Institut für Informatik

Ludwig Maximilians Universität München

LMU

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit Medieninformatik

Diplomarbeit

A Wall-sized Focus and Context Display

vorgelegt von

Sebastian Boring

Aufgabensteller:	Prof. Dr. Andreas Butz
Betreuer:	Dipl.-Inf. Otmar Hilliges
Beginn:	11. November 2005
Abgabedatum:	29. März 2006

Abstract

Several instrumented environments have been implemented in different research laboratories during the past years. Their objective is to create simulations for ubiquitous computing which can be used to test certain scenarios. This paradigm is descriptive for the idea of computers appearing almost invisible to users while being available on any spot.

For further insight on how to display information in the environments mentioned above, the need of interaction with those emerges. Thanks to the desktop metaphor this is not a problem for pure display devices like computer monitors. For displaying them throughout a whole room, however, new interaction techniques have to be created in order to modify digital information directly.

This thesis is to present such an interaction technique. It will show how an entire wall can serve as one single display on which users can interact with it by hand. To achieve that, several graphical display technologies will be combined to depict information on the wall's surface. Additionally, cameras that enable the interaction are to be installed. Since the size of the wall will allow multiple users to stand in front of it, potentials of multi-user bimodal interaction will be gained.

The combination of divers technologies makes it possible to have the wall act as an interactive surface with different finesses. It enables users to selectively modify detailed information while having an overview of the whole content. The actual combination of technologies remains incognizable to them, resulting in one sole system in their imagination.

Zusammenfassung

Während der letzten Jahre wurden mehrere instrumentierte Umgebungen in verschiedenen Forschungslabors erschaffen. Ziel dieser ist es, Simulationen für *allgegenwärtige Computer* zu geben, in denen bestimmte Szenarien getestet werden können. Dieses Paradigma beschreibt die Fähigkeit von Computern für den Benutzer nahezu unsichtbar und überall verfügbar zu sein.

Eine weitere Frage besteht darin, wie Informationen in solchen Umgebungen angezeigt werden können. Basierend darauf entsteht zusätzlich der Bedarf an Interaktion mit diesen. Auf reinen Anzeigegeräten, wie zum Beispiel Bildschirmen ist dies mit der Desktop-Metapher kein großes Problem. Spricht man jedoch von der Darstellung in ganzen Räumen müssen neue Interaktionsmöglichkeiten geschaffen werden, die es dem Benutzer erlauben, digitale Informationen direkt zu verändern.

Diese Arbeit präsentiert eine solche Interaktionsmöglichkeit. Es wird gezeigt, wie eine gesamte Wand als einziger Bildschirm dienen kann mit dem Benutzer mit ihren bloßen Händen interagieren können. Hierzu werden verschiedene, grafische Ausgabetechnologien kombiniert, damit auf der gesamten Wand Informationen dargestellt werden können. Zusätzlich werden Kameras montiert, welche die Interaktion damit ermöglichen. Da diese Wand durch ihre Größe die Möglichkeit bietet, dass mehrere Benutzer gleichzeitig vor ihr stehen können, werden außerdem Möglichkeiten für eine Vielzahl von Benutzern geschaffen, die wiederum mit beiden Händen arbeiten können.

Durch die Kombination verschiedener Technologien ist es möglich, die gesamte Wand als eine einzige interaktive Fläche zu gestalten. Diese beinhaltet verschiedene Feinheitsgrade der Interaktion und ermöglicht den Benutzern gezielt detaillierte Informationen zu verändern während sie den Überblick über den gesamten dargestellten Inhalt besitzen. Den Benutzern selbst bleibt durch das vorgestellte System die Kombination der Technologien jedoch verborgen.

A Wall-sized Focus and Context Display

TYPE OF THESIS: Diploma Thesis (6 months)

SUPERVISORS: Prof. Dr. Andreas Butz, Dipl.-Inf. Otmar Hilliges

DESCRIPTION: The project Fluidum investigates interaction in instrumented environments. These environments contain different kinds of displays and sensors. The instrumented room in the basement of Amalienstrasse 17, will, for example, contain three large back projection displays next to each other, covering the full width of a wall. In addition, a steerable projector can display objects directly on the wall above and below the displays. While the middle display will be interactive by itself (SmartTech SmartBoard), we need a form of interaction for the rest of the wall. The overall setup will then form a large focus-and-context display with exact interaction on the middle display and coarser interaction on the side displays and on the wall.

The idea how to build the coarse interaction is to place four fire wire cameras in the four corners of the wall, so that they look flatly over the wall along the diagonal. Within the camera images, we just look at the 2-5 rows of pixels which show the area immediately close to the wall. If a user touches the wall with a hand, each of these cameras will see a few pixels of different color in this row, suggesting in which direction the hand is seen. From the four patterns (actually already from two of them), the position of the hand on the wall can then be inferred more or less exactly. If we observe several rows, it might be possible to detect and distinguish a hand hovering over the wall.

The student is expected to mount the four cameras (the same as those used in the AR praktikum), connect them to a PC, and write software to read them out, analyze the necessary part of the image, and calculate one or several hypotheses for hand positions at interactive frame rates. A demo could, for example, display a halo on the display following the hand, or a simple form of click and drag across the wall. Alternatively, the same setup could be tried out on a desk surface instead of a wall first.

REQUIRED SKILLS: C/C++ or Java/JMF

Unterschrift eines Betreuers

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 29.03.2006 Sebastian Boring

Preface

ABOUT THIS WORK: This thesis has been written as *Diplomarbeit* (*diploma thesis*) at the *Ludwig-Maximilians-Universität München* (*LMU*), *Lehr- und Forschungseinheit Medieninformatik* during the past five months (November 2005 through March 2006). This work has been embedded in a research project named *FLUIDUM* and is part of its instrumented environment. *FLUIDUM* investigates interaction with different types of information within instrumented environments. For that purpose several displays embedded into a wall as well as a steerable projector mounted on the ceiling have been established. Different displaying techniques turn the entire wall into a display. The intention of this work is to solve the problem of interaction with such a large display for multiple users.

STRUCTURE OF THIS THESIS: This thesis addresses various aspects of my work and is thus of interest for varying readers. I would like to give a short overview of the different audiences that might read this piece of work including the chapters that could be of interest to them.

GENERAL READERS might not be familiar with ubiquitous computing and instrumented environments and should read chapters 1 and 2 to get a description and an overview of what has been done so far. Chapter 3 introduces the project *FLUIDUM* and explains the problem statement of this thesis followed by chapter 4 explaining the design decisions. Chapter 5 describes the theory of finger recognition including a mathematical background whereas chapter 6 is explaining the system's setup. Chapters 7 and 8 illustrate the implementations of both, the system and a demo application. The following two chapters give a summary of conclusions as well as future work.

FUTURE DEVELOPERS should read chapters 5, 6 and 7 to get a deeper understanding of how the system has been built. Finally, chapters 9 and 10 will give a summary of the performance of the system and describe the desired future work.

THE FLUIDUM TEAM should be familiar with chapters 1, 2 and 3. Chapter 4 is important to understand what hardware and software decisions have been made regarding this system. Furthermore, chapters 5, 6 and 7 are of interest to get an insight of how the system's theory and of how the system has been implemented. Chapter 8 illustrates how applications can make use of the tracking system whereas chapters 9 and 10 show conclusions and future work.

ACKNOWLEDGMENTS: A work of this size and complexity is not possible without the help of other people. At this point I would like to thank all people who assisted to get this done. I would like to thank the supervisors of this thesis, namely Andreas Butz and Otmar Hilliges who supported me with ideas and new opportunities at any time. Additionally, I would like to thank Lucia Terrenghi who always encouraged me to finally get this work done.

Furthermore, I would like to thank all of my friends for their support, encouragement and endurance during the past five months. Special thanks are going to Moritz Bartl, Christoph Donovan, Richard Seidl and Joachim Wildmoser for their extended proof-reading of this document. Additionally, I want to thank Daniela Knop together with my whole family, namely Jeanett and Rolf Voss, for their encouragement, support and inexhaustible patience during the past five months.

Table of Contents

Chapter 1

Introduction

With the new computing systems being faster, smaller and smarter, a new aspect has been formed, known as *ubiquitous computing* [55]. This new paradigm describes the ability of computers to be invisible to users while they support our everyday lives. One of many examples for this would be electronic devices integrated into a car that increase the convenience for the driver. These can be implemented as proximity sensors that open the door whenever a transducer gets within a certain distance of the car, for example. This paradigm also supports user activities in their original process. One example for this is a therapist using a paper-based technique which is then supported with a digital pen and paper to provide additional information. The therapist's practice itself is not changed due to the integration of a computer system [21].

Besides the paradigm of *ubiquitous computing* suggested by Weiser et. al., several other options for interfaces have been proposed, such as *ambient* [10] and *pervasive computing* [14]. All of them more or less describe the same function of computers: The computer should be a secondary device to the user, which is mainly invisible to him or her.

To simulate *ubiquitous computing* in specific situations many instrumented environments have been implemented in different research laboratories during the past years. One of the major questions asked is how information can be displayed at different levels, as well as how users are able to interact with it. For this, many more or less applicable approaches to the problem exist.

1.1 Instrumented Environments

Due to restrictions of computational power in association with the computer's size, the vision of *ubiquitous computing* cannot be realized today or in the near future. Thus, researchers need to implement specific environments where ubiquitous computing can be simulated in particular situations. One example of such an environment is a residential house augmented with sensors to test the paradigm in a real-life setting [15]. Other research groups use instrumented environments to simulate office or meeting situations [13]. All of them are, of course, prototypical settings which will not be used in real environments in the future.

To ensure ubiquity, sensors and other tracking devices need to be integrated in those rooms in a way that the user will not recognize them. Thus, people in those rooms should not be equipped with any additional computing devices such as sensors. Instead, they usually bring their own computers, such as a laptop, a cell phone or a *personal digital assistant* (*PDA*). Furthermore, multiple sensors need to be integrated in an instrumented room, to allow interaction with the room that seems to be a single device to the user.

One major question is how information can be displayed in such an environment, which needs to be highly interactive. Users should be able to modify digital information, which, for example, includes spatial arrangement across several displays using gestures as input [57]. This is a major challenge in current computer science research. For this reason, display walls (several displays on a large wall) have been implemented in instrumented environments [12][16][20][27][33][42]. Most of them have been built as rear-projection displays. Those are equipped with additional sensing technologies to allow the noted interaction. This shows the prototypical setting as the envisioned screen will not be implemented in this way in most cases. Instead, researchers are interested in the effect of a wall-sized interactive display. In the future those displays might be realized as electronic wallpaper.

Having multiple screens on a wall primarily relies on obsolete assumptions that have been made for desktop computers and screens. The most common one is a fixed screen size of a single desktop display where control elements of applications or even the operating system have a well-defined position and are easily accessible for a single user. This will not be true if every wall can act as an interactive display. One example of such a fixed position of a control is the Microsoft Windows “Start” button on the lower left of the screen. This is an unwanted situation if one is thinking about a wall with a large screen width. Thus, new display and interaction techniques need to be developed. One might think of a moveable menu realized by other display technologies such as steerable projectors, for example.

The interaction on a large display wall is another major field in computer science. There are several solutions using different tracking and recognition methods whereas most of them are planned and designed for a fixed maximum screen size and are thus not scalable, due to technical or financial issues. For this reason, new technologies will be examined and implemented to figure out the benefits they have for instrumented environments. Additionally, a wall-sized display needs to allow interactions with more than one user. The main problem is that users do not want to be equipped with hardware as they enter the room. Instead, the room should already be aware of the user’s position and the actions s/he wants to undertake. This is an open topic in computer science as it is very complex even for humans to know what a person wants to do. Additional complexity will be added having multiple people in one room since the computer then needs to know what tasks they want to perform and whether those are collaborative actions or not.

1.2 Focus plus Context Paradigm

Regarding the display technique, several approaches have been taken to implement new technologies, which assist users in accomplishing their tasks. One possibility would be the paradigm of zoom and pan [5]. This gives the user the ability to change the scene while panning or zooming. Panning may be realized with a fixed virtual camera, while moving the information, or with fixed information and a moving camera. Zooming allows the magnification or miniaturization of the information. In everyday life, several applications require this aspect. One for example is a user who wants to find the shortest path between two points on a city map. For accomplishing this task, s/he needs to zoom in to identify special points of interest (one way streets, for example) and to zoom out to find the spatial relation of the departure and destination point [3]. These are complex steps for the user as s/he needs to memorize the overview of the whole map when s/he is in detailed mode. Furthermore, all detailed information needs to be kept in mind when s/he is in overview mode. This example describes the principle of pan and zoom taken in the last years.

Another approach is the focus plus context paradigm, which suggests that the user – while working on a document’s part in detail – should still have the ability to have an overview of the complete document. The advantage the user gains with this paradigm is that s/he is able to view changes of the full information even if only a small portion (in the focus area) has been varied. This paradigm provides the user with the ability of concentrating on his or her task completely. There are several visual implementations for this paradigm while most of them are distortion-based ones. These allow the user to see the complete information with the focus region being enlarged and the context region being distorted. In the example given above, the outer parts of the focus region can be distorted using the *fisheye view*, for example [23]. This results in losing some information of the context but still gives the overview needed by the user.

The *Focus plus Context* technique does not only regard display technologies. Several implementations exist that combine multiple tracking systems resulting in one application. In the field of *Augmented Reality*, for example, different technologies have to be used since one system might not work inside (e.g. GPS[1]) and another one could only be operable indoors. This is mainly used in applications that need to track users in large areas. In the approach taken in this thesis, the system is a wall with a fixed size. In this case, the context region is not been intended for fine-grained interaction. Since I am not aware of visual tracking systems that are able to cover a wall, new input methods are needed besides existing solutions that have a smaller size. Thus the context region can use the same tracking technique with a lower resolution whereas the focus region has a high resolution.

In this thesis, I will describe a system designed to track people that want to interact with a large display wall. I will start with the related work that has been done in the research field (chapter 2). After this, I will introduce the thesis context (chapter 3) to describe the goals of this work and the design decisions (chapter 4) made to realize the proposed system. Chapters 5 and 6 will discuss details on finger recognition and the system's setup. Chapter 7 describes the implementation of the tracking system, whereas chapter 8 shows a first demo that has been implemented to demonstrate the system. Finally, I will discuss the conclusions (chapter 9) and give a perspective what will be done in future work regarding the system (chapter 10).

[1] GPS (Global Positioning System): Satellite-based positioning system developed by the Department of Defense (DoD) of the United States of America (Official start of operation: July 17th, 1995)

Chapter 2

Related Work

In this chapter I will give an overview of related work that has been done by others in the research fields. I will divide this into three major categories: “instrumented environments”, “focus plus context displays” and “touch sensitive displays”. I will compare each of them to the work I have done in this thesis, which also includes explaining the advantages and disadvantages of the introduced related work. I will also illustrate the reason why I have chosen certain parts of those solutions for this project.

2.1 Instrumented Environments

There are several implementations of instrumented environments as well as techniques for manipulating digital information that have been built in research. In this section, I will first describe those environments before I will show solutions which have been realized for such instrumented rooms.

The *Aware Home Research Initiative* (*AHRI*) [15] has been realized at the Georgia Institute of Technology and involves a complete three-storey, 5040 square feet house which functions as laboratory within a living environment. In this building, several researchers are able to implement and test their technologies. The main focus is on ubiquitous tracking and sensing to locate the inhabitants and recognize their activities [20], mainly using visual based tracking technologies [37][38]. Another focus lies on the design for elderly people and how technology has to be brought into their life to allow simple use of it. As this project covers a real living home, it is not applicable to the environment used in this thesis. The main reason for this is that the room in this case is only a single living room or office. However, this initiative generates a lot of ideas on what is possible and what might be possible in the near future in instrumented environments.

Another project is the *Interactive Workspaces Project* (*iWork*) of the Stanford University [45]. Its research focus and setup are much closer to the environment I have used in this thesis. It envisions a meeting and office room to support collaboration including three large displays (back-projected smartboards for interactivity) mounted on one wall. Additionally, a table including a non-interactive screen has been placed in the middle of the room. The most important implementations within this room are *iROS* (*interactive room operating system*) and *iStuff* (*interactive stuff*) [6][19] that both create a flexible software framework. This allows applications running in this setting to communicate with each other in a customizable way. Especially because of this dynamic architecture, *iROS* has been chosen for communication in this thesis. Besides this, *iWork* does not support interactivity throughout the whole wall and is thus not applicable to the proposed solution. Furthermore, the use of a table without any built-in interaction possibilities is not desirable by our project and will thus be extended as described later.

The project *Roomware* [33] also envisions an instrumented environment including a wall-sized display (*DynaWall*), several tables (*InteracTable* and *ConnecTable*) and a mobile armchair (*CommChair*). This scenario realizes the situation that is of interest in this thesis. The main focus is the *DynaWall*. It consists of three large displays that allow interaction with digital information. The interaction is limited to pen-based gestures and thus is not the desired final solution. Instead, the ideas given in this project will be used whereas the tracking will be generalized to a stage where additional hardware is not needed.

Furthermore, implementations for continuous workspaces [36] exist. This work presents a computer augmented environment in which users are able to interchange their data between various displays implemented in the room. These displays include stationary wall and table screens as well as portable computers brought into the room by users. Having all those tables allows people to use these as a virtual extension of the limited screen size given by personal computers and laptops. The interaction is realized by several installed digital cameras. The approach taken in this thesis still differs from this idea. Although I use large displays on the wall as well, they will have several technologies to track the user's input. Furthermore, the table will be interactive as it will have a DViT overlay to track two fingers simultaneously. Finally, the screens will not be an extension to existing display devices. They will build a single continuous display with personal devices still being a part of it.

After describing the visions for instrumented rooms, I will briefly introduce some technologies that have been used to support such environments. The main focus will be on steerable projectors and manipulation technologies. Rekimoto [35] has shown a new user interface allowing the physical copy of data among various computers and their attached screens. Internally, the data will be still transferred across a network infrastructure, but the conceptual model of physically picking data with a pen and then dropping it with the pen on another screen is simple to understand. The limitation of this system is the use of pens, which need to be available to every user. My approach is heading towards users without augmentation such as sensors and/or pens.

Another technique for interactive rooms is to transform surfaces into displays by using a steerable projector. This allows projection on any surface within the environment and is referred to as *multi-surface interactive display projector* (*MSIDP*) [32]. The tracking of user interactions, such as pointing and clicking, is realized with a camera mounted next to the projector. This work also describes a technique for the correction of distorted projected images as they occur whenever the optical axis is not orthogonal to the projection surface. A sample application of a steerable projector can be found in [8] where the projector acts as a pointing device to identify previously recognized items using visual markers. In this thesis, the approach is to take the advantages of the steerable projectors described in this related work without having the camera attached to the projector. This avoids interactions the camera cannot recognize as they are occluded by people standing between the interaction surface and the camera.

2.2 Focus plus Context Displays

The *Focus plus Context* technique "allows dynamic interactive positioning of the local detail without severely compromising spatial relationships" [23]. This means that a region of special interest will be shown in greater detail (*focus* region) whereas the global view is preserved at reduced detail (*context* region). The focus region does not leave out any parts of the context region and thus everything is visible. Implementations of such visualization techniques mostly use distorted views (*bifocal display*, *fisheye view*, *perspective wall*, etc.). There are several implementations for displays using this technique. One of them will be discussed in detail in the following.

Focus plus Context screens as described by Baudisch et. al. [3][4] try to combine display technologies with visualization techniques. The simple problem is that users often work with large documents that require them to manipulate a small portion while having an overview of the complete information. In today's computers, this can be realized with zooming or scrolling, which limits the user to see only a portion of the information at a time. Thus, the screen's size needs to be extended, resulting in expensive large high-resolution displays. Another approach is to use an off-the-shelf computer screen as focus screen and projecting the context region around the screen with a projector. The focus display has a higher resolution and thus allows users to work on a detailed portion of the document while having an overview of the complete information through the projection.

In this thesis' context, this paradigm will be modified as the display wall uses three projectors with the same resolution. Thus, the context region has the same visual quality as the focus region. The visualization technique will become a tracking technique, which means that tracking in the focus region will be much more accurate than in the context region. The projected image using a steerable projector does not have the same resolution as the embedded displays and serves as visual context display, which then might use both accurate and non-accurate tracking methods provided by the display wall.

Overall, the idea of focus plus context displays is the basic concept used in this thesis to implement the tracking system. Users will be able to do their fine-grained interaction on the focus display while having an overview of the complete information on the two context screens with the same resolution. Additionally, they are able to interact with digital items on the context screens as well as having a much lower resolution regarding the tracking.

2.3 Touch Sensitive Displays

This is the main part of related work regarding the tracking of this system. Within this field of research many technologies have been implemented and tested. The important point is whether the system uses infrared light (*IR*), pure visual or capacitive tracking. In the following I will discuss several implemented systems and explain whether they are applicable to this system or not.

The first design is the *HoloWall* [27] which uses infrared light as tracking method. The basic idea is a glass wall with a rear-projection sheet as display. Besides the projector, there are infrared light emitting diodes and a camera with an attached IR filter mounted behind the wall. The emitted light is reflected by the users hand as it approaches the wall. This becomes visible to the camera and can be evaluated by software using frame differencing as a simple image processing technique. The approach used in this thesis is not the same since I have a concrete wall surrounding the displays. Because of this, the technique would only be able to detect fingers touching the displays but not on the wall. For this reason, using IR-based tracking is not applicable with rear-mounted cameras.

Another approach is the use of *frustrated total internal reflection* (*FTIR*) [16] which also uses infrared light to detect a finger's position. The difference to *HoloWall* is the use of FTIR, the basic idea of fiber optics. When light travels across the border of two different media it usually becomes refracted to a certain extent. This depends on the angle of incidence and a specific critical angle corresponding to the entered medium. If another material is touching the interface, it frustrates the total internal reflection which causes the light to escape [16]. In this implementation, the light is being brought into an acrylic pane by attached infrared LEDs on each of its sides. The escaping light is then captured by a camera with an IR filter. This implementation has a high accuracy and the ability to track the touch of multiple fingers simultaneously. Unfortunately, this technique has some disadvantages that do not allow the use of it in the thesis. First, hovering is not possible as the only time light will escape the acrylic pane is when a finger actually touches the surface. Another point is that the LEDs would use a large amount of power as the emitted light has to have high signal strength to be used on a large wall. Besides these two disadvantages, the system also assumes having a camera mounted behind the display wall which is not possible as mentioned above.

