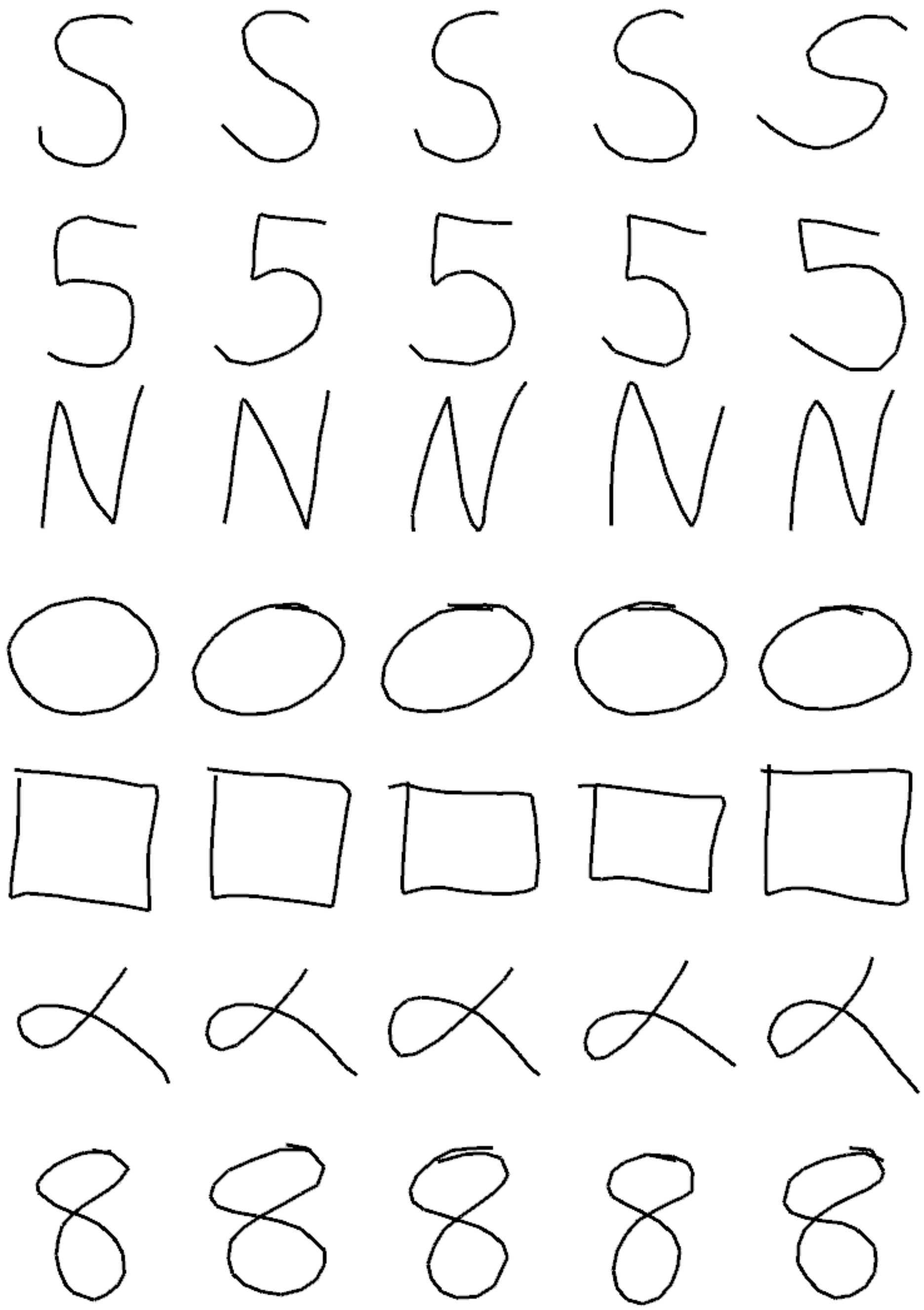


Figure C.5: Nomalized version of gesture recognition test set 5.