The next solution in this field is an industrial product created by SMART Technologies [41]. This system is very close to the desired solution as it does not use back-mounted cameras. Instead, the cameras are built into a frame which is used as an overlay for a projection screen or television. It is equipped with four cameras to ensure a robust detection of positions using triangulation as tracking method [42]. Additionally, the frame contains infrared LED arrays which allow a stable detection. Whenever a finger or an object such as the provided pens touches the surface, the emitted light will be interrupted and the camera detects this missing light as stripe in the captured image. The usage of these arrays is highly desired but as shown above, using LEDs at this distance would increase the power consumption. Furthermore, the DViT only detects two objects at a time, which is not sufficient for a wall sized display. For this reason, only the position of the cameras and the idea of triangulation as tracking method may be obtained by the proposed system.

A further technique is to use capacitive sensing as it is implemented in today's touch screens and touch pads. The *Diamond Touch* [12] developed by the *Mitsubishi Electric Research Laboratories* (*MERL*) [31] is one of those systems. The idea is to use the user as an active part of the system where s/he acts as a coupling device between antennas and receivers. The antennas are implemented as an array in the interactive display whereas the receivers are attached to the chairs the users sit on. This allows a clear association of fingers placed on the screen to users as the system receives the signals from each user's chair simultaneously. The system is also robust to various objects being placed on the screen as these objects will not send any signals back to the computer. The main problem in using this system is that the user more or less needs to be connected to the computer. This fact is not desirable in this thesis as the users need to get into the room, move arbitrarily and interact with the system from wherever they want. Furthermore, the *Diamond Touch* only associates signals to chairs and not to the users [12]. Thus, using this technique is not applicable for this work.

Other capacitive sensing methods are extremely accurate as well, but highly expensive for a large display wall. Thus, the key factor is to use camera-based visual tracking as implemented in the DViT frame used for the smartboards. Also their tracking with the method of triangulation seems to be sufficient for the purpose of this system. In the following chapter, I will describe the problem as it exists after evaluating the related work in this field.

Chapter 3

Thesis Context

In this chapter I will illustrate the context of the presented work. First of all, I will introduce the project FLUIDUM at the LMU Munich with its core ideas, research goals and, of course, its members. After that, I will explain the initial problem statement and how it has been changed during the planning phase and implementation phase to the final one which has been realized in this work.

3.1 The FLUIDUM Project

The project FLUIDUM is a research project and is located at the LMU in Munich. It is part of the *Lehr- und Forschungseinheit* (*LFE*) Medieninformatik [28] and funded by the *Deutsche Forschungsgesellschaft* (*DFG*). The project has three permanent members (Prof. Dr. Andreas Butz, Dipl.-Inf. Otmar Hilliges and Dipl.-Design. Lucia Terrenghi) as well as several students working on different parts of the overall project.

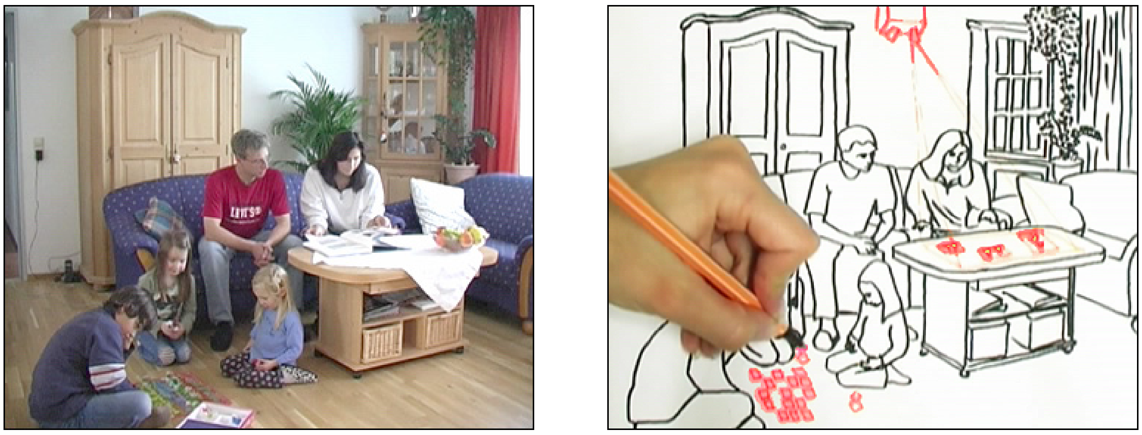


Figure 3.1: Left shows a common setting at home. Right illustrates the augmentation of this setting with continuous information spaces using display technologies.

The name FLUIDUM stands for *Flexible User Interfaces for Distributed Ubiquitous Machinery* and basically describes the main goal of this project. It examines the paradigm of ubiquitous computing [54] which is the disappearance of computers in the form of desktop PCs. Since those computers get smaller, faster and smarter, they can be embedded in our everyday life such as in furniture. The augmented items obtain computational capabilities and thus gain new functionality a person can use [13]. One example of such items would be a table acting as a standard table as well as an interactive surface by having a screen embedded in it. Another instance would be walls in a house which can act as display surfaces as well. With these ideas, the project's research focuses on *continuous information spaces* as exemplified in Figure 3.1. One of those information spaces has been built in the basement of the LFE Medieninformatik. In this room, several input and output components which will be described later in this thesis have been installed to create the illustrated setting.

Another research focus is the collaborative aspect of ubiquitous computing. In general this means, that large display surfaces will have multiple users working together to accomplish a specific task. Thus, all integrated components will be implemented to accept multi-modal input simultaneously allowing users to work on different digital snippets at the same time.

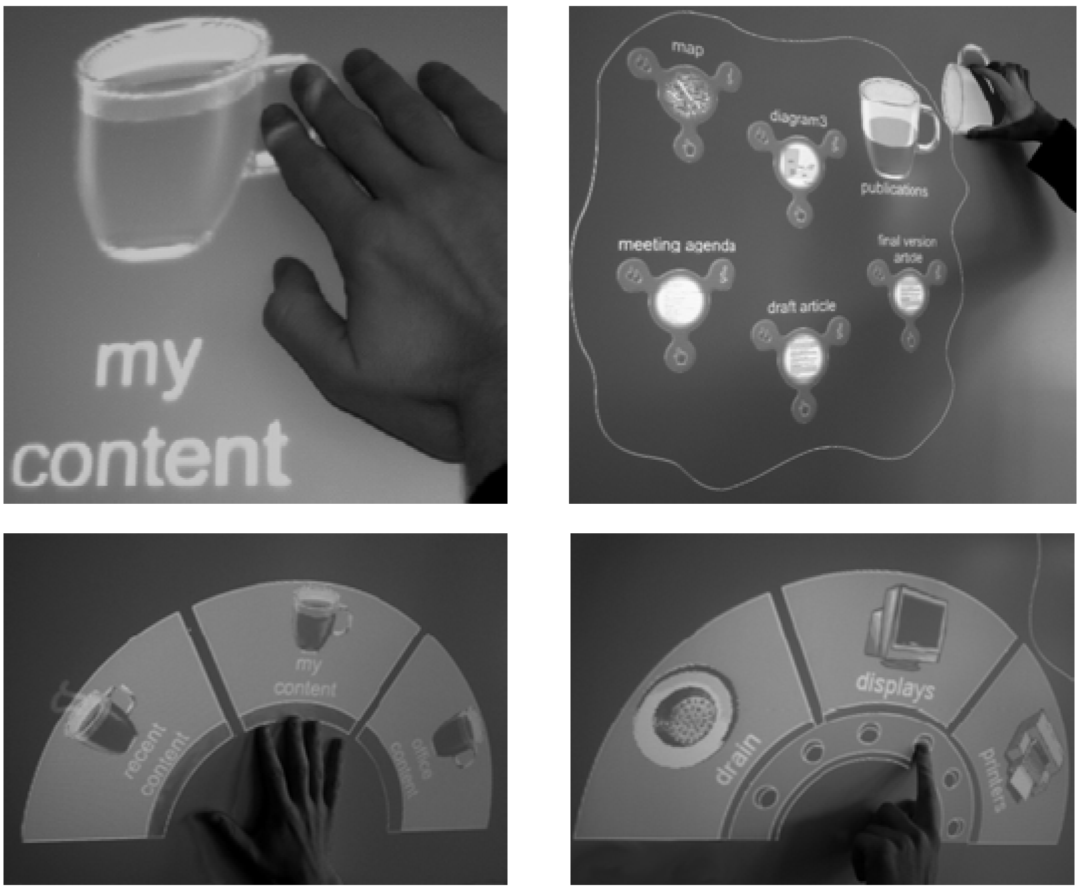


Figure 3.2: Top left shows a virtual mug filled with information. Top right shows the user turning the mug to pour out its content. Bottom left shows the user's dominant hand handling information. Bottom right shows the user's non-dominant hand managing resources [47]

A third aspect is the use of so-called *tangible user interfaces* (*TUI*). These allow users to interact with digital information using real-world objects with their specific and well-known behaviors. One example for such a well-known behavior is the Mug Metaphor [47] as shown in Figure 3.2. The mug has greatly acquainted attitudes such as its content being poured out. A tangible solution would be the handle of scissors as a tool to span a personal region on a screen. This can be done using the non-dominant hand whereas the dominant one will interact with the display in a more detailed manner.

Overall, the project investigates several interaction techniques with virtual information. As noted before, this includes having multiple users that might want to use both of their hands to interact with the displayed content. As I have now described the main goals of the project, I will introduce the problem statement related to this work and how it will be included into the research of FLUIDUM in the next section. This will be a brief overview of the goals I want to achieve during the thesis.

3.2 Problem Statement

As mentioned above, the project FLUIDUM has built an instrumented room in the basement of the LFE Medieninformatik. Several input and output technologies have been placed in the room to get a *continuous information space*. As information can be displayed throughout the room on all its walls, the project needs to enable interaction with the contents being shown anywhere. In the first phase only one wall will be instrumented to allow interaction. This wall additionally contains three large displays (one smartboard and two projection screens) as well as the possibility to have a projected image on any place of its surface through a steerable projector mounted on the ceiling in front of the wall.

The mentioned interaction needs to be available without having to equip the user with special sensors or visual markers. There are several industrial solutions for this problem without having the user augmented with sensors, but they are extremely expensive if used on a complete wall. For this reason, I am trying to find a much more inexpensive solution to solve the problem of having interaction on a complete wall. One approach to this is to augment the wall with cameras to capture interaction of users. To ensure that the system will track different users with the same performance, the cameras will only observe a small stripe shortly before the wall's surface. Whenever fingers touch the wall, the cameras will recognize them as shown in Figure 3.3. Once these positions have been recognized, the system should be able to determine the finger's position on the wall using mathematical calculations.

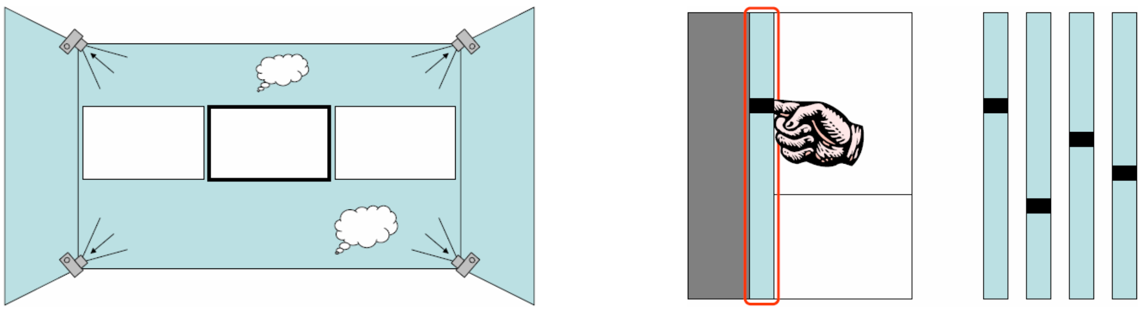


Figure 3.3: Left shows the basic idea of the wall's setup. Right shows the disturbance in the captured images resulting from a finger touching the wall (Drawings by Prof. Dr. Andreas Butz)

An additional feature is to track multiple fingers. This can be done by capturing multiple disturbances in the image of each camera. After the recognition, the software needs to calculate the positions on the wall. Having multiple fingers on the wall also needs a motion prediction to ensure a correct association between detected positions and previously recognized fingers. This component must then be able to feed other running applications with the newly created data on the current interaction happening on the wall. It can be implemented independently of the main tracking system if it permanently gets the position data from it. The component also should accept multiple inputs from various devices such as the main tracking system based on cameras and the interactive smartboard in the center. This ensures that an arbitrary number of applications can use the data in a unified form.

Furthermore, the tracking system needs to be inexpensive, which especially affects the used cameras. A suggestion would be to use webcams instead of high-performance video cameras with a frame grabber. This is an additional challenge for the software as it has to deal with low resolution images.

On the output side of the system, several components need to be connected. These include the display wall which has multiple screens as well as the steerable projector. Both need to be able to use the calculated position data within their applications. As the display wall is fixed, this can be easily done by giving screen coordinates to it. The steerable projector instead needs to get real-world three-dimensional data to know where it has to project the requested information. This needs to be provided by the component merging the input data from multiple devices.

The main research question for this thesis is to figure out if relatively inexpensive hardware is able to compete with industrial solutions as well as technologies described in section 2.3. This includes the accuracy of detected positions (measurable with the Euclidian distance of real and detected position), the system's execution speed (measurable with the time difference of touching the wall and actually executing the desired action) and the system's robustness (measurable with the number of detection errors). This also includes having several people use the system to evaluate if the system is able to perform correctly independent of the users.

Another question is to find out if the measured performance of the system is applicable for users. This might be answered by developing a demo application which takes all described components into account. Then, several tasks need to be defined to figure out if the users are able to use such a system. After evaluation, it is possible to see whether the system is performing well enough for users or not.

To answer most of these research questions, I will implement a fully functional prototype as well as a demo application regarding to the problem stated in this section. In the following I will describe in detail how these steps have been performed.

Chapter 4

Design Decisions

In this chapter I will describe which decisions have been made regarding the hardware and software design of the system. I will start with the resulting requirements to build the system as envisioned in section 3.2. Further, I will explain the decisions made regarding the hardware components. Finally, I will describe how the software decisions have been made regarding programming languages and third-party libraries.

4.1 System Requirements

In this section I will describe the basic requirements to implement the desired system. Since this application falls into the category of tracking applications, it needs to have a low latency as well as high accuracy. This includes decisions about on which cameras and computers to use in the system. Also, the output is required to be fast in order to give a quick feedback to the user. To get a deeper understanding of the general requirements, I will describe them separately, divided into *functional*, *non-functional* and *pseudo* requirements [7].

Functional requirements describe specific behaviors of the system during runtime. They are supported by *non-functional* requirements which usually describe adjectives of the system such as performance, accuracy or reliability. Table 4.1 summarizes the *functional* requirements that have been defined to implement the system.

Requirement	Description
Visual-based tracking	The system needs to capture images from streams produced by attached cameras which then will be evaluated.
Image processing	Useful methods need to be implemented to process images and detect objects within them. This may be done using third party libraries.
Calibration procedures	The system must provide calibration functions. With them, the user can re-calibrate the system, should it become inaccurate. The calibration should also allow adjustments of the used filters.
Multi-modal input	The system must support multiple users while keeping it possible for them to interact with both of their hands simultaneously. Thus, a unique association between recognized positions and fingers must be provided.

Unified input device	Input produced by several devices needs to be unified to enable application programmers to use the different input devices as one virtual device.
Adapt Event Heap infrastructure	The tracking system needs to use the Event Heap which already exists in the current setting. Thus, several events need to be defined to identify the desired actions a component wants to execute.
Use of different displays	The system needs to use several output technologies to display information onto the complete wall. This includes having large displays as well as a steerable projector.

Table 4.1: Functional requirements of the system

I will continue with the *non-functional* requirements to introduce the user-related aspects of the system. Table 4.2 gives a detailed summary of the main (non-functional and pseudo) requirements that will be used to implement the system.

Requirement	Description
Availability	All components that are producing input events need to be available at all time. Without them, the system is unable to work properly.
Reliability	All components that are producing input events need to work properly without creating wrong events and/or associations between positions and fingers. Additionally, the output devices must not display any incorrect content.
Robustness	Wrong inputs such as tapping the finger multiple times on the same place must not affect the system because users want to learn how the system works and get frustrated if errors occur too often.
Fault tolerance	The system must be able to handle failures occurring during image capturing as well as position detection. Additionally, the system should be able to handle connection errors.
Responsiveness	The system must provide fast feedback to users to let them know whether an operation succeeded or failed. This must be provided by the system at any time and in any circumstances.

Table 4.2: Non-functional and pseudo requirements of the system

Besides these requirements, there will be no further limitations regarding the programming language used to implement the complete system. In the following sections I will give an overview of the hardware and software decisions that have been made according to the requirements gathered above.

4.2 Hardware Decisions

For the used hardware, I had to make several decisions which took a large amount of time while planning and implementing the system. The first decision had to be made regarding the large display wall. The visual output will be given with three projectors mounted behind the wall for rear projection. The wall itself contains three displays. Due to financial constraints, only the center screen is equipped with its own tracking provided by a smartboard from SMART Technologies [41]. For the side displays simple projection foil with Perspex has been used. The three projectors display one desktop screen which means that they all need to be attached to a single computer. Thus, this computer needs to have a graphics card with three output connectors and auxiliary software to display the content. Additionally, the smartboard will be attached to this computer as part of the large screen. This results in a high load of capacity for this machine which makes it impossible to accomplish other high-performance tasks. For this reason, the visual tracking system needs to be placed on another machine. Since the task of capturing images is resource-consuming as well, it must be the only task running on this computer to avoid leaks in the capturing process. It also needs to have at least four USB 2.0 connectors to ensure that all cameras can be attached. Since both machines are dedicated to their tasks, a third machine is being needed to serve the steerable projector (see Figure 4.1) mounted on the ceiling.

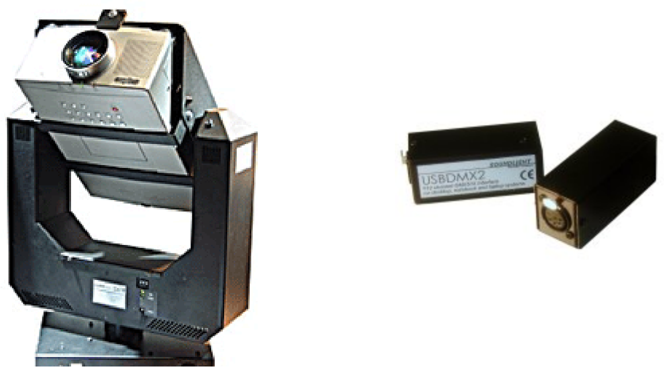


Figure 4.1: Left shows an example of the steerable projector unit [11] used in the system. Right shows the USB to DMX interface which is used to connect the projector to a computer [42]

This machine also needs a USB port to connect the projector, as well as a graphics card with two connectors to have a standard display attached besides the projector. The projector has already been installed prior the implementation has been started. It is a Sanyo PLC-XP45 [39] attached to the steerable unit beamMover 40 from publitec [34]. Table 4.3 summarizes the different computers that will be used for the system.

Designation	Resources
Display component, smartboard	Pentium 4 with 2.4 GHz and Hyper Threading 1 GB of main memory Matrox Parhelia 256MB, AGP 8x, Triple Monitor Out USB 2.0 port for the smartboard
Tracking system	Pentium 4 with 2.4 GHz and Hyper Threading 1 GB of main memory Four USB 2.0 ports for the cameras
Steerable projector	Pentium 4 with 2.4 GHz and Hyper Threading 1 GB of main memory NVIDIA GeForce 6200 TurboCache^{TM} USB 2.0 port for the DMX interface (projector)

Table 4.3: Summary of the computers and their designations being used in the system

The second decision was the choice of appropriate cameras. Since the system should be realized at low cost, I settled for off-the-shelf webcams with high resolution, fast connection interfaces, high frame rates and a wide-angle lens. Thus, I intensively tested four products to figure out which camera matches the system's needs.

First, I tested the Logitech QuickCam Pro 4000 [26] since it was already available at the LFE Medieninformatik in Munich. This product comes with a USB 1.1 connection, a resolution of 320 by 240 pixels at 30 FPS and a horizontal field of view of about 45 degrees. Compared to the cameras tested subsequently, it did not match the desired requirements at all.

The next camera was the Unibrain Fire-i digital camera [51]. The LFE Medieninformatik also had some of those cameras which made tests easily possible. This product uses FireWire as connection interface providing up to 400 MBit per second. Additionally, it has a video resolution of 640 by 480 pixels with 15 FPS. The only disadvantage is the small horizontal field of view (42 degrees), which means that the camera is not useful for this system.

The third camera I have tested is the Creative WebCam Live! Ultra [9]. With a field of view of 76 degrees diagonally, 640 by 480 pixels of resolution at 30 FPS and USB 2.0 as connection interface (up to 480 MBit/s), this camera would have been perfect for the system. Unfortunately, it was not possible to connect more than one camera to a single computer. If a second camera was attached, the captured images of both devices are merged into one image stream.

Thus, I had to test and evaluate another camera, which was the Logitech QuickCam Fusion [25]. It provides a resolution of 640 by 480 pixels at 30 FPS, a USB 2.0 connection interface and a field of view of 72 degrees diagonally. Additionally, it has a very low image distortion which is negligible. This camera satisfies the requirements and is used in the system. Table 4.4 finally summarizes the four cameras with their parameters.

Model name	**Logitech Quick-Cam 4000 Pro**	**Unibrain Fire-i digital camera**	**Creative Web-Cam Live! Ultra**	**Logitech QuickCam Fusion**
Product view				
Resolution	320 x 240	640 x 480	640 x 480	640 x 480
Frame rate	30 FPS	30 FPS	30 FPS	30 FPS
Field of view	45°	42°	76°	72°
Connection	USB 1.1	FireWire	USB 2.0	USB 2.0
Problems	Connection interface too slow (15 MBit/s)	Low field of view	Only one camera per computer possible	–

Table 4.4: Tested cameras with their specifications of interest to the system (Image sources from left to right: [26][51][9][25])

With these decisions, I finally had all hardware components to build the system. In the following section I will describe the decisions made regarding the software including programming languages, third-party libraries and device drivers.

4.3 Software Decisions

In this section I will describe the software decisions I made to implement the system on the chosen hardware. First, I needed to figure out which programming language fits best for the main tracking system. This decision included the possibility of rapid prototyping, simultaneous support of four capture devices and fast processing of images. I intensively examined Java 1.5 [46] and C# with the .NET Framework [29]. Because of a strong teaching in Java by LMU, I wanted to use it for this system, but due to several problems I finally decided to use C#. Before I will explain this decision, Table 4.5 will summarize the comparison of both languages.