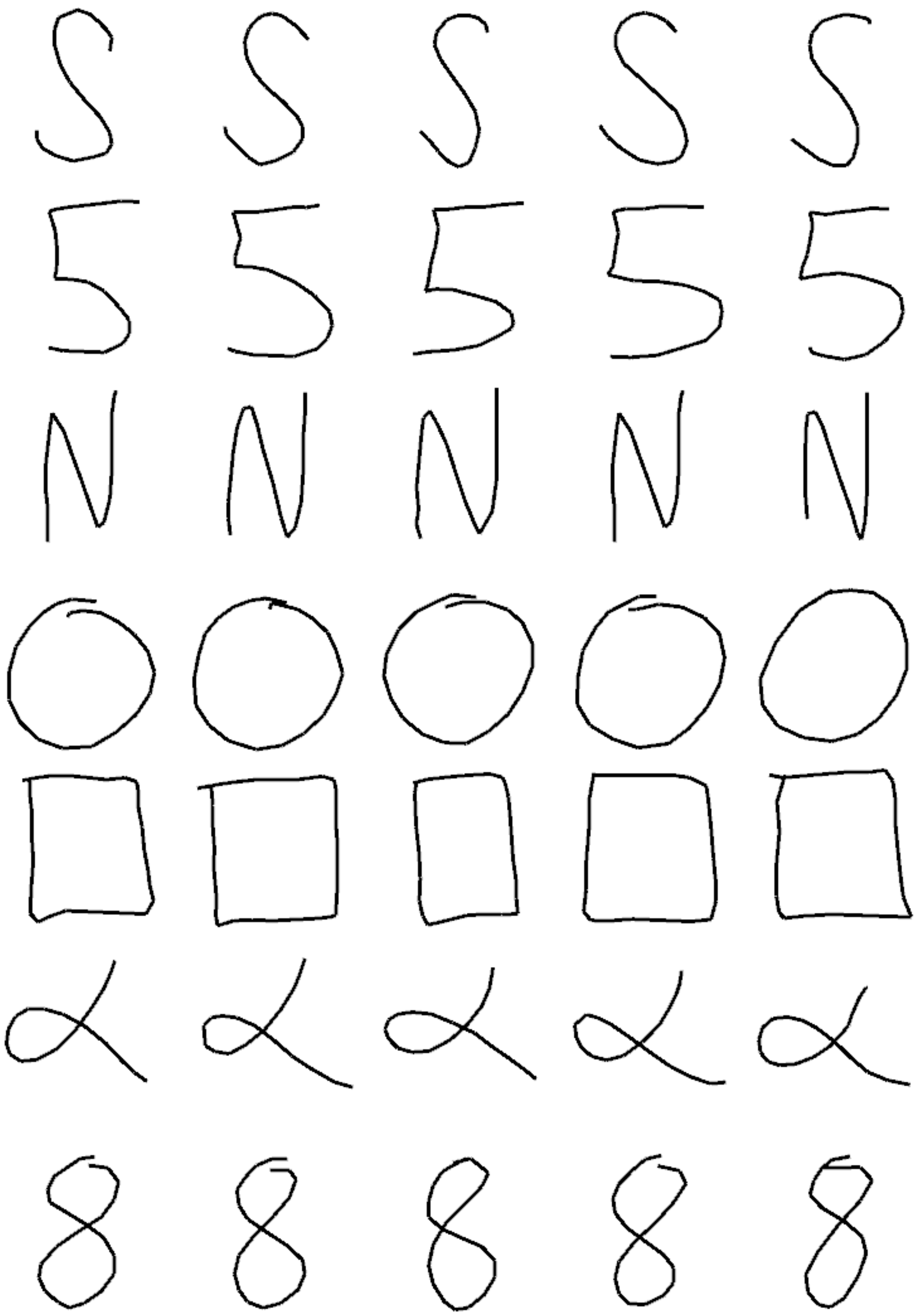


Figure C.6: Nomalized version of gesture recognition test set 6.

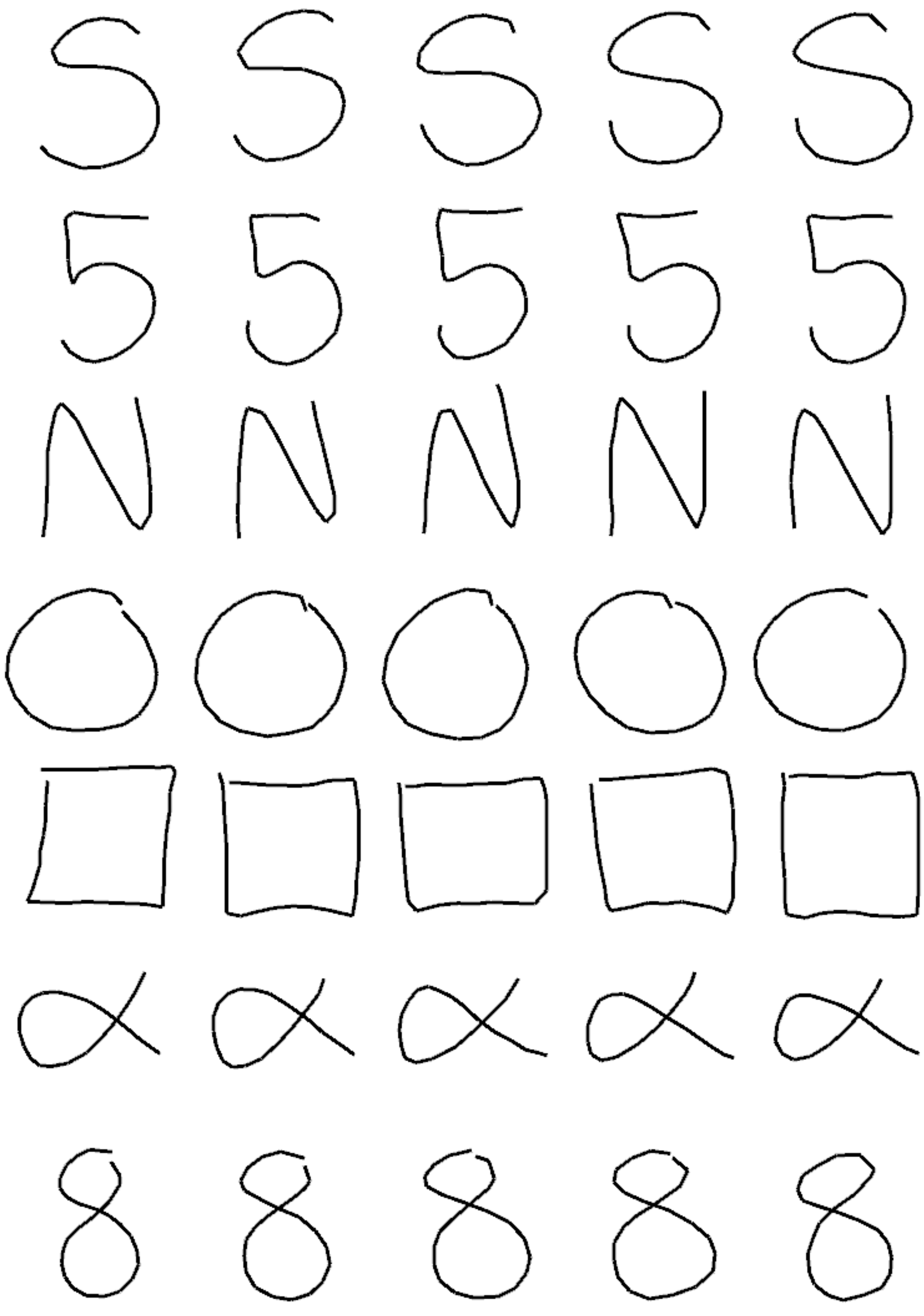


Figure C.7: Nomalized version of gesture recognition test set 7.