	Java	**C#**
Platform independence	Available	Restricted to Windows
Execution speed	Fast	Fast
Rapid prototyping	Given	Given
Support of capture devices	None in standard version One at maximum with JMF[2]	None in standard version Arbitrary number (DirectShow)
Image processing	Good in standard version Fast with JAI[3]	Fairly good in standard version Fast with external libraries

Table 4.5: Summary of the comparison of programming languages

As shown in this table, the use of Java would have been possible if it supported more than one camera at the same time. Since this is not given, I decided to use C#. Thus, I had to find libraries that can handle multiple devices. The possibly best solution is the DirectX SDK [30] from Microsoft which has the DirectShow package in its extra components. With this library, I was able to use the FilterGraph concept provided by DirectShow. This graph allows a hierarchical structure of the attached devices as well as filters (see Figure 4.2). Although it comes as a C/C++ library, it is easily embeddable into C# using a COM wrapper [49].

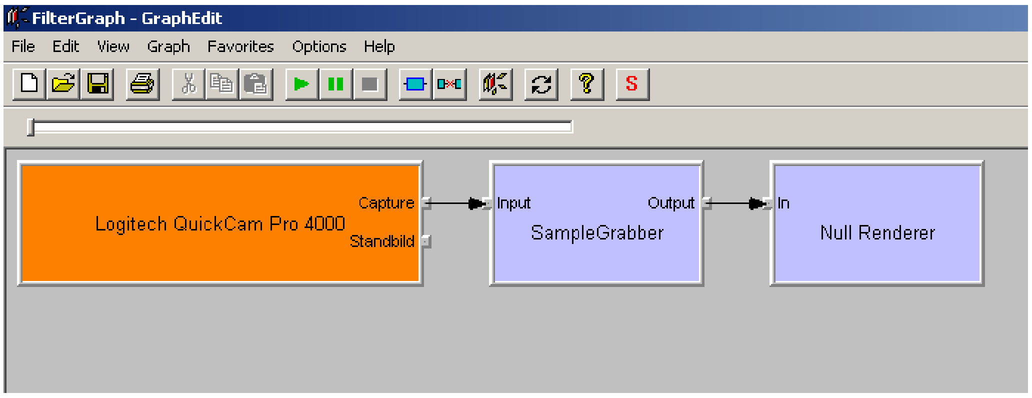


Figure 4.2: Shows the *GraphEdit* application included in the DirectShow package used to build and test a created filter graph.

The next step was the decision about the imaging library. There are basically two libraries I had to choose from. One is OpenCV from Intel [17] written in C/C++ and the other one is

[2] JMF (Java Media Framework): Framework for capture and access applications of digital media such as audio and video streams. Latest version is 2.1.1e

[3] JAI (Java Advanced Imaging): Java API for digital image processing. Latest stable version is 1.1.2

AForge from Andrew Kirillov [48], written in C#. OpenCV provides a lot more functions than AForge, but those are not of interest for this system. Since the desired operations are provided by both libraries, I decided to use the second one because it is easier to integrate in the system's main software component, as it is written in the same programming language.

As described in section 4.1 the system needs to use the Event Heap infrastructure written in Java. Thus, all distributed components should be easily attachable. This is the main reason for the third decision I made. Besides the main tracking system, all other components will be implemented in Java to ensure high compatibility without using wrappers and/or native interfaces. As shown in Table 4.5, Java is as fast as C# and as such appropriate for tasks that do not need to use multiple capture devices.

The final decisions made were on the device drivers of all attached components. The first is the smartboard in the wall's center region. The University of Queensland (Brisbane, Australia) provides an SDK [50] to receive events produced by the smartboard. It is written in C/C++ and comes with a native interface for Java and C#. Thus, it is easy to integrate the smartboard into applications written in Java. The second device attached is the steerable projector. It only comes with a C/C++ dynamic-link library (DLL) to control the projector's DMX interface. For this reason, I need to use C/C++ to implement a library that can be called from a Java application using JNI[4]. The third decision was made on the device drivers of the capture devices. Since DirectShow provides most of the functionality needed to operate and configure the devices, I decided against using the Logitech QuickCam SDK [24] to avoid a programming overhead. After that, all decisions had been made regarding the software components. Table 4.6 will finally summarize the different languages to be used for the complete implementation of the system.

Component	Programming Languages
Tracking system	C# (basic tracking) Java (v1.4.2 / connection to the Event Heap)
Display wall	Java (v1.4.2 / display component and smartboard SDK)
Steerable projector	Java (v1.4.2 / display component) JNI (DMX to USB interface)

Table 4.6: Summary of the used programming languages to implement the system

As I have described all design decisions regarding both, hardware and software, I will continue with the theoretical thoughts of finger recognition in the next chapter. Subsequent, I will explain the system's setup and its implementation using the decisions illustrated in this chapter.

[4] JNI (Java Native Interface): Provides connectivity from Java to native C/C++ libraries

Chapter 5

Finger Recognition

This chapter will give an overview of how the system is able to detect objects and fingers respectively. First, I will give a short overview of tracking methods currently available. After that, I will describe how fingers are recognized in captured images. In section 5.3 the principle of triangulation will be introduced to explain the basic concepts used for calculating positions. In section 5.4, I will show how positions match up with detected fingers to ensure continuity of movement. Finally, an overview of the calibration procedures will be given to demonstrate how the system and its cameras can be calibrated and how they can make use of this data during runtime.

5.1 The Tracking Method

Today, there are lots of different tracking methods which can be categorized in two groups. Some of them are able to track the position of an object, while others additionally can recognize its position. They can also be classified with the parameters they observe during tracking. Before I describe the most common techniques in detail, Table 5.1 will summarize the discussed methods and rank them.

Position method	Observable	Measured by
Proximity sensing	Cell-ID, coordinates	Coverage range of sensor
Lateration	Range	Traveling time of emitted signal Attenuation
	Range difference	Traveling time difference of emitted signal Difference of attenuation
Angulation	Angle	Receiver arrays
Dead reckoning	Position	Any other positioning method
	Direction of motion	Gyroscope
	Velocity	Accelerometer
	Distance	Odometer
Pattern matching	Visual images (markers)	Camera
	Fingerprint	Received signal strength

Table 5.1: Summary of positioning and tracking methods [22]

Every tracking method has advantages as well as disadvantages regarding the costs to build and/or operate them, their accuracy and their detection speed. Thus, it is necessary to define the requirements of a tracking system and its results, respectively. In mobile communication systems such as GSM it is not important to know the exact position of the user (i.e. a cell phone) while airplanes with GPS navigation systems need a high accuracy.

The requirements of the built system are high accuracy as well as nearly real-time processing of information. Another important attribute is that the user will not be equipped with any sensors or additional markers. Thus, the only assumption that can be made is that s/he acts as a light emitter. With this in mind, it is obvious to use either the direction or the distance of the emitter which results in a decision between lateration and angulation. I will describe both methods in detail and match them with the technical limitations as well as the desired properties of the system.

5.1.1 Lateration

This method is a common technique in positioning objects. The most popular tracking and positioning system that uses lateration is GPS. The basic idea is to measure the distance or distance difference between the object and at least three receivers. It can be subdivided in two different strategies – circular lateration (distance measurements) and hyperbolic lateration (distance difference measurements) – to gather the position of an object.

Given the distance between the object and three receivers as used in circular lateration, the process of calculating the position mathematically in two-dimensional space is as follows: First, create a circle around the receiver with its distance from the emitter. Second, intersect those circles and get the object's position. Figure 5.1 shows the basic procedure of circular lateration.

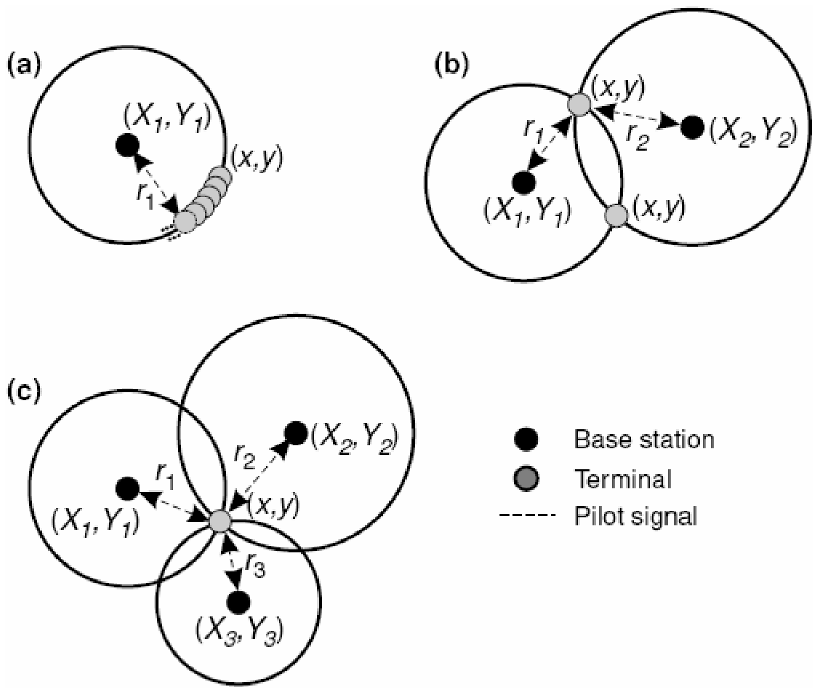


Figure 5.1: Circular lateration (2D): (a) shows only one receiver and thus a circle of the possible position. (b) illustrates two receivers and therefore two possible positions. (c) uses the third receiver to get the exact position of the emitter. [22]

Hyperbolic lateration does not use the absolute distance between the object and a receiver but the distance difference of a pair of receivers. These distances result in a hyperbola of a point's position. Using a third receiver, all the hyperbolas (of every pair of two receivers) intersect in one point and thus give the object's position. Figure 5.2 demonstrates the basic procedure of hyperbolic lateration.

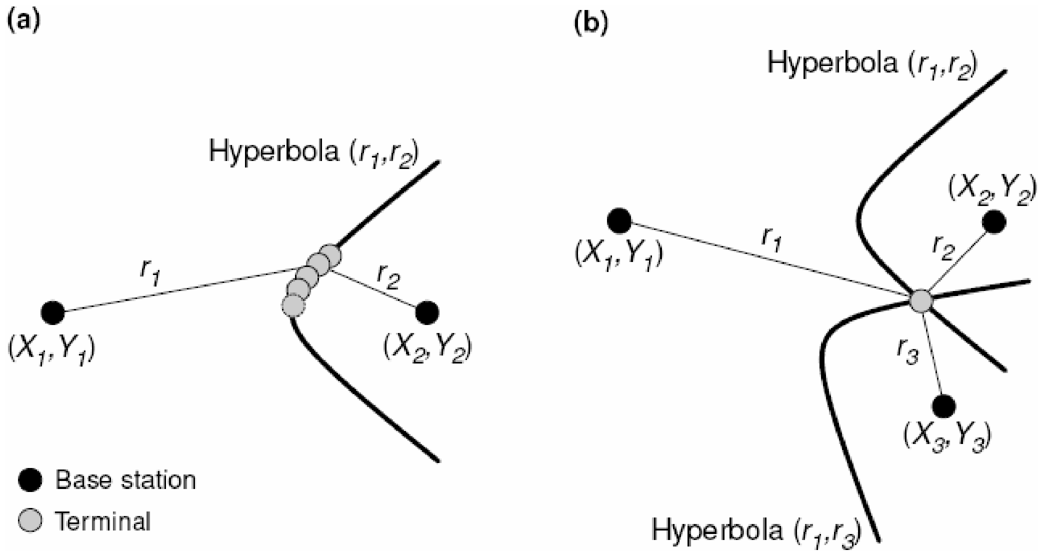


Figure 5.2: Hyperbolic lateration (2D): (a) shows the hyperbola for two receivers. (b) demonstrates a second hyperbola to calculate the emitter's exact position. [22]

Of course, both methods suffer from measurement errors caused by physical impacts such as multipath propagation, velocity of propagation in different media or signal strength. These errors can be corrected using Taylor series as described in [22]. In general, lateration is also known as ToA (Time of Arrival).

5.1.2 Angulation

This method is another technique to gather the position of an object. In difference to lateration it uses the incoming angle of the received signal and not the object's distance. With the known coordinates of the receivers, the system only needs two of them to recognize an object's position (see Figure 5.3).

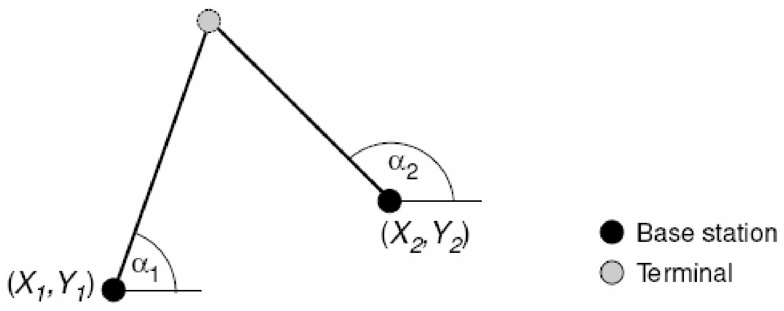


Figure 5.3: The basic principle of angulation (2D) with two receivers. The positions X_i, Y_i are known before the measurement of the angles [22]

It is not widely used in mobile communication systems, since either the base stations or the mobile terminals have to be equipped with sectorized antenna arrays. It also suffers from detection errors which can be solved using Taylor series [22]. In general, angulation is also known as AoA (Angle of Arrival) or DoA (Direction of Arrival).

Given these two methods, lateration seems to be the easier choice. Looking at the requirements and limitations it is not possible to use it in this system because objects (e.g. fingers) are not able to determine the time when they emit light. Also it is not possible to emit light only at discrete points of time since it is a continuous physical emission. Thus, I decided to use angulation as tracking technique. The described antenna arrays are implemented within the used webcams as I can use every pixel in the captured image as a single antenna. In the following two sections I will describe the calculation of angles with captured images (section 5.2) and the subsequent calculation of positions with those parameters (section 5.3).

5.2 Calculating Angles from Live-Captured Images

To calculate the final positions of fingers on the wall it is necessary to gather the angles from each finger to the receivers. To accomplish this, the system is equipped with four cameras, which are able to capture live images. These images need to be analyzed by the system to get the relevant information (e.g. the fingers) within them. I examined several ways to do this step resulting in a final solution for this prototype.

It is important that the system will not detect non-existent objects (*false positives*) as well as letting existent objects be undetected (*false negatives*). As described in section 3.2, the four cameras of the system capture a small stripe over the wall's surface. With this knowledge it is possible to create the assumption that the image's background (white wall) as well as the fingers' color (other than white) will not change over time. Thus, I used simple frame differencing in the first iteration. Image differencing is the difference of each pixel $p(x, y)$ of two images I_1 and I_2. Mathematically it can be described as follows:

$$\forall p_1(x,y) \in I_1, p_2(x,y) \in I_2 : p_{21}(x,y) = \left| RGB(p_2(x,y)) - RGB(p_1(x,y)) \right| \in I_2 - I_1;$$

This solution worked fine until I switched on the displays (e.g. the projectors) on the wall. The background and the finger turned into the display's color which made differencing impossible. Figure 5.4 shows the results with and without having the displays turned on.

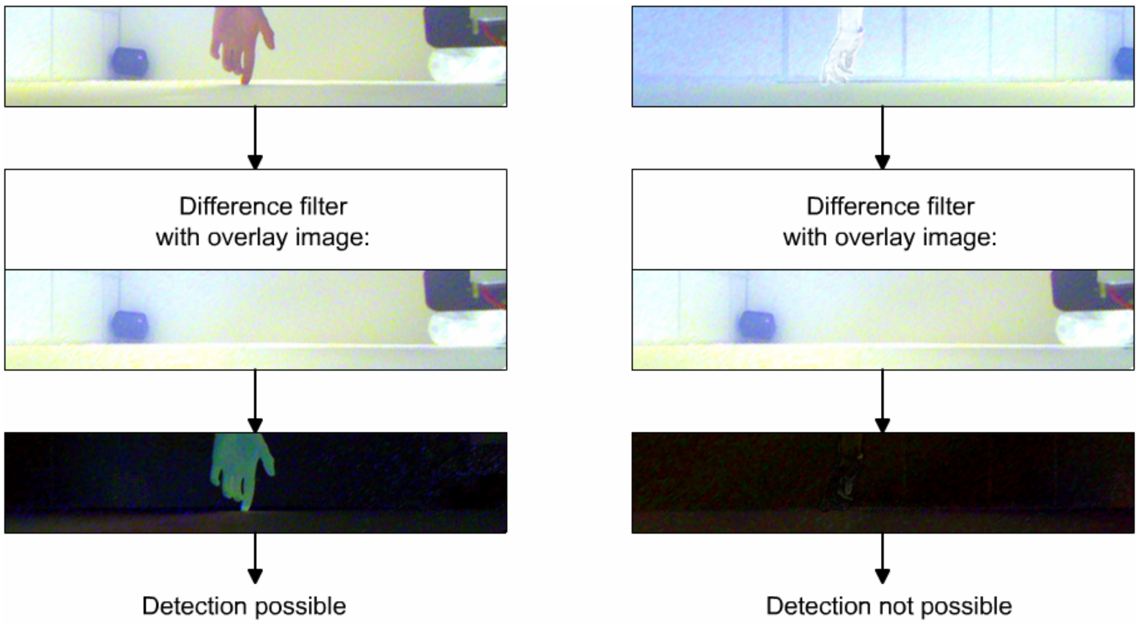


Figure 5.4: Left shows the detection using differencing without having the displays turned on. Right shows the same method while the displays are producing color radiation.

Another solution exists with the usage of HSL filtering[5]. The basic idea is to replace pixels with a single predefined color (e.g. black) that are not of interest to the system (see Figure 5.5). This filtering method takes all three HSL components into consideration which means that it

[5] HSL (Hue, Saturation, Luminance): Color model representing each pixel in these three components

only allows pixels that are within the predefined hue (0 through 360), saturation (0.0 through 1.0) and luminance (0.0 through 1.0) range. This assumes that the fingers' color will not change during runtime and will be in a given color spectrum respectively. As described above, the color changes due to the displays' radiation. Thus the system does not know which color spectrum it needs to use for detection. Figure 5.5 summarizes the results with and without having the displays turned on.

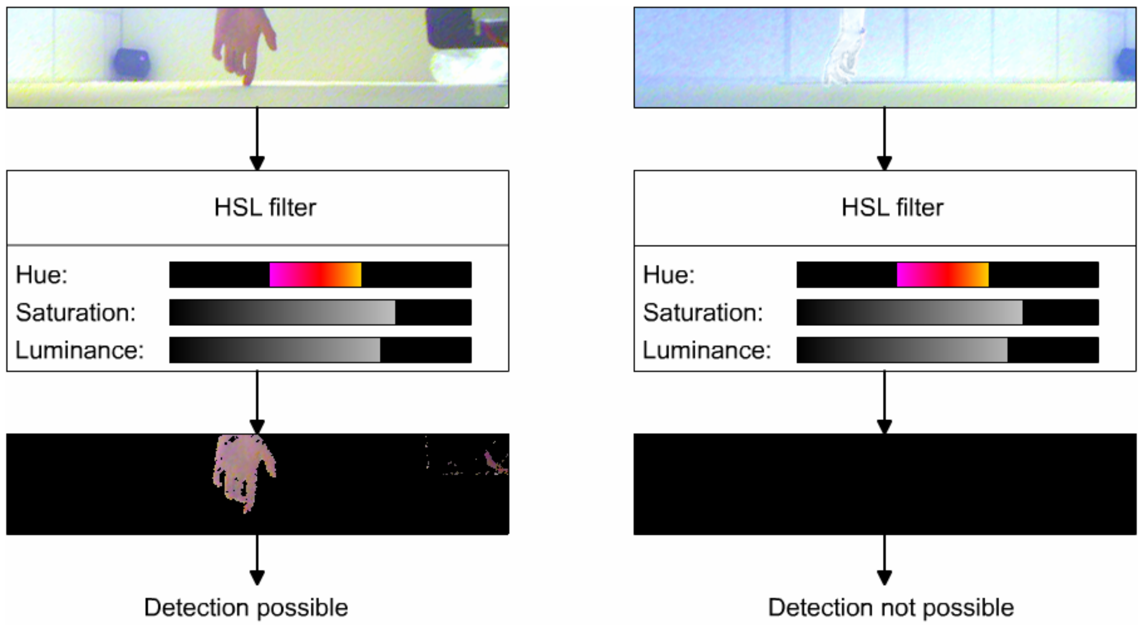


Figure 5.5: Left shows the detection using HSL filtering without having the displays turned on. Right shows the same method while the displays are producing color radiation.

In the third solution I used differencing again, because this is a fairly fast procedure. Thus, I needed to avoid the errors the system made in the first iteration. The basic problem was the changing background color of the image to the color of radiation produced by the displays. To get a unified background I mounted black velvet onto the part of the wall which is covered by the cameras (see section 6.1). Black velvet has the characteristic of a very low reflection of light (lower than 5% [53]) and will therefore appear as a black background independent of the displays' color of radiation at any time. With this technique the fingers' color can change as long as it is not black. This will not even happen if the projected image is black because the addition of a finger's color and black is still the finger's color. Figure 5.6 shows the results I achieved with this method.

After evaluating these methods, it is clear that the third solution to recognize fingers in the captured image is the most suitable. Additionally this procedure has further advantages: First, it does not detect shadows caused by fingers on the wall because a thrown shadow on a black surface does not affect the surface's color. Also, it avoids noise in the background because a black surface will not cause background noise (assuming the camera has a high contrast value).

Finally, I am able to skip the process of differencing because a subtraction of the color black is not necessary (it has the color value of 0). An additional threshold filter provides a binary image with only two colors, black (background) and white (finger). It uses a predefined level of luminance to decide whether the beheld pixel is part of the background (low luminance) or part of the finger (high luminance). This binary image can be searched for blobs, which are a minimum bounding box (rectangle) for a detected finger.

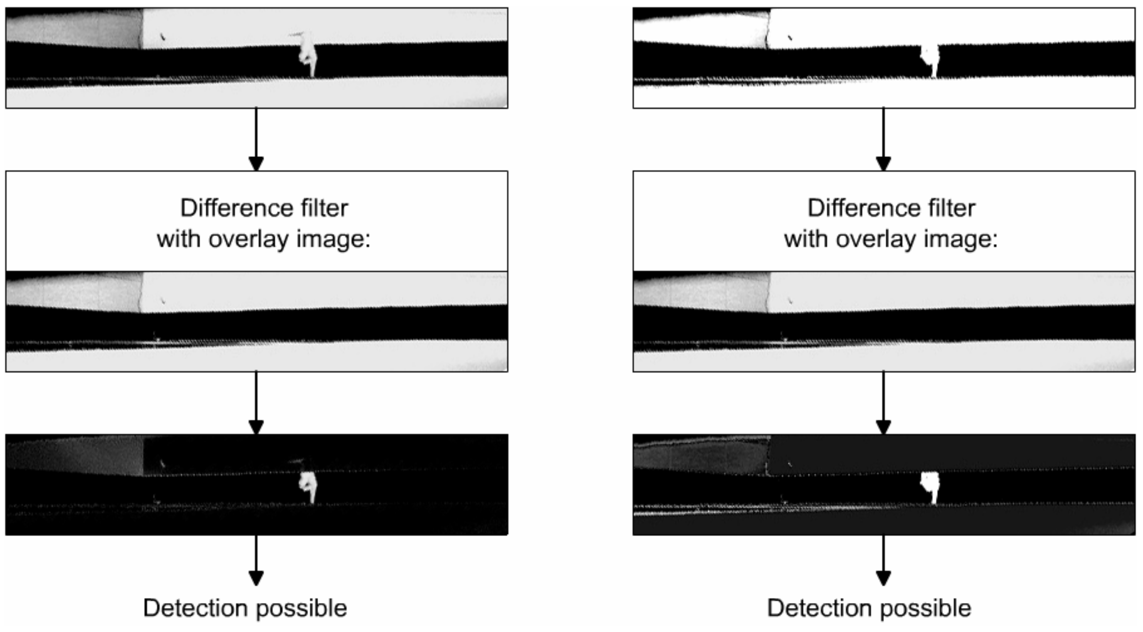

Difference filter with overlay image:
Difference filter with overlay image:
Detection possible
Detection possible

Figure 5.6: Left shows the detection with black velvet mounted on the wall with no display radiation. Right shows the same setting with having the displays turned on resulting in radiation.

Once we have the blob's position within the image, it is possible to compute the "Angle of Arrival" of this received signal. For this, two parameters are needed, such as the camera's angle of aperture α and the width w of the image. The calculated angle β will be the angle between the camera's normal and a virtual line from the camera's origin to the detected point. Figure 5.7 illustrates the mathematical background of calculating the angle.

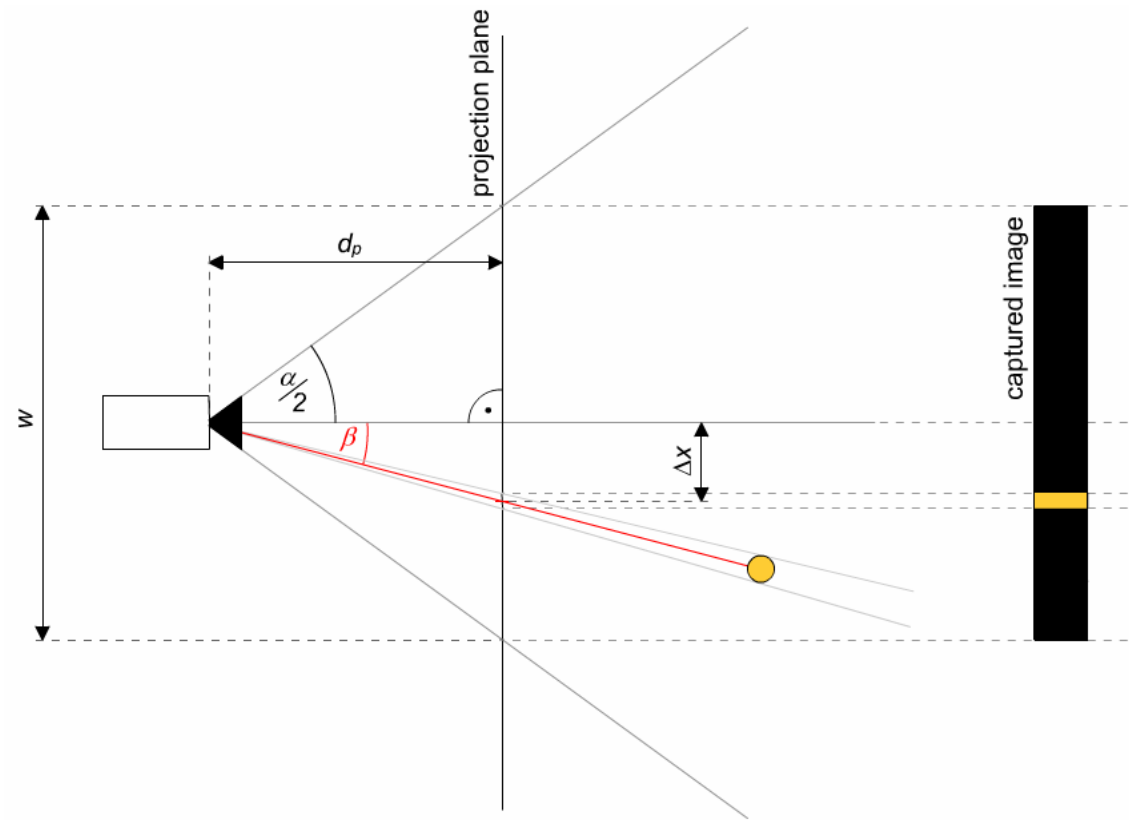


Figure 5.7: Geometrical illustration of the calculation of images captured by a camera.

Given these parameters, the calculation of β can be done with the computed Δx, which is the blob's distance from the center of the image. The parameter d_p (virtual distance from the projection plane to the center of projection) can be calculated with the angle of aperture α:

$$\tan{}^{\alpha}\!/_{2} = \frac{w}{2 \cdot d_p}; \Rightarrow d_p = \frac{w}{2 \cdot \tan{}^{\alpha}\!/_{2}};$$

$$\tan\beta = \frac{\Delta x}{d_p} = \frac{2 \cdot \Delta x \cdot \tan{}^{\alpha}\!/_{2}}{w}; \Rightarrow \beta = \arctan\left(\frac{2 \cdot \Delta x \cdot \tan{}^{\alpha}\!/_{2}}{w}\right), \ \Delta x \in \left[-\frac{w}{2}; \frac{w}{2}\right];$$

The parameter Δx can be calculated with the processed images shown in Figure 5.7. Since every horizontal pixel value within the image will result in a different angle, one can say that every vertical line of pixels is one element of the array of receivers used for angulation. Furthermore, the width w of the captured image gives the tracking resolution and is therefore a significant identification of the system's quality.

5.3 Process of Triangulation

As mentioned before, the principle of triangulation assumes two fixed receivers that detect the arriving angle of signals by the emitter. In this case, the emitter is a finger, which reflects incoming light rays while the receivers are the cameras. The used tracking algorithm is called "Angle of Arrival" (AoA, see Figure 5.3) and is used – for some parts – in mobile communication having sectorized antennas.

The idea of triangulation is to create two virtual lines g and h from each of the receivers A and B with the known angles of arrival α and β. These lines can be intersected, and the intersection gives the emitter's position P. If more than two receivers are used, the system gets more robust by reducing inaccuracies of the calculated angles. Figure 5.8 will give an impression of how triangulation works with four receivers.

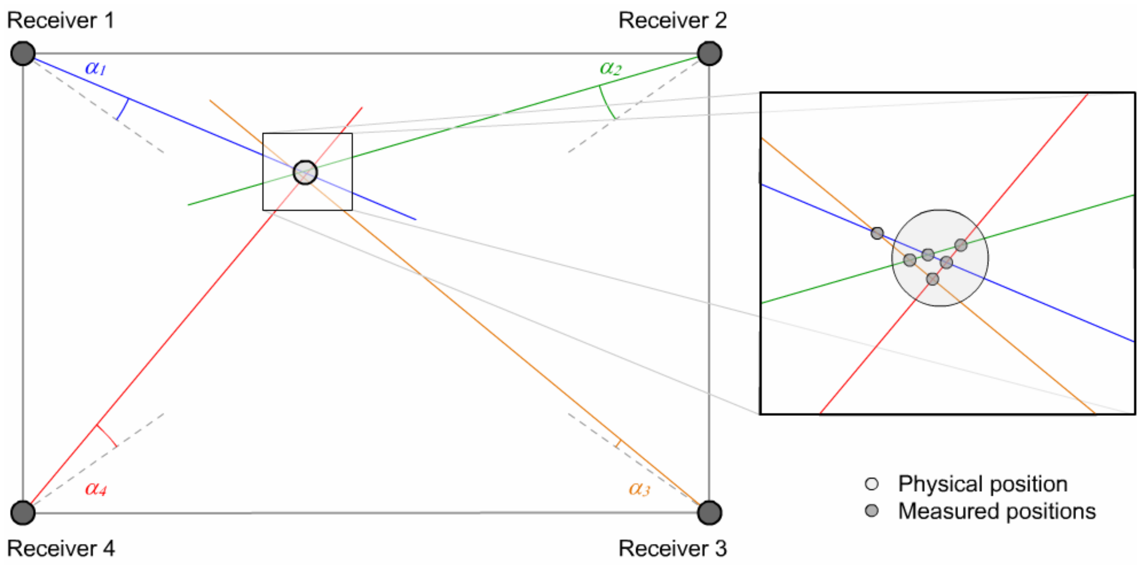


Figure 5.8: The positioning using angulation with four receivers. The magnification illustrates the error potential for a single position. Small circles indicate intersections of a pair of lines from two receivers.

As shown in Figure 5.8, four of those lines do not necessarily intersect in one point. For this, I made assumptions to calculate the position. At first, the intersections of all lines are very close to each other, which leads to the fact that an intersection of two lines must be within a given radius from an intersection of a different pair of lines. If an intersection is not within this radius, the calculated point can either be noise or another point on the wall.

As noted before, triangulation is the calculation of a point by intersecting two virtual lines. Mathematically, these lines can be constructed by transforming the normalized line n_c (representing the camera's normal) which is given as follows:

$$n_c : \vec{x} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \lambda \cdot \begin{pmatrix} u_x \\ u_y \end{pmatrix}$$; c_x, c_y: camera's coordinates, $\begin{pmatrix} u_x \\ u_y \end{pmatrix}$: camera's orientation vector

Since the received angle is the deviation of the camera's normal, we need to rotate n_c about this angle with the camera's center of projection (*cop*) as the rotation center. This leads to the following representation of a new line g_i:

$$g_i : \vec{x} = \begin{pmatrix} c_{x_i} \\ c_{y_i} \end{pmatrix} + \lambda \cdot \begin{pmatrix} \cos\alpha_i & -\sin\alpha_i \\ \sin\alpha_i & \cos\alpha_i \end{pmatrix} \cdot \begin{pmatrix} u_{x_i} \\ u_{y_i} \end{pmatrix} = \begin{pmatrix} p_{x_i} \\ p_{y_i} \end{pmatrix} + \lambda \cdot \begin{pmatrix} u_{x_i} \cdot \cos\alpha_i - u_{y_i} \cdot \sin\alpha_i \\ u_{x_i} \cdot \sin\alpha_i + u_{y_i} \cdot \cos\alpha_i \end{pmatrix};$$

$$g_1 : \vec{x} = \begin{pmatrix} c_{x_i} \\ c_{y_i} \end{pmatrix} + \lambda \cdot \begin{pmatrix} v_{x_i} \\ v_{y_i} \end{pmatrix}, \text{ with } \begin{pmatrix} v_{x_i} \\ v_{y_i} \end{pmatrix} = \begin{pmatrix} u_{x_i} \cdot \cos\alpha_i - u_{y_i} \cdot \sin\alpha_i \\ u_{x_i} \cdot \sin\alpha_i + u_{y_i} \cdot \cos\alpha_i \end{pmatrix};$$

Given another line g_2 with parameters of another camera (including another calculated angle α_2), it is easy to calculate the intersection S of both lines using determinants in the resolving linear system of equations. This leads to the following representation of S:

$$S_{g_1 g_2} \left(\frac{c_{x_1} v_{y_1} v_{x_2} + v_{x_1} v_{x_2} \cdot (c_{y_2} - c_{y_1}) - v_{x_1} v_{y_2} c_{x_2}}{v_{x_2} v_{y_1} - v_{x_1} v_{y_2}} \middle| \frac{-c_{y_1} v_{x_1} v_{y_2} + v_{y_1} v_{y_2} \cdot (c_{x_2} - c_{x_1}) + v_{y_1} v_{x_2} c_{y_2}}{v_{x_2} v_{y_1} - v_{x_1} v_{y_2}} \right);$$

As mentioned above it is necessary to verify the calculated intersection, which is only possible if another camera also has detected a finger. To proof this intersection as a valid point, the distance between this point and a line from a third or fourth camera needs to be calculated. To calculate this, we need to use the normalized form of a line g_i:

- Normal vector of g_i: $\vec{n_i} = \begin{pmatrix} -v_{y_i} \\ v_{x_i} \end{pmatrix}$;
- Normalized form of g_i: $g_i : \frac{1}{\|\vec{n_i}\|} \cdot \left(v_{x_i} y - v_{y_i} x - c_{x_i} v_{x_i} + c_{y_i} v_{y_i} \right) = 0$;

For every point on the line this equation equals 0. For all other points it is equal to the point's distance from the line. This needs to be done for every detected finger (e.g. stripe in the image).

As shown above, only two receivers are needed to calculate the position of one emitter. Unfortunately, this is only valid for one emitter, but this system needs to be capable of detecting up to four fingers (two persons with two hands each). Thus, using two receivers for recognizing two emitters will fail (see Figure 5.9) and the two further receivers are needed for calculation of positions of multiple fingers.

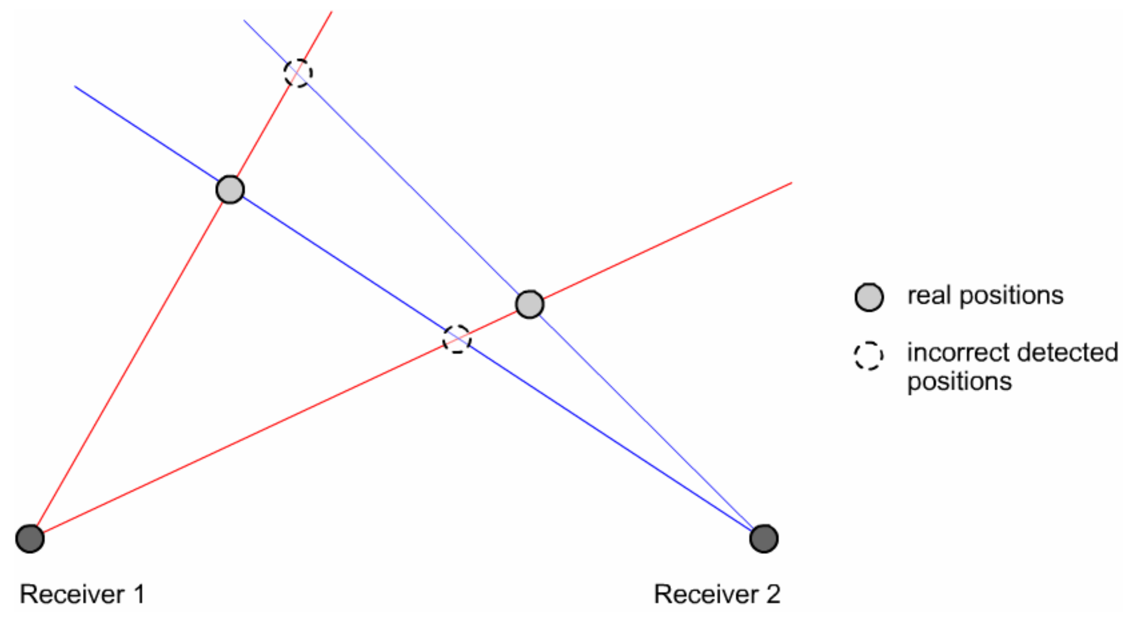


Figure 5.9: Detection of two emitters with two receivers. The real positions cannot be determined and the result will thus be four positions.

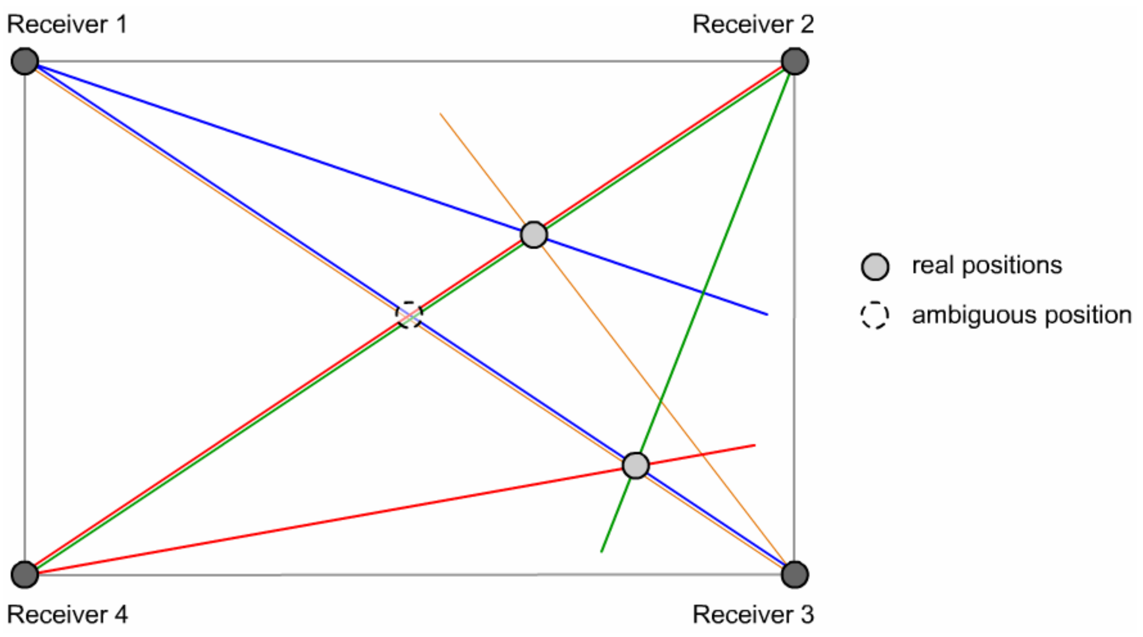


Figure 5.10: Detection of a position where it is not known to the receivers if this position exists. This can only be solved with lateration methods

However, using four receivers only avoids this problem in the case that emitters are not occluded by other emitters. This results from the use of light as signal. Figure 5.10 illustrates the problem of detecting too many positions with four receivers. To avoid these problems I created further criteria regarding a detected intersection. The set of restrictions an intersection must fulfill to be an accepted position is:

- **Clearness of involved lines:** An intersection must provide at least three lines without having a pair of them nearly parallel. This means that involved lines must not have an inner angle of less than a certain angle. Otherwise, the two lines will become one involved line regarding this intersection.

- **Unambiguousness of mapping lines to intersections:** Two intersections must not have the same lines involved (at least three lines must be different). If this happens the more accurate intersection will be taken while ignoring the other one. It is important to let them have one line together due to occluding problems.

Given the second criteria, it is obvious that tracking will not work in the center region of the observed surface. This can be compensated by using another tracking technique such as more receivers only capturing this area.

5.4 Continuity of Finger Movement

The system is now able to detect positions on the wall but it does not know which finger caused the position information. This would not be necessary if the system is restricted to detect positions for clicking operations such as "finger down" and/or "finger up". Since it needs to capture movement of fingers as well, it needs a clear association between measured positions and previously detected fingers.

In practice, there are several ways to accomplish this task. One method, for example, would be using a Kalman [56] filter. The Kalman filter provides a statistical state estimation for dynamic systems while minimizing the least square error. Since it is an iterative process it is predestinated for real-time applications such as this system. This filter is used in airplane navigation where inertial sensors measure acceleration and orientation of the aircraft. These measurements need to be merged with GPS positioning data to get the best possible estimation of its orientation and position.

Due to time constraints it was not possible to use the Kalman filter. Instead I used a simple method called "Dead Reckoning" (*Deduced Reckoning*) [22]. This is not a motion prediction filter but another position detection method. It assumes that – in addition to the last known position – either the velocity or the distance to the next point is known. Having one of these two parameters given automatically results in the other variable. Since the system does not know both of the parameters, they need to be estimated. For this, it assumes that the orientation and speed will not change until the next position is detected by the system. The only time the orientation and velocity are not known is the first detection of a finger. In a simple version of this filter, even the velocity does not need to be known. I will describe in detail how the system uses "Dead Reckoning" combined with the estimation of orientation.

Whenever a position is recognized by the system for the first time the orientation (vector) will be set to its zero value. As soon as the system associates a newly detected position P_{i+1} with a previously recognized finger's position P_i, it replaces the orientation v_i with a new one v_{i+1}:

$$v_{i+1} = \begin{pmatrix} p_{x_{i+1}} - p_{x_i} \\ p_{y_{i+1}} - p_{y_i} \end{pmatrix};$$

To associate a newly detected position with an already recognized one, it is necessary that the new position satisfies several criteria:

- It is within a given range of the estimated position (see Figure 5.11 (a))
- It is the best possible position out of all recognized positions for the old finger
- It is not closer to an estimated position of another finger (see Figure 5.11 (b))

Once a position satisfies all these criteria, the system will store it as a new position and also change the orientation vector for the next prediction. Mathematically, the distance d of a newly recognized position P_{i+1} to an estimated one E_{i+1} is computed as follows:

$$E_{i+1} = P_i + v_i = \begin{pmatrix} p_{x_i} + (p_{x_i} - p_{x_{i-1}}) \\ p_{y_i} + (p_{y_i} - p_{y_{i-1}}) \end{pmatrix} = \begin{pmatrix} 2p_{x_i} - p_{x_{i-1}} \\ 2p_{y_i} - p_{y_{i-1}} \end{pmatrix};$$

$$d = \sqrt{(2p_{x_i} - p_{x_{i-1}} - p_{x_{i+1}})^2 + (2p_{y_i} - p_{y_{i-1}} - p_{y_{i+1}})^2};$$

Figure 5.11 (a) demonstrates the function of the prediction, while Figure 5.11 (b) illustrates a violation of the third criteria during associating two positions to two fingers though the first and second criteria are fulfilled.