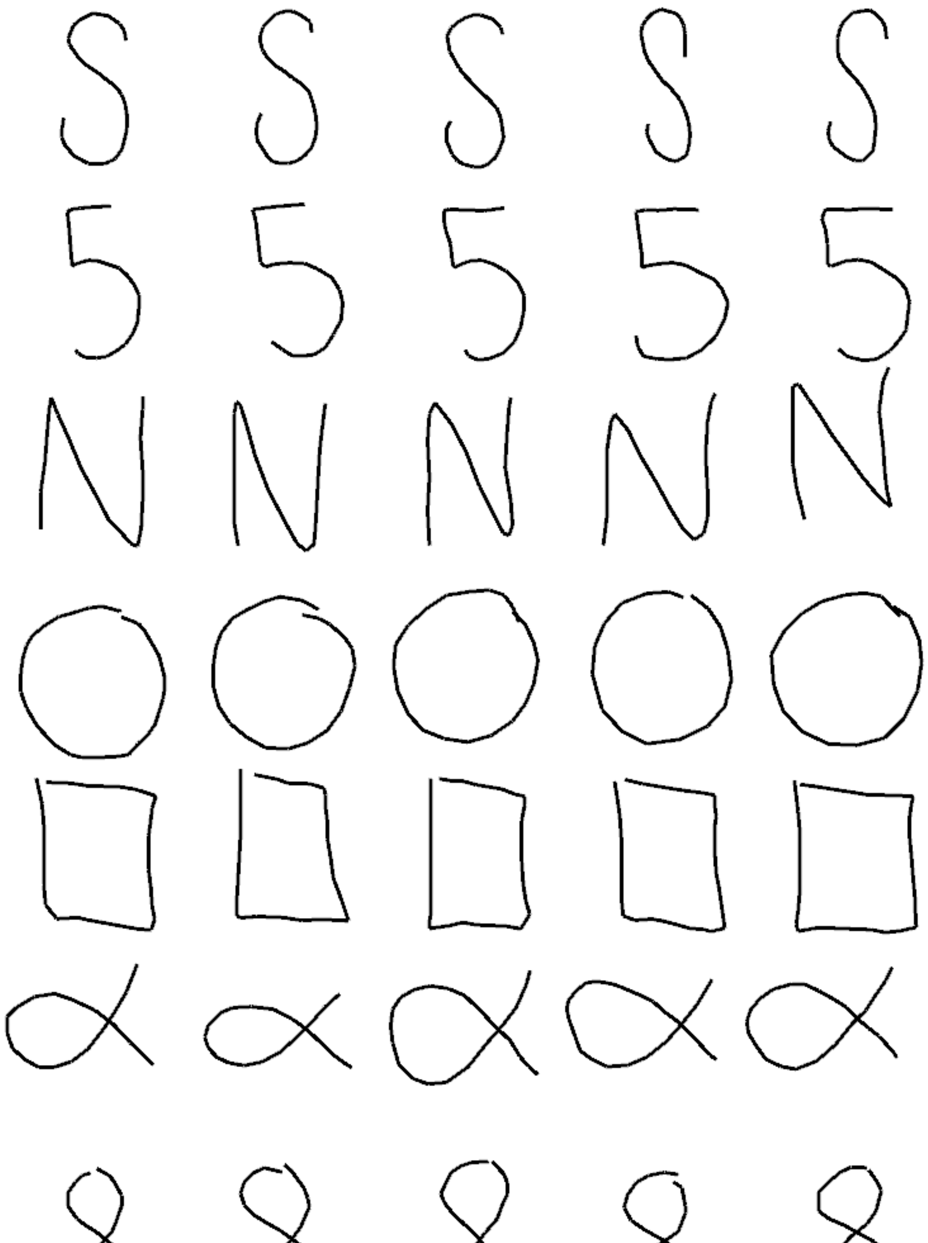


Figure C.8: Nomalized version of gesture recognition test set 8.