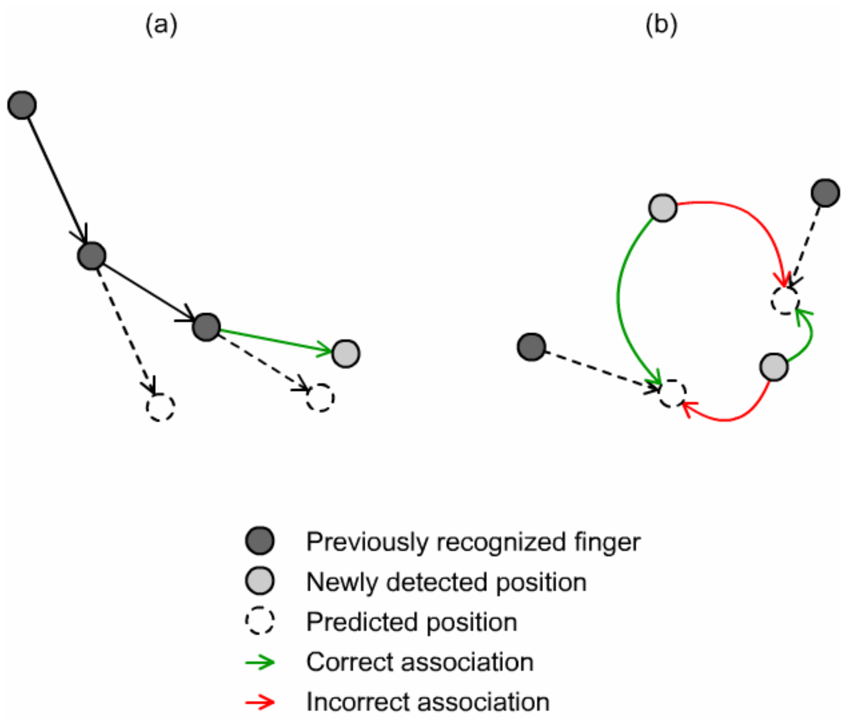


Figure 5.11: (a) shows a correct association of the next finger's position. Figure (b) shows an incorrect association of fingers (red) if only the best match for an old position would be considered. The green lines demonstrate the association considering not only the best match.

Using this mathematical background, I will describe in 7.4 how the steps discussed in the previous sections are implemented in the system. I will also discuss how different components of the system interact with each other to accomplish the complex task of tracking fingers and converting them into virtual mouse positions.

5.5 Calibration Issues

To use all the explained methods it is necessary that the system has calibration that can handle different settings. These include a different wall setup (e.g. different size and/or different display alignment) and various light conditions of the room. I divided the calibration into two parts: camera calibration and system calibration. I will start with the camera calibration.

5.5.1 Camera Calibration

The camera calibration is the first step that needs to be done to ensure a stable operation of the system. Before the cameras can be calibrated in the system, they need to be adjusted with their company-dependent software from Logitech [24]. This includes settings such as brightness, contrast and saturation. Additionally the cameras have global settings such as sharpness, backlight compensation and white balance. While the mentioned parameters affect the direct manipulation of the image, the software provides more settings for the camera's hardware. These include shutter speed, gain and anti-flicker values (50 Hz for Europe, 60 Hz for USA, resulting from different frequencies in their power systems). Setting values to get the desired results must be performed with testing those under real conditions. Table 5.2 shows the different parameters of the manufacturer and notes if they are global or camera-dependent.

Parameter	Values	Effects
Brightness (individual)	[0, …, 10000]	Increases or decreases the brightness in the captured image
Contrast (individual)	[0, …, 10000]	Increases or decreases the contrast in the captured image
Saturation (individual)	[0, …, 10000]	Increases or decreases the saturation in the captured image
Sharpness	[0, …, 10000]	Setting for the sharpness of the image. Low sharpness means a slight blur effect
White Balance	[0, …, 10000]	Sensitizes the camera to the color temperature at the scenario to be captured
Gamma Correction	[0, 1100, 2220]	Improves the balance between the separated color channels
Backlight Compensation	[0, 1, 2]	Indicates whether captured objects will have a glooming effect surrounding them or not
Exposure	[1/100 Hz, 1 Hz]	Shutter speed of the camera
Gain	[0, …, 10000]	Mean ratio of signal output to signal input. It indicates whether the image colors will be amplified or not
Anti-Flicker	50 Hz (Europe) 60 Hz (USA) off	Setting for reduction of flickering due to artificial lights in the captured scene. These flickers result from different frequencies used in the power system

Table 5.2: Shows a summary of the camera's settings provided by the manufacturer. The keyword "individual" indicates that the setting can be different for each camera attached to the system

Once this step is done, the cameras can be calibrated within the system. The system uses several parameters to calibrate its cameras individually. I will give an overview of the different attributes, as well as how they serve together as one calibration:

- **Height:** The height of the rectangle that is to be captured by the system. It is the stripe over the wall's surface where fingers can be recognized and thus gives the distance from the wall where positions can be detected.

- **Calibration image:** This parameter is optional, since the system actually does not need a calibrated image assuming perfect light conditions. However, light conditions might change and a previously captured image can be used to reduce noise in an actually captured image by using a differencing filter (see 5.2).

- **Camera position:** It holds the actual position of the camera and can have four values in the system's case: top left, top right, bottom left and bottom right. Associated to these values are the position and the orientation of each camera.

- **Filter threshold:** This value indicates the luminance level *L* of the used threshold filter. If a pixel's luminance is below the threshold it will colored black. Otherwise it will be colored white. This provides binary images where blobs can be detected.

These values need to be set by the user before the system can work properly. They can be re-calibrated if the system does not run in a correct mode or if it has been transported to a different place.

5.5.2 System Calibration

Besides the cameras, the system also needs a calibration. This can be done once the cameras are capturing useful images and thus recognize positions correctly. One reason for another calibration step is that the physical display size or its resolution might change over time due to better display technologies or different setups.

The basic idea is to match up raw wall positions (given in millimeters) to two-dimensional display coordinates (given in (x, y)-values). Thus, the system needs a transformation matrix to be able to calculate every screen coordinate out of the recognized positions. Assuming that there is only one display embedded in the wall, the calibration takes only one transformation matrix. If there are more displays the system needs to have such a matrix for every display since they will never be aligned perfectly. First, I will describe how to get one transformation matrix.

The first step is to get an association between world coordinates and the corresponding screen coordinates to perform a basis transformation. Since we want to use a single transformation matrix, we need to transform all two-dimensional vectors into homogeneous coordinates by adding a third component with the value 1. The linear mapping can then be written as a multiplication with a 3x3 matrix *A*:

$$v' = A \cdot v; \begin{pmatrix} v_x' \\ v_y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11}v_x + a_{12}v_y + a_{13} \\ a_{21}v_x + a_{22}v_y + a_{23} \\ a_{31}v_x + a_{32}v_y + a_{33} \end{pmatrix};$$

The resulting linear system of equations has three equations and nine unknown variables and is thus not solvable. Since the equation must be true for every point, we can take three arbitrary, linearly independent points into consideration to compute the matrix with the following equation:

$$X' = A \cdot X = \begin{pmatrix} u_x' & v_x' & w_x' \\ u_y' & v_y' & w_y' \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ 1 & 1 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} a_{11}u_x + a_{12}u_y + a_{13} & a_{11}v_x + a_{12}v_y + a_{13} & a_{11}w_x + a_{12}w_y + a_{13} \\ a_{21}u_x + a_{22}u_y + a_{23} & a_{21}v_x + a_{22}v_y + a_{23} & a_{21}w_x + a_{22}w_y + a_{23} \\ a_{31}u_x + a_{32}u_y + a_{33} & a_{31}v_x + a_{32}v_y + a_{33} & a_{31}w_x + a_{32}w_y + a_{33} \end{pmatrix};$$

This linear system of equations is solvable as it has nine equations and nine unknown parameters. To get the transformation matrix A, we need to transform the equation:

$$X' = A \cdot X \Rightarrow A = X' \cdot X^{-1};$$

X^{1}: inverse matrix, given as: $X^{-1} = \frac{\left(X^{adj}\right)^{T}}{\det X}, \det X \neq 0;$

X^{adj}: adjoint matrix of X, given as $X^{adj} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix};$

$A_{ij} = (-1)^{i+j} \cdot D_{ij}$, where D_{ij} is the sub determinant to the element a_{ij};

Once we have the transformation matrix A, we can start calibrating the system. This is done with using four points including their known (x, y)-coordinates on the display. As mentioned above, the calibration only needs three points to work properly. The reason of using a fourth point is to minimize the errors of detection during calibration. Therefore, the system computes four transformation matrices using paired dissimilar sets of three points. It then calculates the mean value of each component of the matrix with the four temporary created matrices:

$$A = \frac{1}{n} \cdot \sum_{\substack{i=1, j=2, k=3 \\ i \neq j \neq k \\ i < j < k}}^{n} A_{ijk};$$

A_{ijk}: calculated transformation matrix of p_i, p_j and p_k;
$i, j, k \in [1, \ldots, n]$;
n: number of calibration points;

As mentioned above, the system should also be capable of handling multiple displays. Therefore, this procedure needs to be done for every display using four calibration points on every display. The resulting matrices will not be merged together in a matrix with average components but need to be used separately in the system. The system needs to determine which matrix is the best to use as a starting point for transformations during runtime. This can be done by getting the resulting mean error of the transformation for every display.

I will briefly describe how the system makes use of the lead matrix. Whenever a new position has been recognized by the system, it first takes the lead matrix to transform it into pixel coordinates. If the pixel is within the display the matrix is associated with, the calculation will stop. If the calculated pixel must be inside another display, the system will take this matrix instead and recalculate the recognized position.

In this system's case I skipped the calculation of the lead matrix due to time constraints, assuming that the center display (out of three displays) will give the best transformation matrix. The procedure for selecting the final transformation matrix is still the same as described above.

Chapter 6

System Setup

One of the major challenges in building the system is the hardware setup. This involves the wall with multiple displays and the cameras attached to the wall to detect the fingers of users. To give an overview of the camera setup an explanation of the wall is needed and will thus be given in the next section. Finally, the complete system setup will be presented.

6.1 The Display Wall

The *display wall* is a large wall with multiple (in this case three) embedded displays. The displays have been implemented as back projected ones with a DLP projector behind the wall for every display. Figure 6.1 will give an idea of how the wall has been set up.

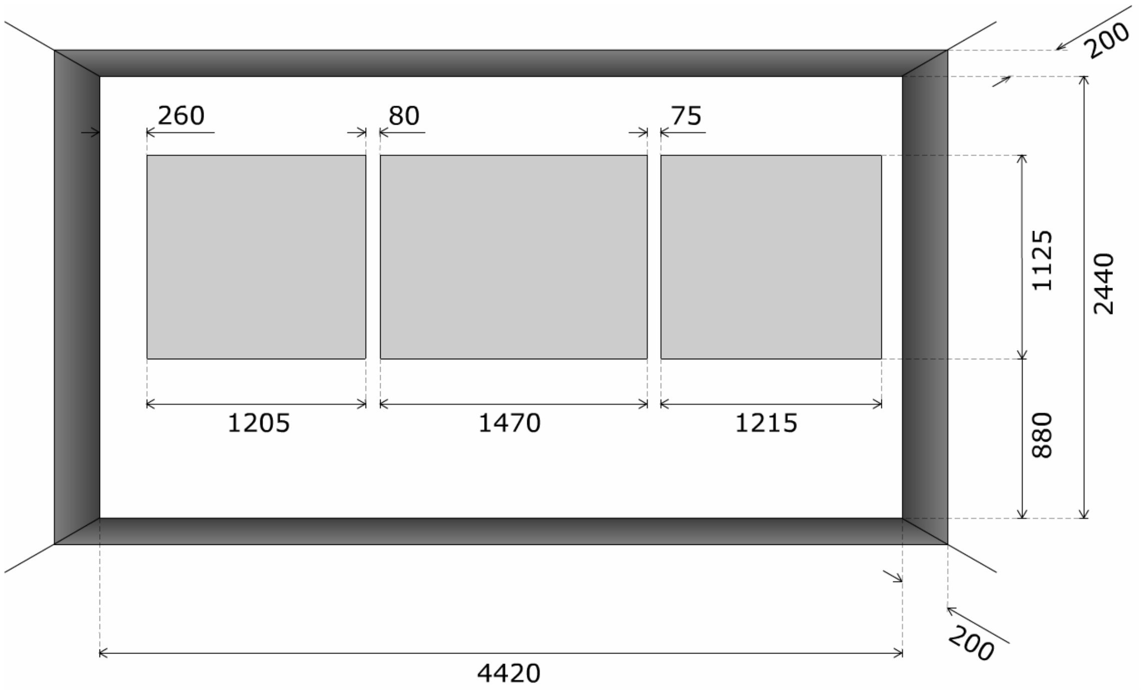


Figure 6.1: Wall setup and display arrangement. Unit of measurement is millimeter.

The center screen is a highly interactive (e.g. touch sensitive) smartboard [41] and will serve as the focus display. The two displays on the left and on the right respectively are “simple” screens which are currently not touch sensitive and will thus serve as context displays. Each of these two displays consists of a back projection foil and Perspex for robustness.

As shown in Figure 6.1 the aspect ratio of both context displays is not the standardized 4:3. Therefore, some of the information given on the screen needs to be cut off in order to get undistorted images. Since I didn't want to lose too much information, the three screens have been combined to one major screen with a continuous region. This means that parts of the cut off information is between the screens and thus is not visible for the user in reality. This gives an advantage that if a user drags an object from one screen to another, the dragged object will not jump between the edges of the screens. Figure 6.2 shows different views of the *display wall* as it has been built in the instrumented room.

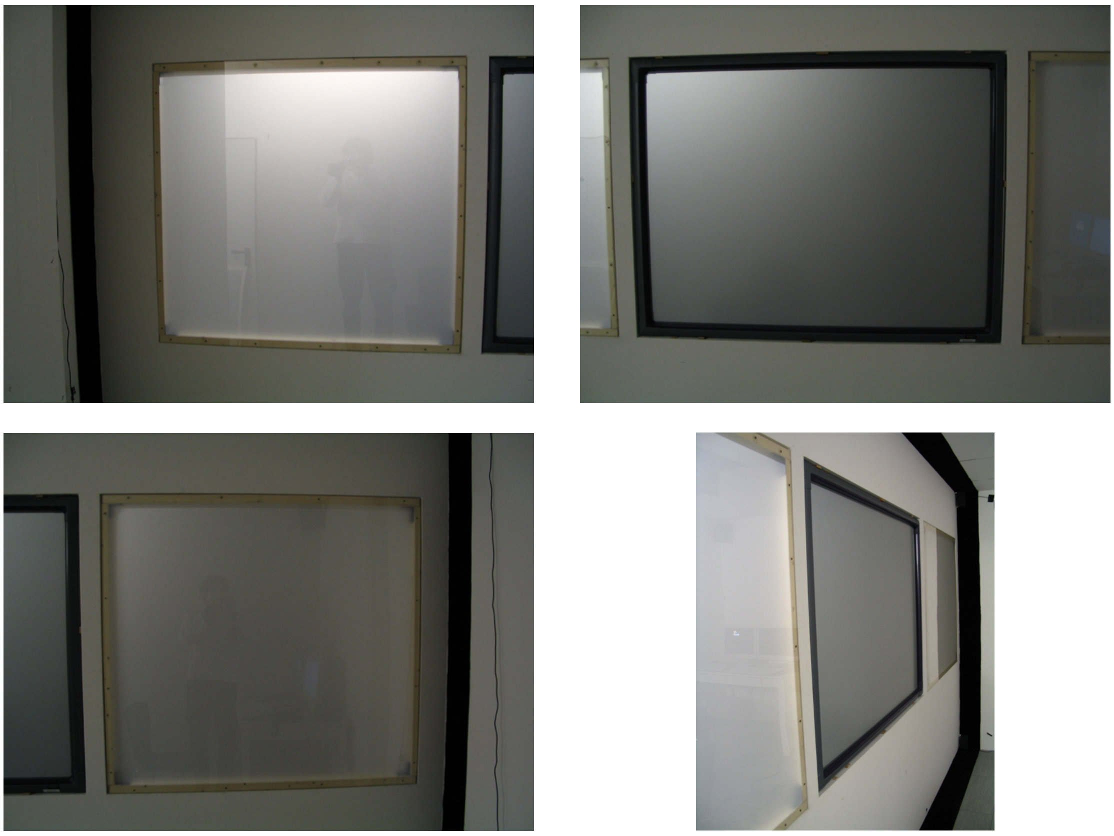


Figure 6.2: Top left shows the left display, top right shows the center display (smartboard), bottom left shows the right display and bottom right shows the complete wall (including the black velvet)

Besides the three displays on the wall, the system will have another display. The fourth display is provided by a steerable projector mounted on the ceiling in front of the *display wall*. This projector is able to project onto any surface in the room which turns all walls, the surface and the ceiling into one large display. Applications using the system will be able to move the projector with pan and tilt values, increase or decrease the zoom factor as well as the focus parameter. It is thus an extension to the three displays embedded in the wall.

Finally, I added black velvet to the side walls of the *display wall* to achieve the results discussed in section 5.2. It had to have a width of at least 200 millimeters to ensure that the cameras do not capture an additional white stripe of the wall.

6.2 Camera-based Tracking System

To track users (i.e. their fingers), the *display wall* has been equipped with four cameras from Logitech [24]. This is a non-trivial task, because several requirements needed to be matched. First, I wanted to have at least two cameras viewing every sub-area of the wall. Second, most of the wall should be covered by the cameras. Third, the coverage area of three or four cameras should be maximized.

With these three requirements, several setups have been examined to find the setting which fits best. Figure 6.3 shows four possibilities of the camera positions and orientations. Finally, setting (*c*) has been chosen.

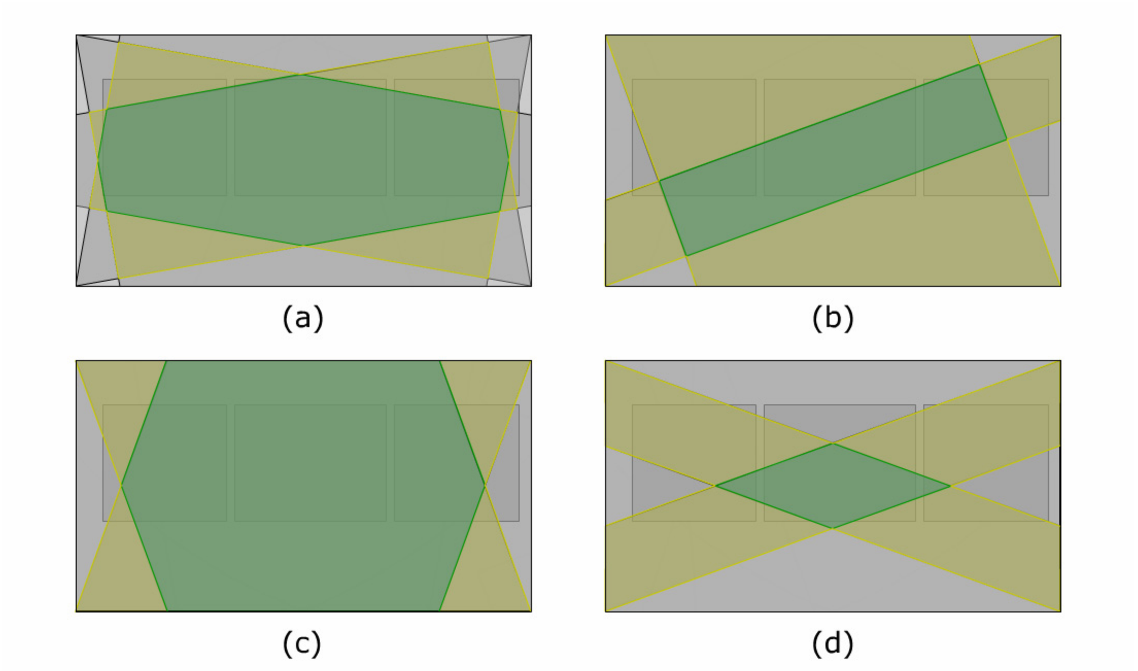


Figure 6.3: (a) shows the angle of aperture of each camera centered (at 45°), (b) visualizes the cameras aligned to one side of the wall (every camera to a different side), (c) shows the four cameras aligned to the bottom and top side of the wall respectively and (d) represents all cameras aligned to the left and right side of the wall respectively. Green areas represent the coverage of all four cameras while yellow areas show the coverage of at least three cameras.

Ideally, the cameras are lowered in the wall so that the captured image is black in one half. I have developed a setting with mirrors to avoid placing cameras in the wall which will be described in detail in the following.

With all cameras in the corners of the wall and the same orientation of both opposite cameras I only had to develop two different camera fixations. I will start with the fixation for the camera in the bottom right and for the camera in the top left respectively. I decided to use a mirror with its longest edge *l* on the wall to make corrections possible due to variations in the production process of the fixations. The different parts of the fixation will be built using wood plates. In the following, I will first introduce the mathematical background that is needed to calculate the mirror's size as well as its position in the fixation. After that, I will give a brief insight of how the fixations have been constructed, including a design drawing.

In the first place, I had to decide on the mirror's shape. It will be a simple triangle instead of a trapeze to preserve a long edge touching the wall. This enables readjusting the camera's position after construction, without being limited to the mirror's size at all. Figure 6.4 shows the desired shape and orientation of the mirror in 3-dimensional space. Corner *A* is in the origin of the coordinate system, corners *B* and *C* need to be calculated.

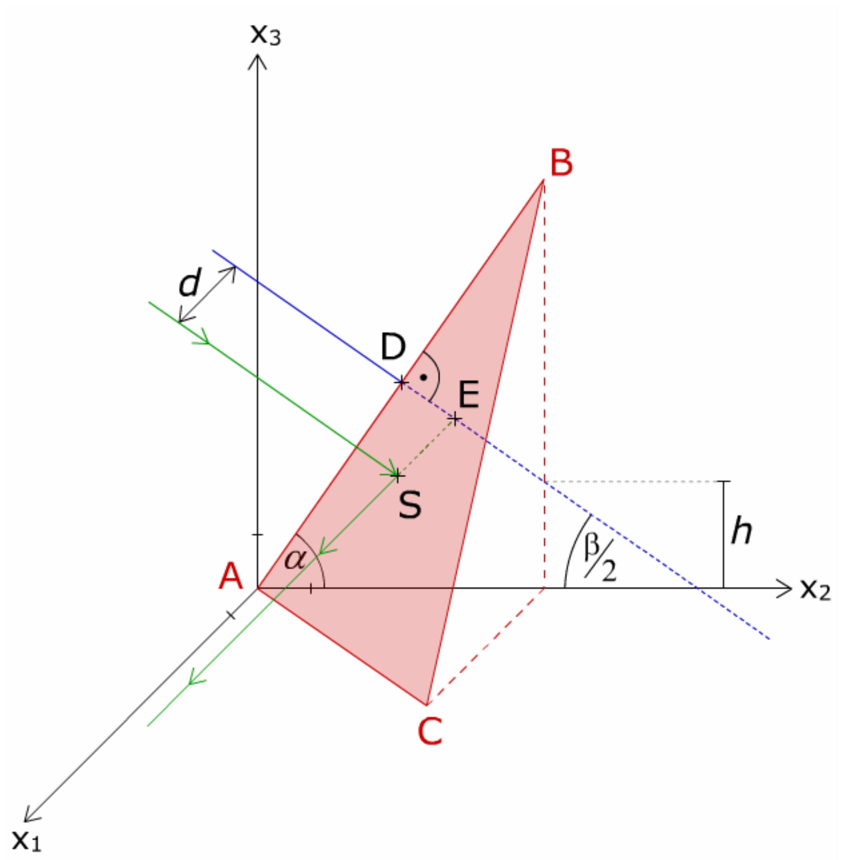


Figure 6.4: The mirror (red surface) in 3-dimensional space with the reflection of a ray of light (green line).

Calculation of point B:
As shown in Figure 6.4 the angle of aperture β of the used camera needs to be known to calculate the position of point B and the camera's orientation. To give a shorter version of this point, we need the angle of inclination α of the line AB:

$$\alpha = 90° - \frac{\beta}{2}; \ \left\|\overrightarrow{AB}\right\| = l; \Rightarrow B(0 \mid l \cdot \cos\alpha \mid l \cdot \sin\alpha);$$

Calculation of point C:
This calculation is a little bit more complicated since we need to arrange the mirror in a way so that the camera is "looking" parallel to the wall surface. If the camera is pointing directly towards the wall (90° between its normal and the wall surface), the angle between the mirror's normal and the camera's orientation needs to be 45° to ensure the law of "Angle of incidence equals angle of reflection". Therefore I set the coordinates of point C to the following:

$$C(x_1 \mid l \cdot \cos\alpha \mid 0);$$

The coordinate x_1 is the only coordinate we need to calculate. First I calculate the normal of the plane of ABC with the vector product of two of the plane's orientation vectors AB and AC:

$$\overrightarrow{n_0} = \overrightarrow{AB} \times \overrightarrow{AC} = \begin{pmatrix} 0 \\ l \cdot \cos\alpha \\ l \cdot \sin\alpha \end{pmatrix} \times \begin{pmatrix} x_1 \\ l \cdot \cos\alpha \\ 0 \end{pmatrix} = \begin{pmatrix} -l^2 \cdot \sin\alpha \cdot \cos\alpha \\ x_1 \cdot l \cdot \sin\alpha \\ -x_1 \cdot l \cdot \cos\alpha \end{pmatrix};$$

Since this normal is dependent of x_1 it is easy to compute this coordinate by forming an angle of 45° between n_0 and the camera's orientation vector which is parallel to (1/0/0). This calculation can be done with the scalar product and normalized vectors:

$$\cos 45° = \frac{\left| \overrightarrow{n_0} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right|}{\left\| \overrightarrow{n_0} \right\| \cdot \left\| \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\|} = \frac{l^2 \cdot \sin\alpha \cdot \cos\alpha}{\sqrt{l^4 \cdot \sin^2\alpha \cdot \cos^2\alpha + (x_1 \cdot l)^2}} = \frac{1}{\sqrt{2}}; \Rightarrow x_1 = \pm l \cdot \sin\alpha \cdot \cos\alpha;$$

In this case the positive solution for x_1 is the desired one. Thus, the triangle describing the mirror's plane with the corners *A*, *B* and *C* is well defined. The next step in calculating the camera's position is the determination of the camera's center of projection (*cop*). For this step I need to add two more parameters to the calculation. First, I need the distance *d* of the camera's orientation (after the reflection) to the wall (see Figure 6.4). The second parameter is the point *a* (line from *A* to *D* in Figure 6.4) on the edge *AB* where the camera points at if *d* equals 0. Both parameters are shown in Figure 6.5.

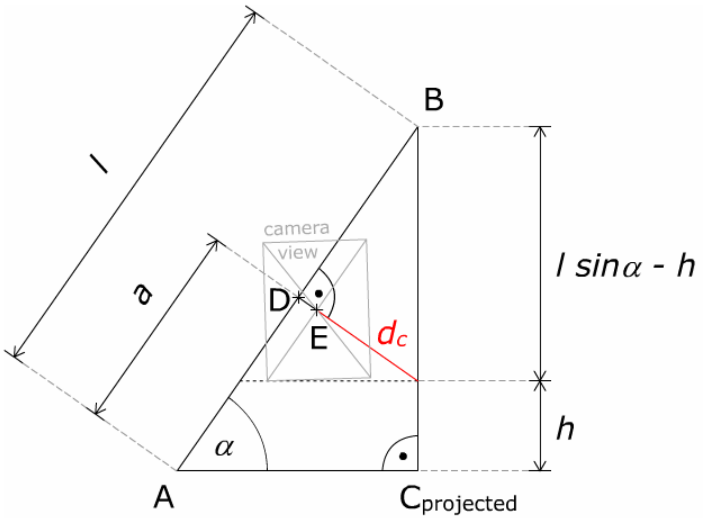


Figure 6.5: x_2x_3-plane projection of the mirror with parameter d_c indicating the distance of the virtual *cop* from point *E* (center of the captured image).

To get the coordinates of the *cop* we need two equations, one for the line *g* and one for the mirror's plane *M*, which are given as follows:

$$g: \vec{x} = \begin{pmatrix} 0 \\ a \cdot \cos\alpha + x_3 \cdot \tan\alpha \\ a \cdot \sin\alpha - x_3 \end{pmatrix} + \lambda \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}; \Rightarrow \begin{aligned} x_g &= \lambda \\ y_g &= a \cdot \cos\alpha + x_3 \cdot \tan\alpha \\ z_g &= a \cdot \sin\alpha - x_3 \end{aligned}$$

$$M: \left(-l^2 \cdot \sin\alpha \cdot \cos\alpha°\right) \cdot x + \left(l^2 \cdot \sin^2\alpha \cdot \cos\alpha\right) \cdot y - \left(l^2 \cdot \sin\alpha \cdot \cos^2\alpha\right) \cdot z = 0;$$

The intersection *S* of *g* and *M* can be computed by replacing *x*, *y* and *z* in the plane's coordinate equation with the three parameterized coordinates x_g, y_g and z_g of *g*. Once we get λ, we are able to calculate the intersection *S*:

$$S\left(x_3 \cdot \cos\alpha \cdot \left(\tan^2\alpha + 1\right) / a \cdot \cos\alpha + x_3 \cdot \tan\alpha / a \cdot \sin\alpha - x_3\right);$$

The first coordinate of *S* needs to match the chosen distance *d*. With this equation it is possible to calculate the parameter x_3, which is:

$$x_3 = d \cdot \cos\alpha; \;\Rightarrow S\left(d \mid a \cdot \cos\alpha + d \cdot \sin\alpha \mid a \cdot \sin\alpha - d \cdot \cos\alpha\right);$$

To get the final position of the camera, the parameter *a* needs to be calculated. Figure 6.5 shows the sketch to get the dependencies of *l* and *a*. Another parameter d_c (distance from the virtual *cop* to the intersection *S*) needs to be introduced. It will be used to get the real distance of the mirrored *cop*. The other parameter *h* characterizes the height from the bottom to the virtual *cop* and thus gives the height of the region which will not be covered by this camera.

$$\frac{l-a}{l \cdot \sin\alpha - h} = \sin\alpha; \;\Rightarrow a = l \cdot \cos^2\alpha + h \cdot \sin\alpha;$$
$$\frac{a \cdot \sin\alpha - h - d \cdot \cos\alpha}{d_c} = \cos\alpha; \;\Rightarrow d_c = l \cdot \sin\alpha \cdot \cos\alpha - h \cdot \cos\alpha - d;$$

With all these parameters calculated, it is possible to give the exact position *P* of the camera in 3-dimensional space.

$$\vec{p} = \begin{pmatrix} l \cdot \sin\alpha \cdot \cos\alpha - h \cdot \cos\alpha \\ l \cdot \cos^3\alpha + h \cdot \sin\alpha \cdot \cos\alpha + d \cdot \sin\alpha \\ l \cdot \sin\alpha \cdot \cos^2\alpha + h \cdot \sin^2\alpha - d \cdot \cos\alpha \end{pmatrix};$$

While the position has been calculated, we need to get the angle of rotation of the camera with the camera's optical axis as the axis of rotation. The maximum angle of aperture is only possible within the diagonal of the camera. Thus, the diagonal must be parallel to the edge of the mirror attached to the wall. This edge has an inclination of α and does not match the angle ε between the camera's diagonal and the shorter side of the camera's projection plane. With the given aspect ratio of 4:3, the angle φ can be calculated as follows:

$$\varphi = \alpha - \varepsilon = \alpha - \arctan\left(\frac{4}{3}\right);$$

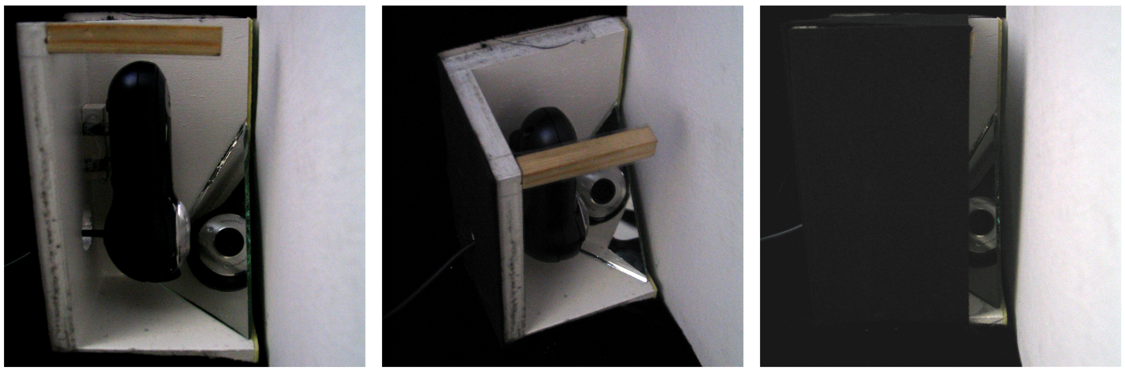


Figure 6.6: Different views of the fixation for one camera. Left and center show the fixation without the black foamed rubber on it. Right shows the final fixation attached to the wall.

Now all the camera's parameters have been defined. The calculation of the parameters of the other two cameras can be done in analogy. Given all the parameters and measurements, I constructed two 3-dimensional models in 3D Studio Max [1] to produce very exact fixations for both, the cameras and the mirrors (see Figure 6.7). After evaluation of those constructions, I built them with wood and fit in the mirror using hot-melt adhesive. Additionally, I attached black foamed rubber to avoid the cameras capturing too much light. The final fixations are shown in Figure 6.6.

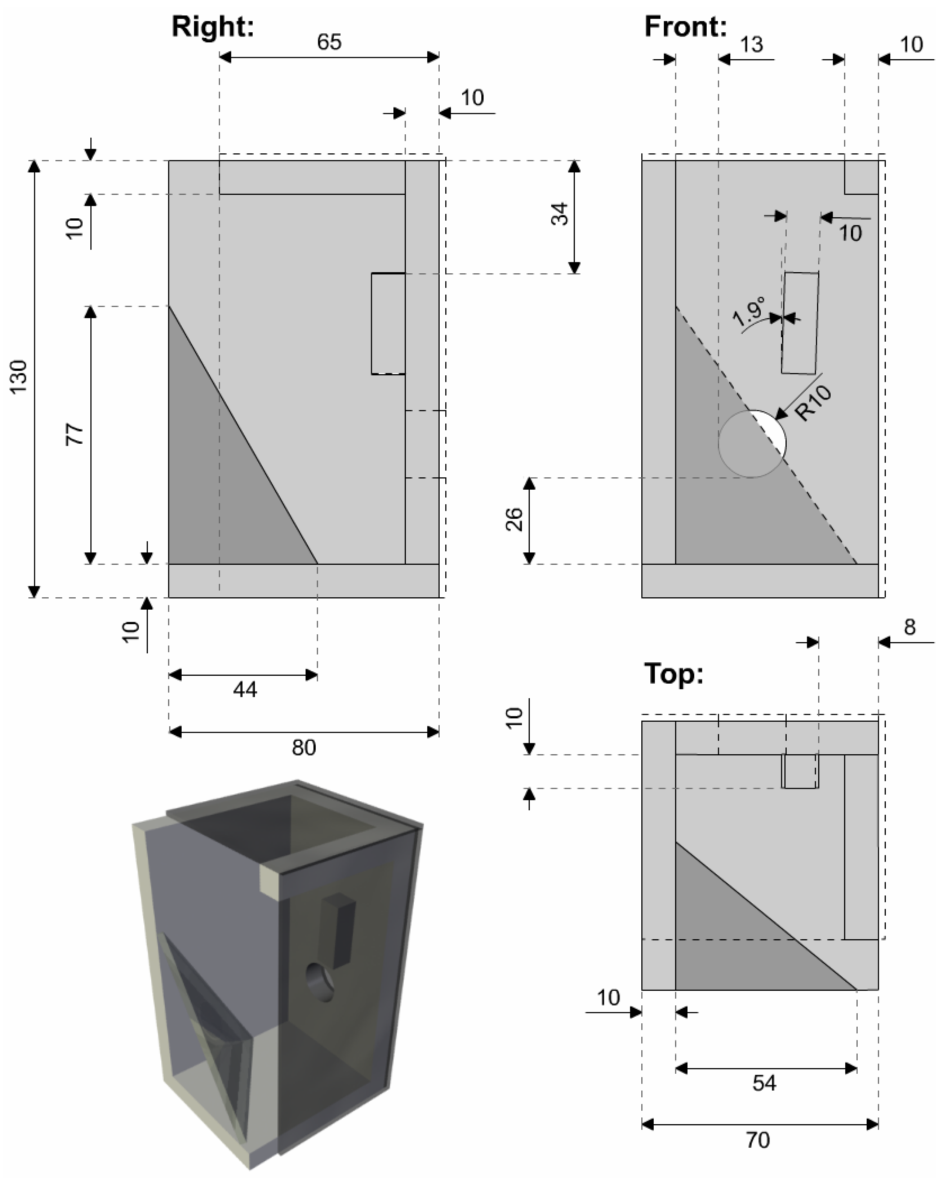


Figure 6.7: Design drawing of the fixation for one camera. Unit of measurement is millimeter.

6.3 Complete System Overview

As I have now described all components separately, I will give a short overview of the complete system setup. This section will show how the different input and output technologies are attached to several computers and how they interact with each other. The reason for having multiple PCs involved has been explained in section 4.1. Additionally, the system needs processing components located on those machines to process multiple input or output. The system uses three computers to serve all I/O components, the display wall PC, the tracking PC and the projector PC. Table 6.1 outlines the different components and which PC they are connected to. For communication, all PCs are connected to each other using a 100 MBit Ethernet LAN. The system uses the Event Heap infrastructure [18] to send and/or receive events.

Component	Attached to
Display Wall:	
• DViT Smartboard	Display wall PC
• Projection screens	Display wall PC
Camera-based tracking system	Tracking PC
Steerable projector	Projector PC

Table 6.1: Summary of all system components

To understand how the single components of the system work together, one needs to know what designation every component has. For the input side, there will be two computers, one for the smartboard and one for the finger recognition on the remaining wall surface. Thus, both components are running on different machines but need to give merged input information to one application. This is why I needed to introduce another component – the so-called "Input Layer". This layer will merge position data received from the smartboard and from the tracking system. It will further associate detected positions with previously recognized fingers. This information will be sent to all running applications that are registered for receiving input events.

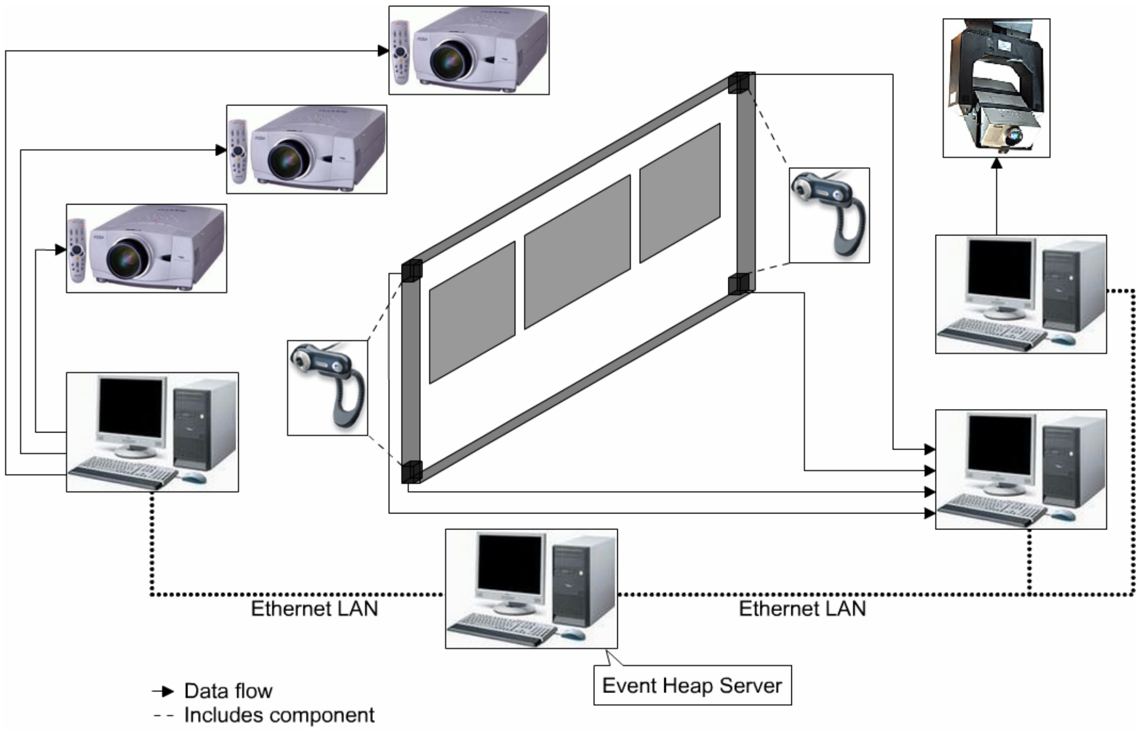


Figure 6.8: Basic setting of the system in the instrumented room (Image sources: [39][25][11][40])

For the output side there might be an arbitrary number of components involved as this room should handle mobile devices as well. In this prototype setting there will be two output devices, the display wall (with projectors) and the steerable projector on the ceiling. These will be registered to receive input events, process them and generate the correct output. Figure 6.8 will illustrate the deployment setting for this preliminary model.

As mentioned above, the system will communicate through the Event Heap infrastructure. It provides a tuple space which can be used by an application is able to send and/or receive events. Events have several definable parameters as listed in Table 6.2.

Parameter	Description
Event type	Name of the event to be sent and/or received
Event fields	List of field names included in the event
Event values	List of values connected to the event fields (only important, if an event has to be sent)
Event class types	List of class types indicating the type of a specific value
Time to live	The time an event will reside on the server

Table 6.2: Most important definable parameters of events used in the Event Heap infrastructure

If a component wants to send an event, it will connect to the Event Heap server and submit its newly created event. To receive events, a component needs to register for a specific event beforehand to let the server know that it wants to get events that match this template. Multiple components are allowed to register for the same events at any time. Once the server receives an event from a component, it immediately pushes it to all registered components.

It is important to know which events will be fired during runtime. Thus, they need to be designed carefully to avoid other running applications being flooded by those events. Figure 6.9 will give a short overview of the different events created during runtime and which event an application has to be registered for to use the input produced by the display wall.

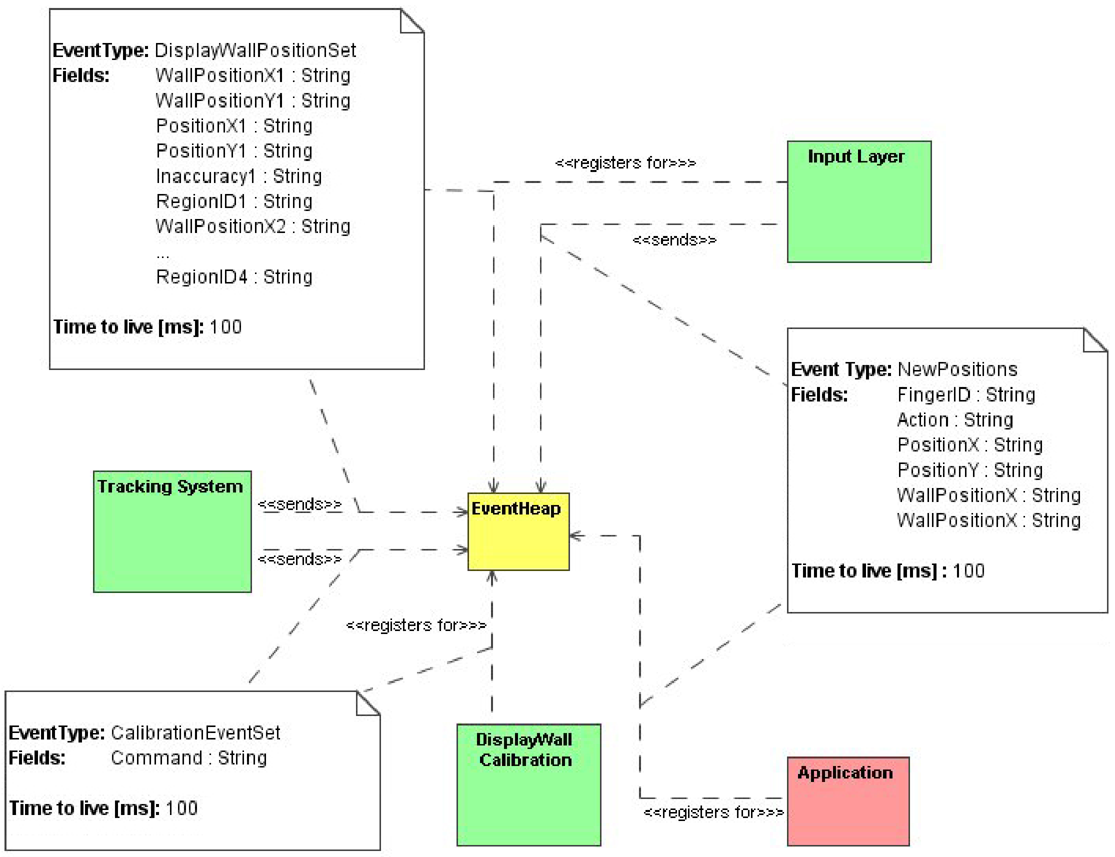


Figure 6.9: This shows the different components of the system (green), the Event Heap (yellow) and additional applications using the display wall (red). The tracking system runs on a separate machine whereas all other components can be together on a single computer.

The shown model is the basic operation for two different machines (steerable projector has been left out) but is easily extensible to more components and/or devices. As I have illustrated how the different technologies are able to work together, I will describe how those interact with each other on implementation level.