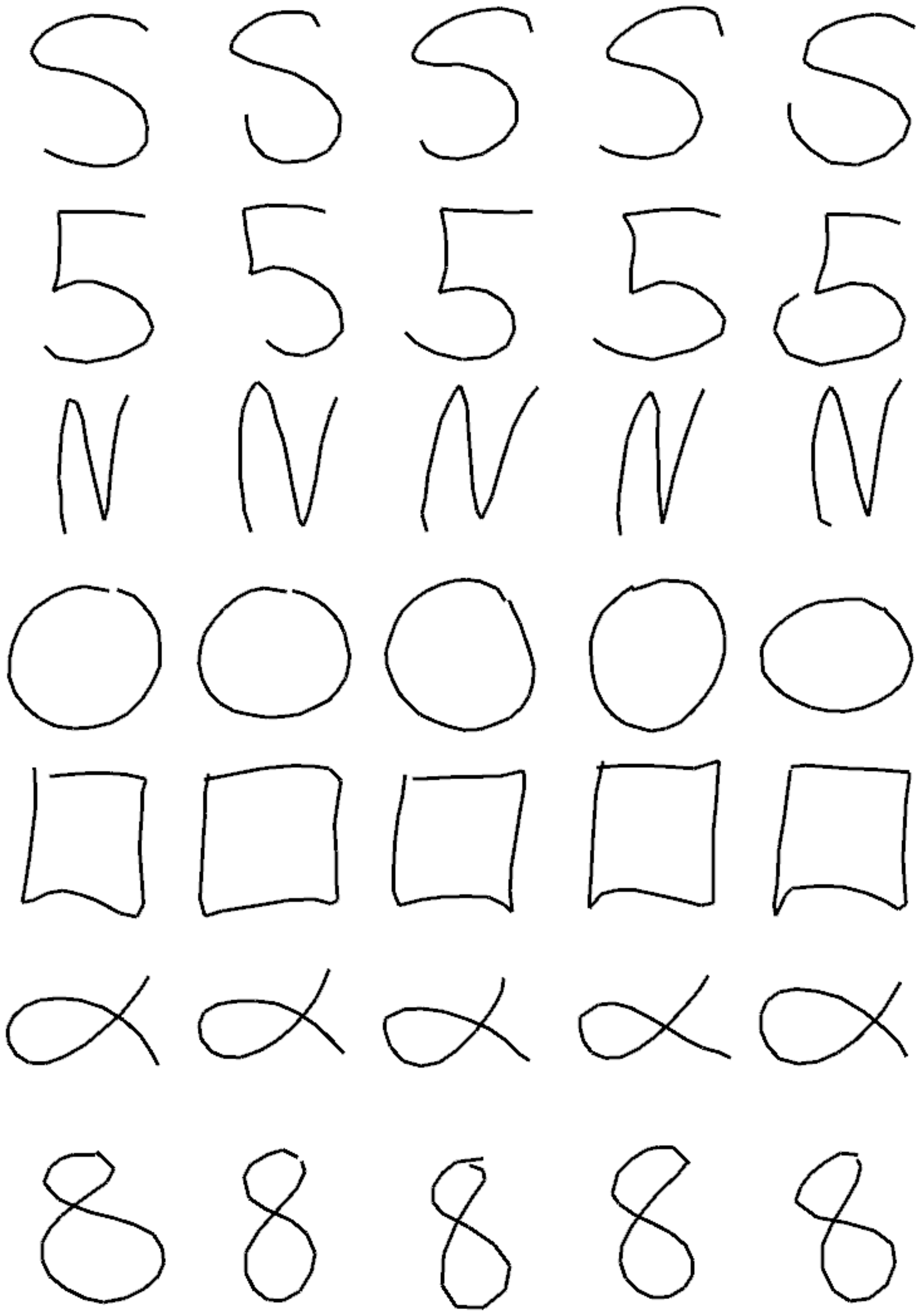


Figure C.9: Nomalized version of gesture recognition test set 9.