Chapter 7

Implementation

In this chapter I will describe the implementation of the concepts explained in chapters 5 and 6 in increased detail. First, I will give a short overview of the tracking system including a detailed description of the events used within. I will continue with the two calibration methods. Subsequently, the position recognition using captured images will be demonstrated followed by a description of how positions are associated to finger positions.

7.1 System Overview

As described in section 6.3 the complete system is distributed to several machines using the Event Heap as communication infrastructure. Thus, in this section I will describe the implementation of the basic detection engine as well as the network communication. I will start with the local process of recognizing positions. Subsequently, I will describe in detail how the network communication takes place.

7.1.1 The Tracking Engine

To understand the process of detecting positions, I will first show the involved components used for this task. The basic detection engine is separated in four packages (see Figure 7.1). I will not describe the implementation of the graphical user interface in this section as it will be added to this model in the following section. The main package is `Control` and it contains only one class, `ApplicationControl`, which serves as the controller for the engine.

Another package called `Devices` includes several classes needed for the tasks of handling the attached webcams and for capturing raw images from their image stream. It contains three classes used for creating capture devices and handling them. The `DeviceManager` is the first class that will be created after the system's startup. It immediately creates four instances of the class `CaptureDevice` using the `DeviceFactory` class. The creation includes loading the calibration data (if available) or creating default ones. The device will then calculate an array with detected lines to make the detection faster during runtime. This list contains a line for each possible position of a blob within the captured image. After this is done, the main task of capturing images will be created using the DirectShow[6] architecture which gives the ability to start or stop the device, as well as grab images from its capture stream. Grabbing images is provided by using a callback method which reads the image directly from main memory. After the image has been captured, it needs to be rotated and cropped since the cameras' diagonal is observing the stripe over the wall's surface. This will be described in section 7.3 in detail.

[6] DirectShow is part of the DirectX SDK extras package developed by Microsoft

The third package is called `Detection`. It is comprised of all classes used for the detection and calculation of positions. It provides an interface called `IDetector` for image manipulation which allows interchangeable detectors using different filters during image processing. The only implementing class at this stage is the `DifferenceDetector`, which uses the methods described in section 5.2 to gather relevant data (e.g. blobs) in the captured image. This class also provides methods to change the filters' attributes such as the threshold value or the difference image. The basic class for detecting positions is the `Analyzer`. It runs within a separate thread and uses the detected lines from the cameras to calculate the intersections found on the wall as described in section 5.3. Two further classes provide structures for recognized positions on the wall to ensure a correct detection. One is `DetectedLine` which comprises a virtual line (including its normalized form) calculated by a capture device combined with an identification number for uniqueness. The other one is `DetectedIntersection` which holds up to four detected lines as well as a position on the wall. It also contains a value of inaccuracy telling the engine whether this is a possibly good intersection or not. The process of calculating positions in the system will be described in section 7.3.

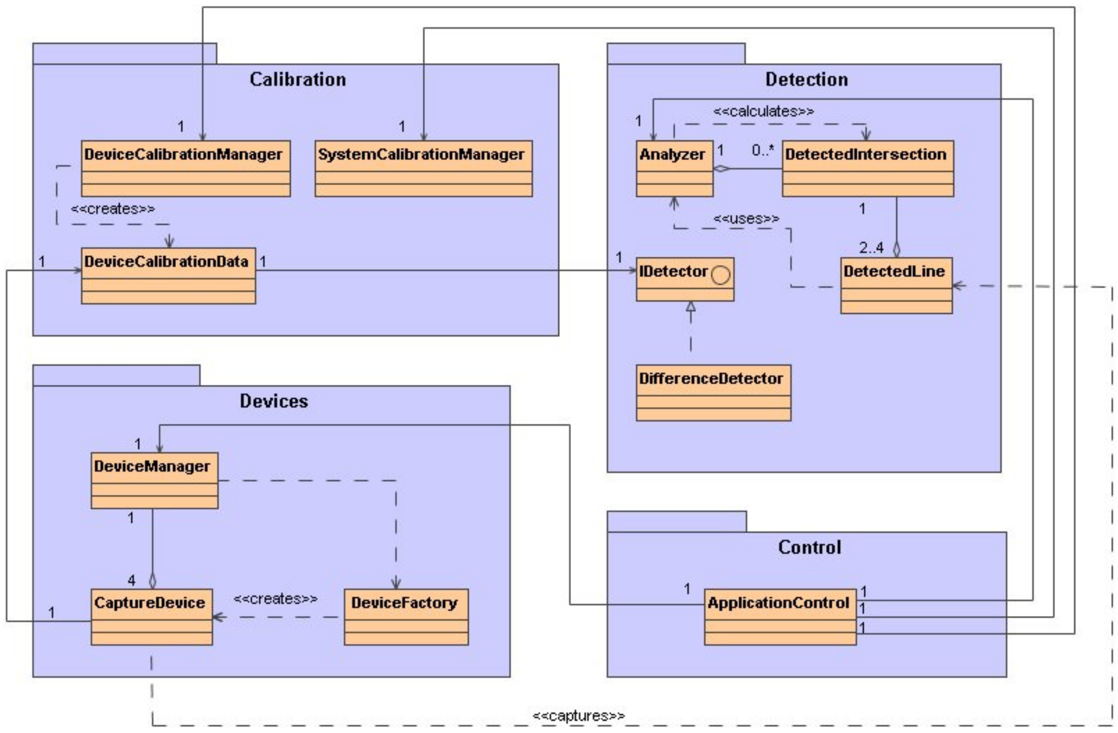


Figure 7.1: Basic detection engine implemented in the tracking system

The last package is the `Calibration` package. It comprises managing classes for both the devices and the system as well as a data class to load and/or store calibration data. The system itself does not need a data class since it only has the transformation matrices which can be stored in the `SystemCalibrationManager`. The procedure of calibration will be described in section 7.2 in more detail.

Now that I have described all components of the tracking system, I will briefly demonstrate how they act together during runtime. The basic principle used is the event system provided by the .NET Framework[7] from Microsoft [29]. The two base classes `EventArgs` and `EventHandler` have been extended since they originally did not contain any information and thus work as a notification for a registered component. The overridden events for the displaying components already contain the images to be drawn, whereas the events for the analyzing

[7] Version 2.0 has been used for implementation and runtime

engine are only comprised of the detected lines of the cameras. The notified components are aware of the events they receive and can thus use the contained information directly. Figure 7.2 shows the components sending and receiving events and characterizes the structure of the different events being used.

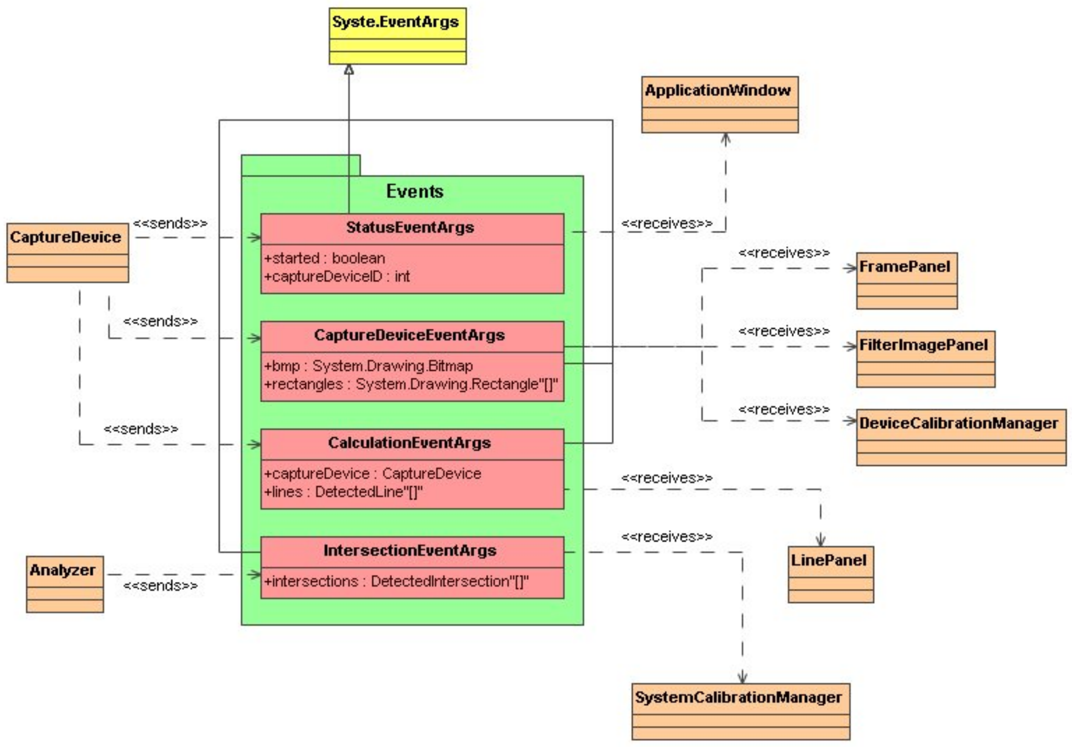


Figure 7.2: Shows the basic event system used for the main tracking system. The components shown on the left send events and the ones on the right receive them.

With the described model it is possible to track objects (e.g. fingers) on the wall's surface. Since more components will be involved in the complete system, I will describe the network communication model being used in the complete application in the following section.

7.1.2 Network Communication

As mentioned before, the tracking system needs to communicate with other applications running on different machines. I used the Event Heap as communication infrastructure since it already is being used in the current setting. The major problem I had is that it is a Java implementation. Thus, I needed to create a bridge between the tracking engine written in C# and the Event Heap. To accomplish this, I added another package called `Network` containing all relevant classes for the described task. Figure 7.3 shows the package and the corresponding Java packages in detail.

The basic idea is that the tracking system first initiates a Java virtual machine through the class `ProxyConnection` using the `Process` class of the .NET Framework. Once the JVM has been started, it creates a socket to communicate with the host application. Whenever the tracking system wants to send data to the Event Heap, it basically sends the data to the Event Heap proxy in a certain way. The proxy itself converts the data into events and forwards them to the Event Heap, where other applications might be registered to receive these. The communication between the `ProxyConnection` class (implemented in C#) and the `EventHeapProxy` class (implemented in Java) is realized by using string objects, which can be easily sent through input and output streams of sockets in both implementations.

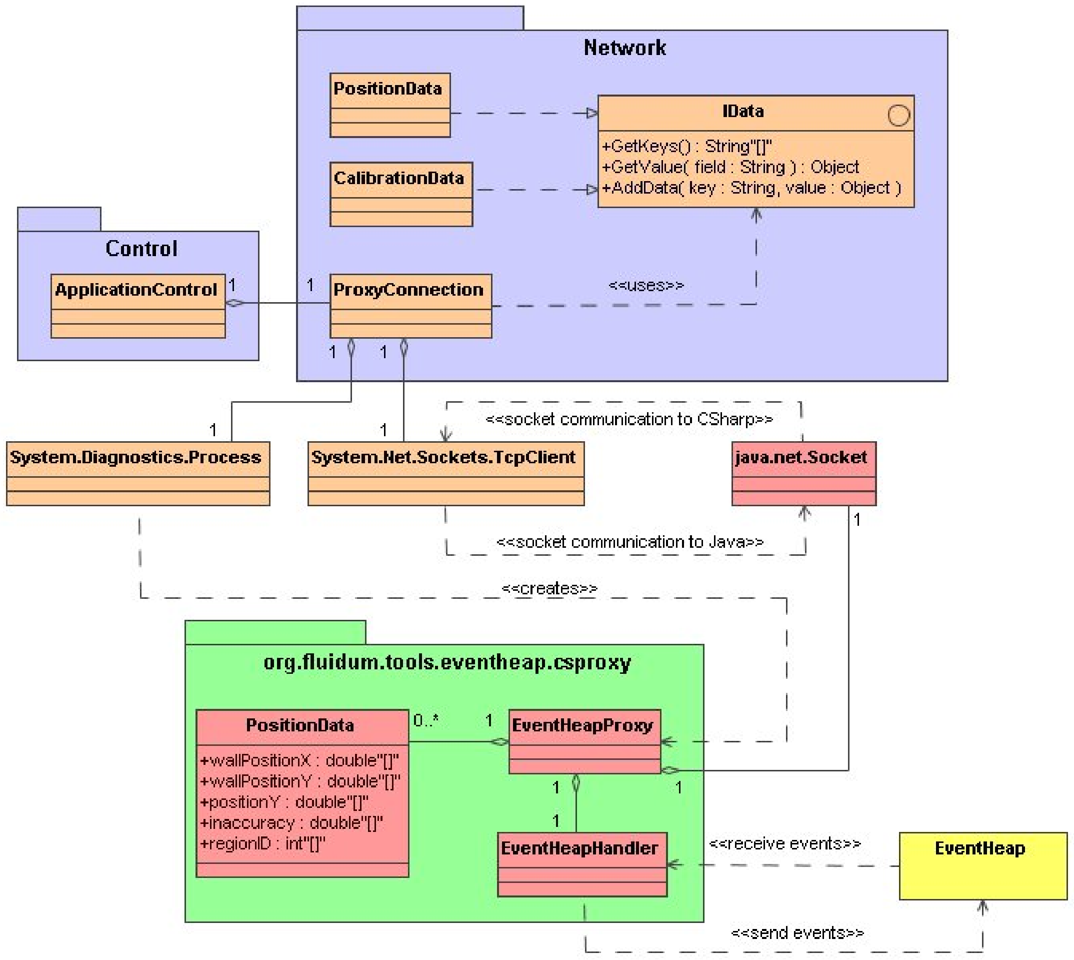


Figure 7.3: Detailed class diagram of the network package. Green packages, red classes and the Event Heap have been implemented in Java, the other ones in C#.

There are two types of events the tracking system wants to send to other applications. One is the `PositionData` event, which includes raw position information of four points currently detected on the wall. It also contains additional information, such as screen coordinates, inaccuracies and region identifications for each point. The structure of the string to be sent is as follows:

```
for each i ∈ [1, …, 4]:
    WallPositionX[i]=[value : double]&
    WallPositionY[i]=[value : double]&
    PositionX[i]=[value : double]&
    PositionY[i]=[value : double]&
    Inaccuracy[i]=[value : double]&
    RegionID[i]=[value : int]
```

Another event is called the `CalibrationData` event. It only contains three commands for the calibration procedure. These commands are explained in detail in section 7.2.2. The structure of the string being sent to the Event Heap proxy is as follows:

```
Command=[value : string] where [value] must be:
    - 'StartCalibration' or
    - 'StopCalibration' or
    - 'NextCalibrationPoint'
```

The Event Heap proxy parses the commands and (if valid) sends them to the Event Heap as a specific event. The type of this event is dependent on the type of command arriving at the proxy. I have created two types of events that can be received by different applications. Those are listed in Table 7.1.

Event type	Fields	Class types	Time to live [ms]
DisplayWallPositionSet	WallPositionX1	String	100
	WallPositionY1	String	
	PositionX1	String	
	PositionY1	String	
	Inaccuracy1	String	
	RegionID1	String	
	WallPositionX2	String	
	…	…	
	RegionID4	String	
CalibrationEventSet	Command	String	100

Table 7.1: Listing of the two events used by the tracking engine

Since the tracking system should operate as close to real-time as possible, the value for an event's lifetime on the Event Heap server needs to be very small. I decided to use 100 milliseconds to avoid having events being sent which are not close to the current action on the wall. This value is the highest possible delay where an application still seems to be real-time to the user [2]. For the processing engine it is not important if two positions are consecutive, or if a position event has been left out.

The model of communication will be extended in the following sections as more components for processing input will be added. I will go on with the calibration procedure used to ensure a high performance of the system regarding execution speed and accuracy.

7.2 Calibration

There are two calibration procedures implemented in the system. One only affects the attached devices and calibrates the offline parameters. The other one calibrates the online parameters of the system and thus needs the display wall as well. First, I need to exemplify the graphical user interface of the tracking system which provides the user with tools for calibration. With this, s/he can also observe the system's performance during runtime. Figure 7.4 summarizes the implementation of the graphical user interface used in the tracking system.

The important part for the camera calibration procedure is the `CameraConfigWindow`, which gives the user all controls to calibrate one of the cameras at a time. It holds an `ICaptureDevicePanel`, which receives events from a `CaptureDevice`, including the image to be drawn. Thus, the user also sees the currently captured image to decide whether the camera is adjusted correctly or not. The user then is able to customize the parameters described in section 5.5.1.

For the interactive system calibration, communication with the display wall component is required to show the calibration screen on the wall. This will be described in section 7.2.2 in detail. I will start with a description of the camera calibration.

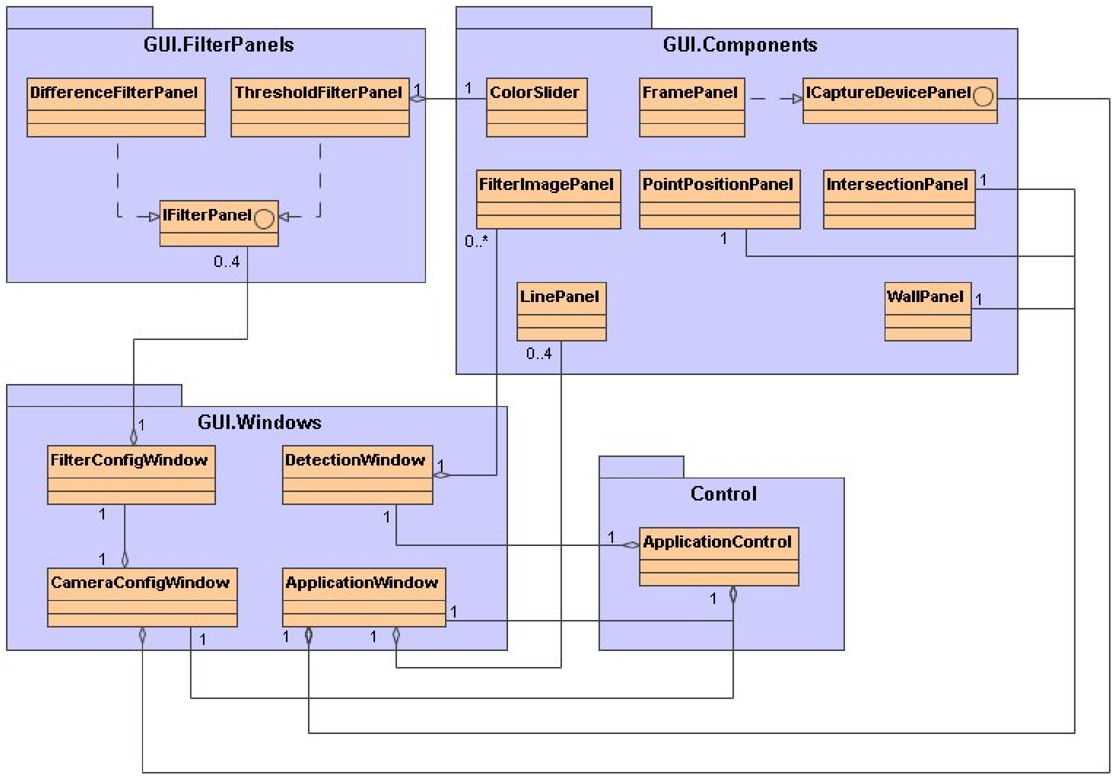


Figure 7.4: Basic illustration of the graphical user interface of the tracking system

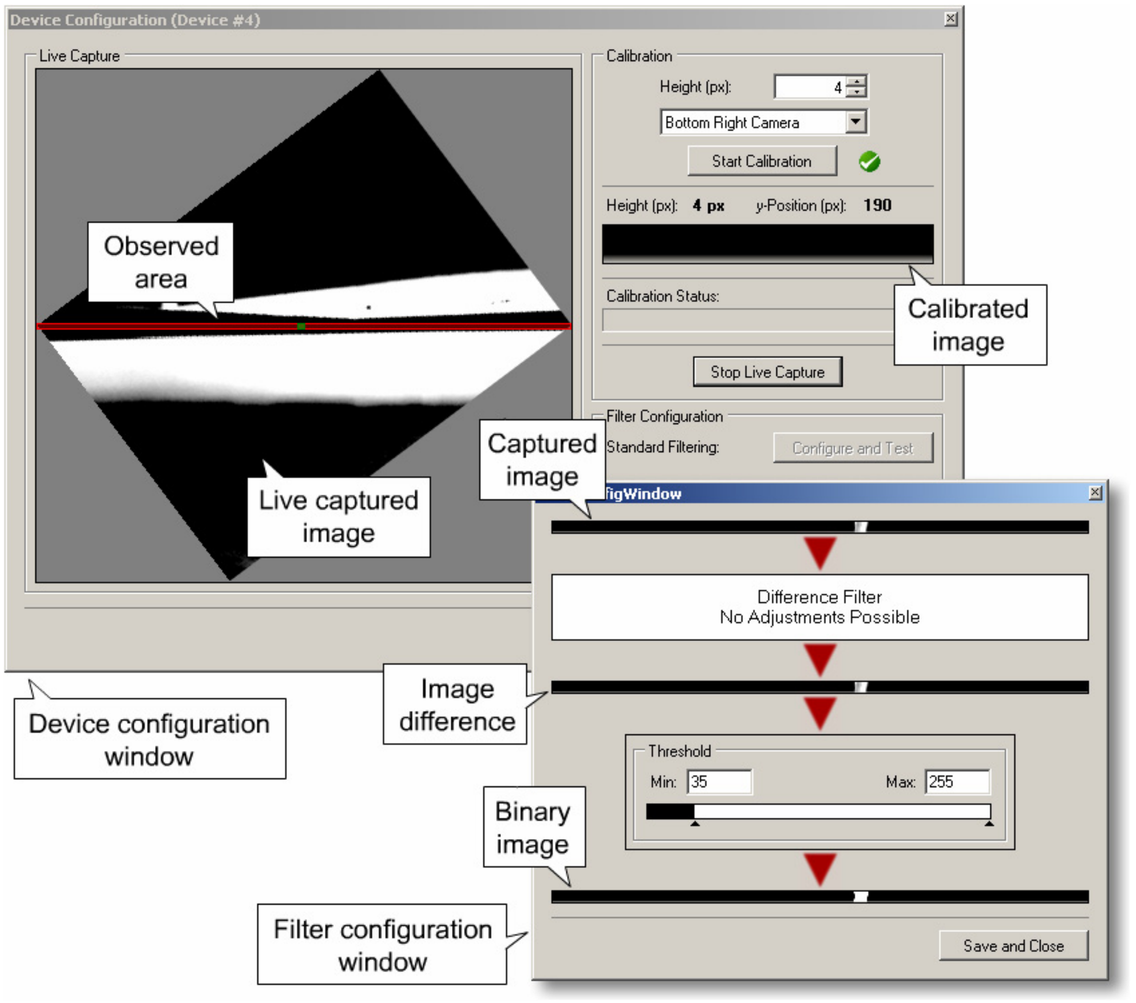
7.2.1 Camera Calibration

As described in section 5.5.1 the user needs to configure the cameras before s/he can start using the system. This is done manually for every camera attached to the tracking system. The calibration of a capture device can be started using the main menu of the application screen by selecting the camera which needs to be calibrated. Once the user has chosen the desired capture device, a window (see Figure 7.5) will appear, where s/he can adjust all relevant parameters, as well as calculate the image used for differencing.