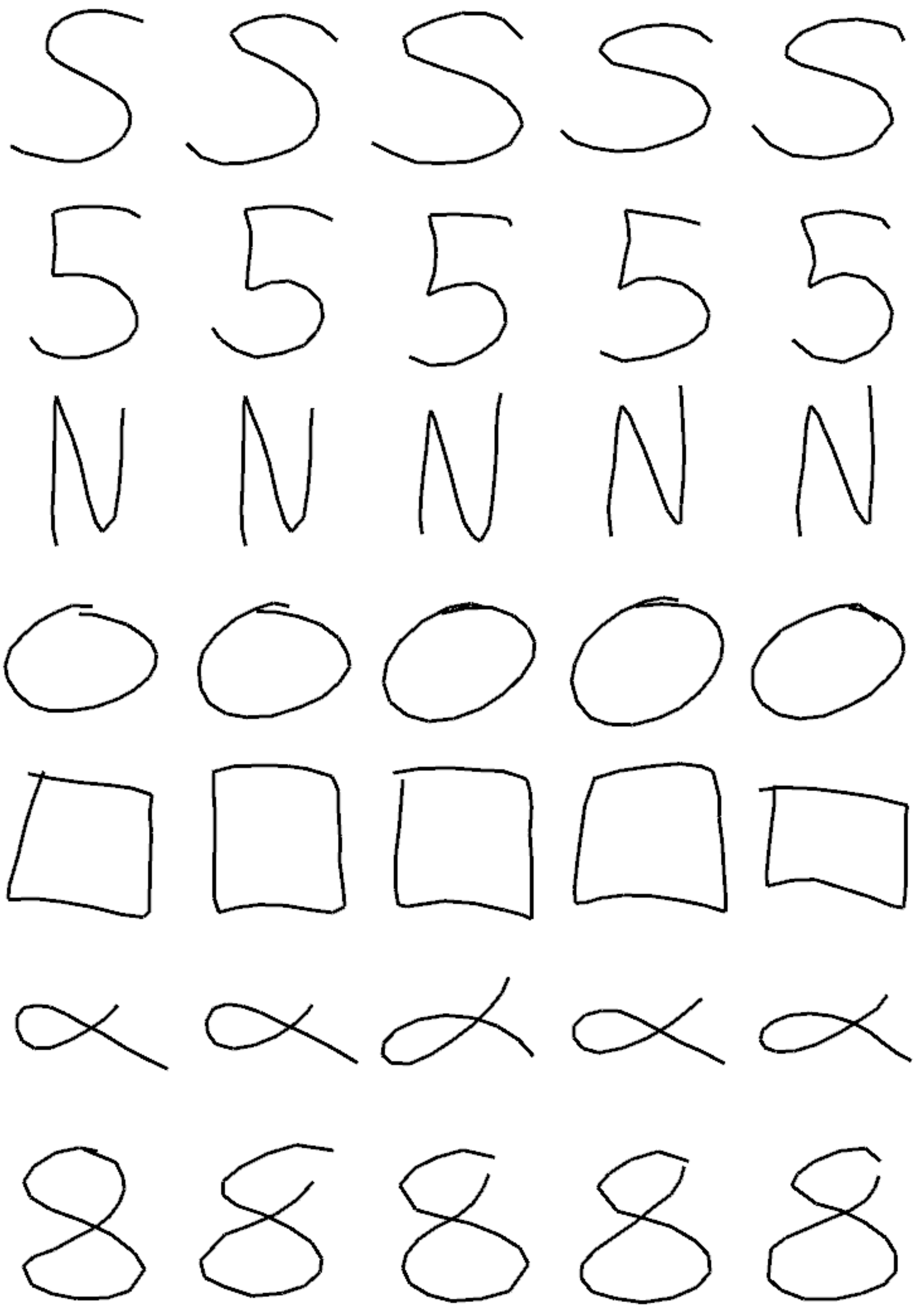


Figure C.10: Nomalized version of gesture recognition test set 10.

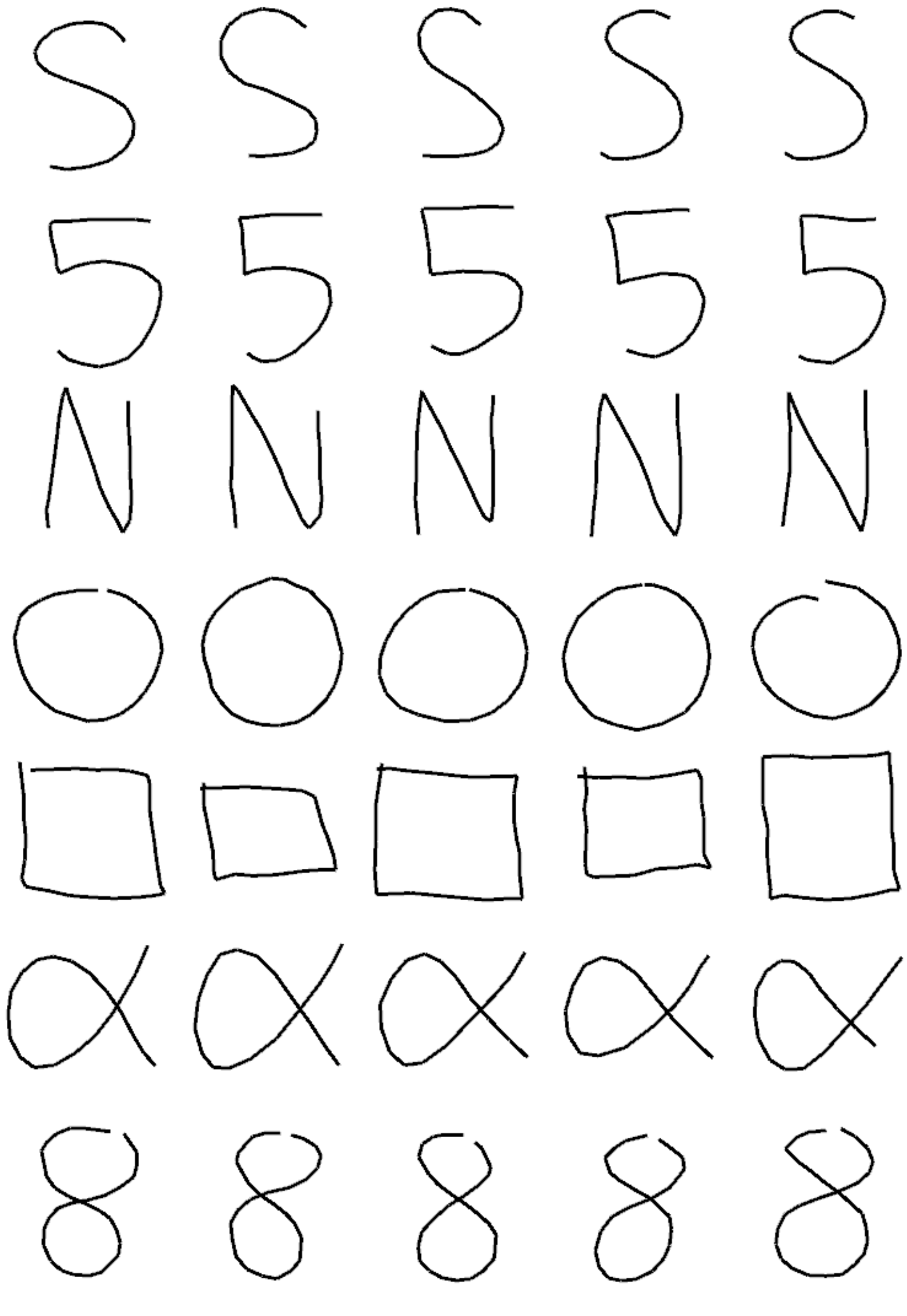


Figure C.11: Nomalized version of gesture recognition test set 11.

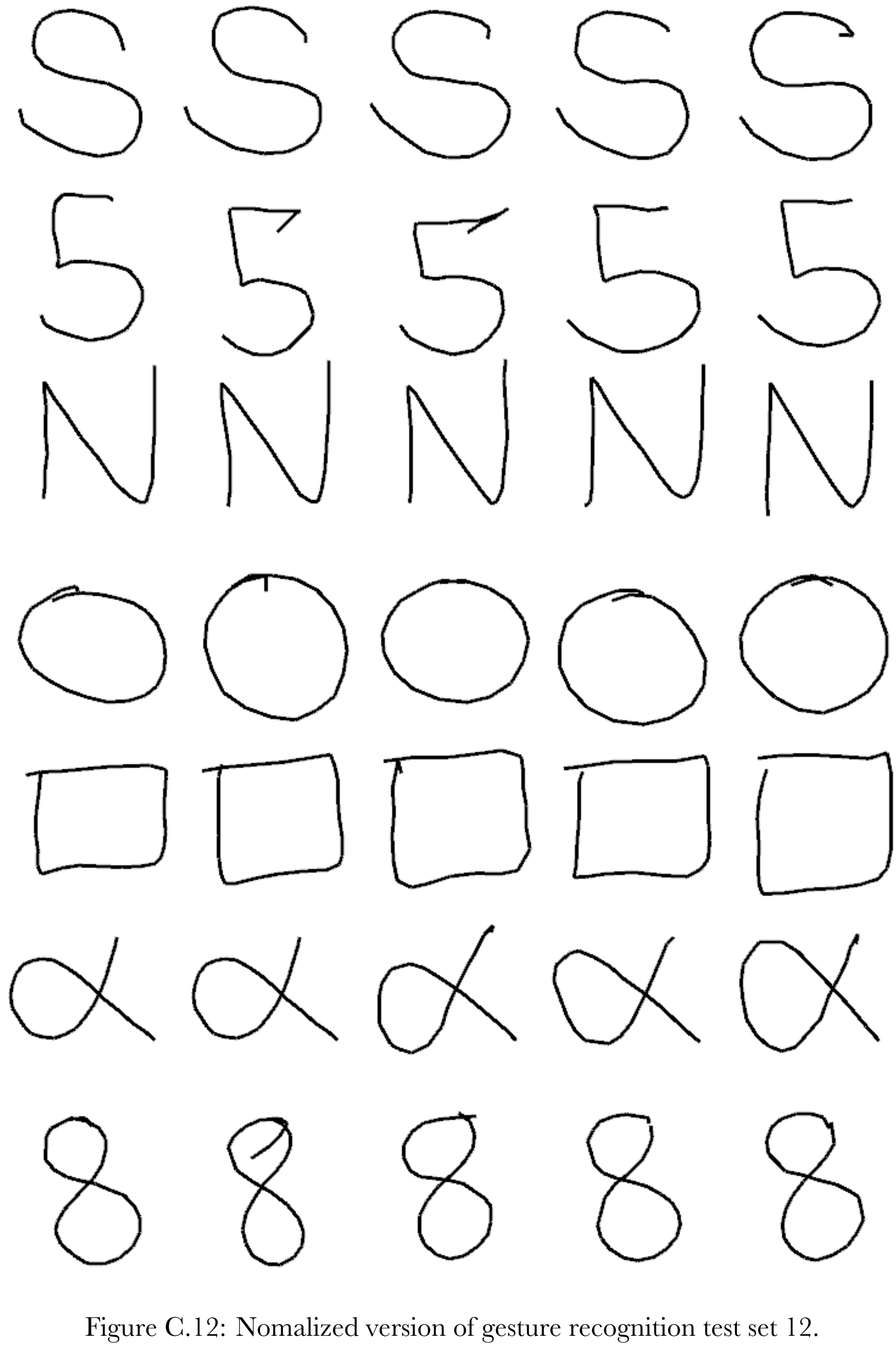


Figure C.12: Nomalized version of gesture recognition test set 12.

Bibliography

[1] Evoluce AG. Evoluce. *http://www.evoluce.com/*, 2010.

[2] Evoluce AG. Evoluce one. *http://www.evoluce.com/downloads/docs/Multitouch-LCD-Evoluce-ONE.pdf*, 2010.

[3] D. Anderson, C. Bailey, and M. Skubic. Hidden Markov Model symbol recognition for sketch-based interfaces. *Making Pen-Based Interaction Intelligent and Natural*, pages 15–21, 2004.

[4] C.M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, USA, 1995.

[5] Greg Brill. Gregs rules of surface and ui design. *http://www.infusion.com/surfaceblog/index.php/gregs-rules-of-surface-and-ui-design*, 2009.

[6] Natural User Interface Group Community. Natural user interface group. *http://www.nuigroup.com*.

[7] Natural User Interface Group Community. *Multi-Touch Technologies.* NUI Group, 1st, edition, 2009.

[8] Oleg Dopertchouk. Recognition of handwritten gestures. *http://www.gamedev.net/reference/articles/article2039.asp*, 2004.

[9] R. Downs. Using resistive touch screens for human/machine interface.

[10] M. Elmezain, A. Al-Hamadi, J. Appenrodt, and B. Michaelis. A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition. *International Journal of Electrical, Computer, and Systems Engineering*, 3(3), 2009.

[11] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, New York, NY, USA, 2005. ACM.

[12] Stantum Inc. Multi-touch using resistive touch. *http://www.stantum.com*, 2009.

[13] Touchco Inc. Interpolating force-sensitive resistance. *http://www.touchco.com/tech.html*, 2009.

[14] M. Kaltenbrunner and R. Bencina. reacTIVision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, page 74. ACM, 2007.

[15] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. Tuio - a protocol for table based tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)*, Vannes, France, 2005.

[16] V. Levenshteiti. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[17] M.T. Maybury and W. Wahlster. *Readings in intelligent user interfaces*. Morgan Kaufmann, 1998.

[18] Microsoft. Microsoft surface. *http://www.surface.com*, 2009.

[19] Microsoft. Project natal. *http://www.xbox.com/en-US/live/projectnatal/*, 2009.

[20] Python multi-touch community. Python multi-touch. *http://pymt.txzone.net/*, 2009.

[21] Nesher. Multi-touch vista. *http://multitouchvista.codeplex.com/*, 2010.

[22] K. Oka, Y. Sato, and H. Koike. Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications*, pages 64–71, 2002.

[23] Kenji Oka, Yoichi Sato, and Hideki Koike. Real-time fingertip tracking and gesture recognition. *IEEE Comput. Graph. Appl.*, 22(6):64–71, 2002.

[24] openCV community. Open computer vision library. *http://opencv.willowgarage.com*, 2009.

[25] Hal Philipp. Please touch! explore the evolving world of touchscreen technology. *http://electronicdesign.com/Articles/Index.cfm?ArticleID=18592*, 2009.

[26] Lion Precision. Capacitive sensor operation and optimization. *http://www.lionprecision.com/tech-library/technotes/tech-pdfs/cap-0020-cap-theory.pdf*, 2009.

[27] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSp Magazine*, 3(1 Part 1):4–16, 1986.

[28] Stjepan Rajko. Ame patterns library. *http://ame4.hc.asu.edu/amelia/patterns/*, 2008.

[29] Stjepan Rajko. Amelia alpha. *http://ame4.hc.asu.edu/amelia/*, 2008.

[30] Tim Roth. Dsi - diffused surface illumination. *http://iad.projects.zhdk.ch/multitouch/?p=90*, 2008.

[31] Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi, Laurence Muller, Patrick Olivier, Tim Roth, and Ulrich von Zadow. Multi-touch surfaces: A technical guide. Technical report, Technical University of Munich, October 2008.

[32] B. Signer, U. Kurmann, and MC Norrie. igesture: A general gesture recognition framework. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, 2007.

[33] Natural user interface group community. Community core vision. *http://ccv.nuigroup.com/*, 2009.

[34] Natural user interface group community. Touchlib. *http://nuigroup.com/touchlib/*, 2009.

[35] Wikipedia. Touchscreen technology. *http://en.wikipedia.org/wiki/Touchscreen*, 2009.

[36] Carl D. Worth. xstroke: Full-screen gesture recognition for x. *http://www.usenix.org/xstroke.html*, 2003.