After the user has set the parameters to his or her desired values, s/he can start the calibration by clicking on the “Start Calibration” button. This will immediately create and start the `DeviceCalibrationManager` which will also receive image events from the capture device connected to the window. Once started, it will capture a predefined number of frames and calculate the resulting average image as the calibration bitmap. Using more frames reduces noise in the calibration bitmap at single pixels, since noise does not occur throughout all pixels within one frame. The calibration image will be shown immediately after the calibration process has finished and can thus be used to configure the filter for recognition.

After the calibration is done, the user is able to configure the filter settings for the detection. For this, another window is shown with live captured images as well as a color slider for the threshold used during runtime (see Figure 7.5). The user can set the value interactively by using trial and error. After this is done, the user can quit the calibration for this device and go on with the next one. After closing the configuration window, the system stores all adjusted data permanently into an XML based file. All calibration files will be loaded whenever the system starts and thus the cameras do not need to be adjusted again. If the system does not run correctly anymore, the user can re-calibrate the cameras, resulting in new calibration files. An example of one of those files is given in the following.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Calibration date="2006-2-7" time="13:33:24">
   <Device id="0" />
   <Position relative="TOP_LEFT_CORNER" />
   <Path>calibration/device_0.bmp</Path>
   <RectPositionY>190</RectPositionY>
   <Rectangle>
        <Width>400</Width>
        <Height>4</Height>
   </Rectangle>
   <Filter minThreshold="50" maxThreshold="255" />
</Calibration>
```



Height (px): 4

Bottom Right Camera

Height (px): 4 px y-Position (px): 190

Standard Filtering:

Difference Filter
No Adjustments Possible

Threshold
Min: 35 Max: 255

Figure 7.5: Screenshot of the device configuration window and the filter configuration window. Both give the user all necessary controls to calibrate and adjust device related data.

Once the calibration has been done for all attached capture devices, the user can start using the basic function of the system which is the detection of real-world coordinates on the wall. As s/he wants to interact with an application running on the display wall, the system needs an interactive calibration which ensures that the system is able to determine the screen coordinates of a detected position. This will be described in the next section.

7.2.2 Interactive System Calibration

This calibration step is the first action that needs more than just the tracking system. The basic idea is that a user points on predefined visual points projected directly on the wall. The system needs to recognize the positions and then calculates the transformation matrix as described in section 5.5.2.

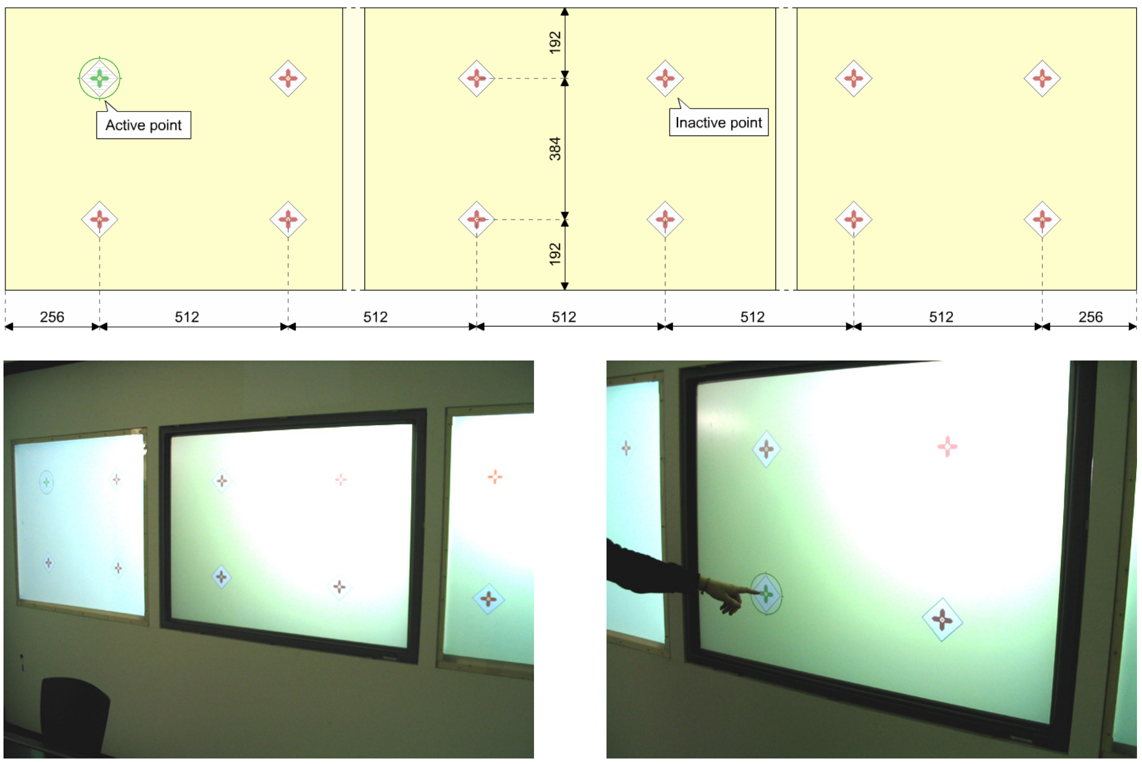


Figure 7.6: Top shows a sketch of the calibration screen (projected onto all three displays) with the first point as active point. Unit of measurement is pixel. Bottom left shows the calibration screen projected onto the displays and bottom right demonstrates a user calibrating the system.

If a user wants to start the system calibration, s/he is able to do this by selecting it from the main menu of the application screen. A second application needs to run on the computer managing the display wall (e.g. the projectors). This is a permanently running application listening for events to start the calibration. Whenever the user selects the system calibration to be started, the tracking system sends an event to the Event Heap, indicating that the calibration screen needs to be shown on the display wall. Figure 7.6 exemplifies this screen directly after starting the calibration and shows a person using the calibration interactively.

The screen coordinates of the points (see Figure 7.6) need to be known to both the calibration screen and the tracking system. Once the calibration procedure has been started the tracking system initializes a new instance of the `SystemCalibrationManager`, which handles the main part of the calibration. It registers itself for receiving events from the `Analyzer`, each of them containing a `DetectedIntersection`. After gathering a predefined number of intersections, the system will compute an average point to reduce errors during the detection. It then sends another event through the Event Heap, telling the calibration application to set the next calibration point as the active one. The user needs to iterate this procedure until the last point has been processed.

After completing the interactive part of the calibration, the system sends a final event to the display wall to hide the screen. Internally, the transformation matrices are calculated as described in section 5.5.2. This is done with a mathematical library containing vectors, lines and points in both, two and three dimensions. It also comprises methods for matrix manipulation and geometrical calculations. Figure 7.7 shows a class diagram of how the mathematical library has been implemented.

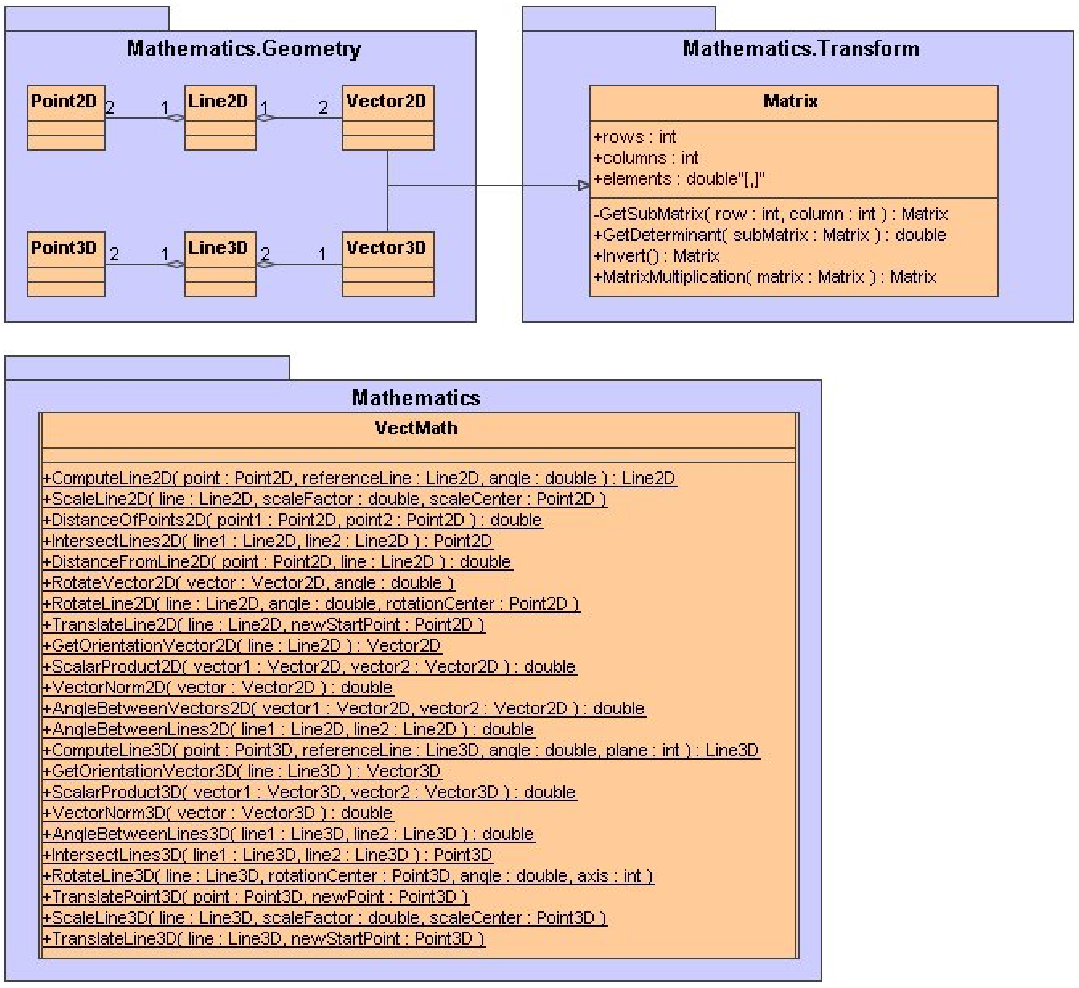

Mathematics.Geometry
Point2D
Line2D
Vector2D
Point3D
Line3D
Vector3D
2
1
1
2
2
1
2
1
Mathematics.Transform
Matrix
+rows : int
+columns : int
+elements : double"[,]"
-GetSubMatrix(row : int, column : int) : Matrix
+GetDeterminant(subMatrix : Matrix) : double
+Invert() : Matrix
+MatrixMultiplication(matrix : Matrix) : Matrix
Mathematics
VectMath
+ComputeLine2D(point : Point2D, referenceLine : Line2D, angle : double) : Line2D
+ScaleLine2D(line : Line2D, scaleFactor : double, scaleCenter : Point2D)
+DistanceOfPoints2D(point1 : Point2D, point2 : Point2D) : double
+IntersectLines2D(line1 : Line2D, line2 : Line2D) : Point2D
+DistanceFromLine2D(point : Point2D, line : Line2D) : double
+RotateVector2D(vector : Vector2D, angle : double)
+RotateLine2D(line : Line2D, angle : double, rotationCenter : Point2D)
+TranslateLine2D(line : Line2D, newStartPoint : Point2D)
+GetOrientationVector2D(line : Line2D) : Vector2D
+ScalarProduct2D(vector1 : Vector2D, vector2 : Vector2D) : double
+VectorNorm2D(vector : Vector2D) : double
+AngleBetweenVectors2D(vector1 : Vector2D, vector2 : Vector2D) : double
+AngleBetweenLines2D(line1 : Line2D, line2 : Line2D) : double
+ComputeLine3D(point : Point3D, referenceLine : Line3D, angle : double, plane : int) : Line3D
+GetOrientationVector3D(line : Line3D) : Vector3D
+ScalarProduct3D(vector1 : Vector3D, vector2 : Vector3D) : double
+VectorNorm3D(vector : Vector3D) : double
+AngleBetweenLines3D(line1 : Line3D, line2 : Line3D) : double
+IntersectLines3D(line1 : Line3D, line2 : Line3D) : Point3D
+RotateLine3D(line : Line3D, rotationCenter : Point3D, angle : double, axis : int)
+TranslatePoint3D(point : Point3D, newPoint : Point3D)
+ScaleLine3D(line : Line3D, scaleFactor : double, scaleCenter : Point3D)
+TranslateLine3D(line : Line3D, newStartPoint : Point3D)

Figure 7.7: Class diagram of the math library used to process all geometrical calculations

Once all the calculations are done, the system stores the final results permanently in an XML file. In the same way as with the camera calibration, this file will be overwritten whenever the user re-calibrates the system. An example of such a file is given below (the field "id" indicates to which display the matrix is associated to):

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Calibration date="2006-2-7" time="13:33:24">
   <MatrixElement id=″0″ row=″0″ column=″0″ value=″1,0″ />
   <MatrixElement id=″0″ row=″0″ column=″1″ value=″0,0″ />
   <MatrixElement id=″0″ row=″0″ column=″2″ value=″0,0″ />
   <MatrixElement id=″0″ row=″1″ column=″0″ value=″0,0″ />
   <MatrixElement id=″0″ row=″1″ column=″1″ value=″1,0″ />
   <MatrixElement id=″0″ row=″1″ column=″2″ value=″0,0″ />
   <MatrixElement id=″0″ row=″2″ column=″0″ value=″0,0″ />
   <MatrixElement id=″0″ row=″2″ column=″1″ value=″0,0″ />
   <MatrixElement id=″0″ row=″2″ column=″2″ value=″1,0″ />
   …
</Calibration>
```

Together with the camera and the system calibration, the system is now fully functional. In the next section I will describe how the system uses the calibration data during runtime to enable a completely interactive wall.

7.3 Position Recognition

In this section I will describe how the tracking system actually works in runtime. I will start with the position recognition (including the image rotation and cropping) and continue with the mapping of positions to fingers in the next section. The detection of intersection has been implemented according to the requirements and concepts described in chapter 5.3.

Once the system has been activated by pressing the "Start System" button on the main application screen, the `DeviceManager` will initialize all devices sequentially. The reason for not starting them all at once is that the devices consume a high bandwidth of the USB bus during activation. Thus, after starting one device, the device manager waits until it has been initialized completely before activating the next one. During this process, the user receives visual feedback of the system's current state indicated by virtual lamps in the application screen. Once all lamps have turned green, the system is fully operational and thus able to detect positions.

After a capture device has been started, its frame grabber automatically gets each frame the camera captures. As mentioned in section 7.1.1, the captured image needs to be rotated and cropped. The reason for this is that the cameras' diagonals capture the desired stripes over the wall's surface. This is also done by the frame grabber since every attached component only uses the transformed images. The first step is to rotate the captured image. After this, the image needs to be cropped to get a stripe which is as high as the specified height in the calibration data. After cropping, the image has objectionable blank spots on the left and right side. The system eliminates this by spreading up the corner pixels of the image on each side. This spreading is done in the first place by extending each side of the original captured frame. Figure 7.8 shows everything, the extension, the rotation and the cropping to get the final image.

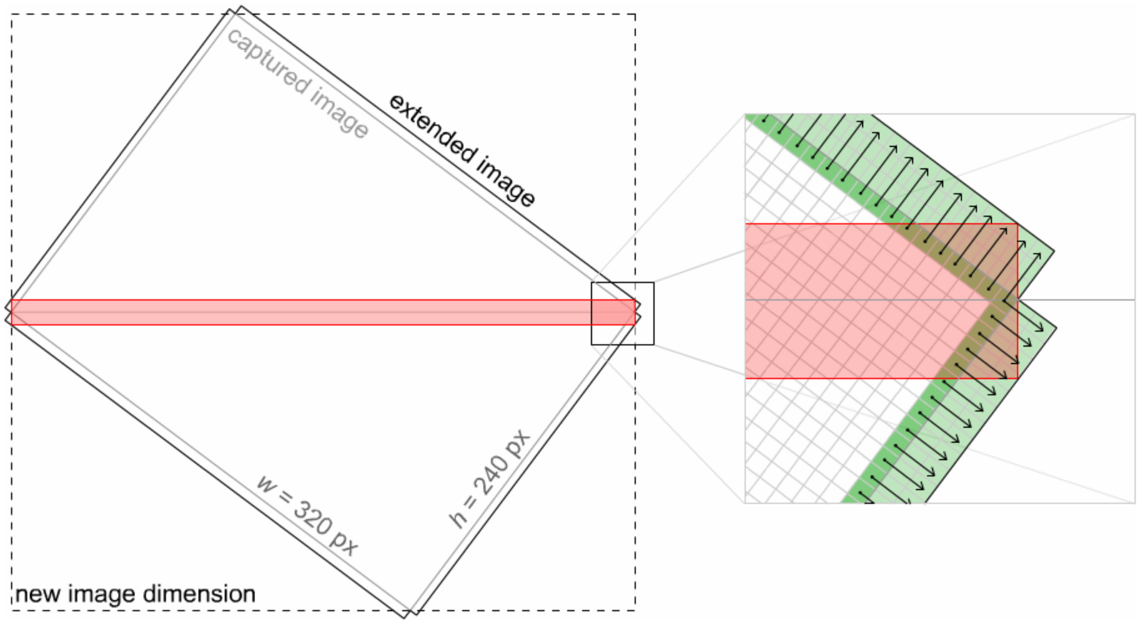


Figure 7.8: This shows the already extended and rotated image. The magnification illustrates how the corners of the resulting image are spread using an extension of each side of the original image. The red bar indicates the portion of the rotated image that will be used for recognition

During runtime, the user gets further feedback using the `DetectionWindow` by selecting it from the application's main menu. This window shows the currently captured frames as well as (if selected) the detected blobs in real-time for every connected camera. Figure 7.9 shows a screenshot of the detection window. Additionally, s/he can use the option of showing the detected lines for each camera which is also done in real-time (see Figure 7.9).

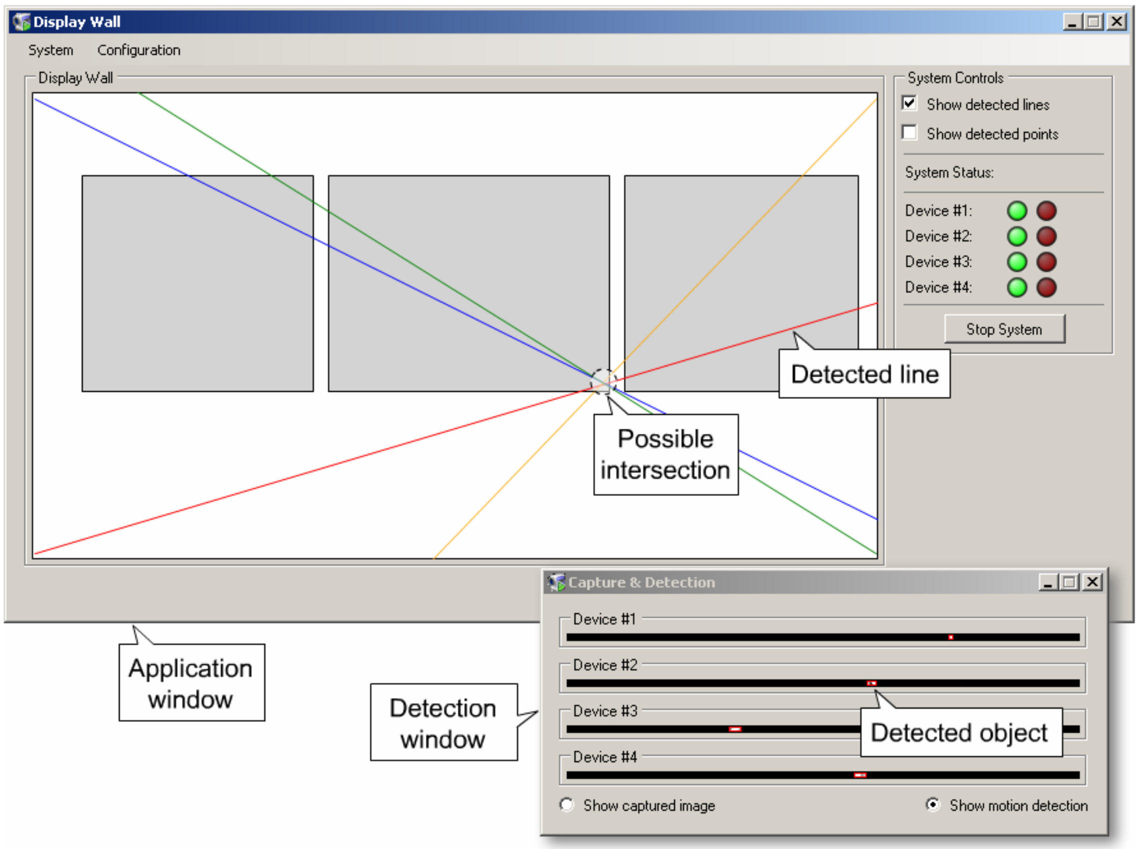


Figure 7.9: Screenshot of the main application window and the detection window. In the detection window, the user is able to see currently detected objects which will be drawn as lines in the main application window.

As soon as the system is in running mode, a capture device uses the last captured frame to detect objects (e.g. fingers) on the observed stripe over the wall's surface. After recognizing the positions in the image using its detector, it immediately gets lines for all corresponding blobs and stores them into an array. This is done by every webcam as soon as a new frame is received by the frame grabber. Another thread – the `Analyzer` – runs separately taking those lines during every cycle. After receiving them, the calculation of intersections can be started.

The first step in calculating the positions is pairing up two cameras to intersect the lines they recognized. Because the system restricts having a maximum number of two fingers on the left and the right side of the wall it is sufficient to use the two left capture devices and the two right capture devices respectively as camera pair. By definition, the system will start with the left camera pair and calculate all possible intersections. This means it intersects all lines from one camera with all lines of the other one. Usually, this gives a higher number of intersections than actually exist on the wall. Thus, the system needs to filter out impossible positions. The first step in accomplishing this is to find out, if the intersection is on the right or on the left side of the wall. Since the left camera is able to observe the complete right side, it can skip all positions found on the left. The next step is to find out which camera of the right pair needs to recognize this point as well. According to the region, the point has been detected in (see Figure 7.10), the system is quickly able to determine this.

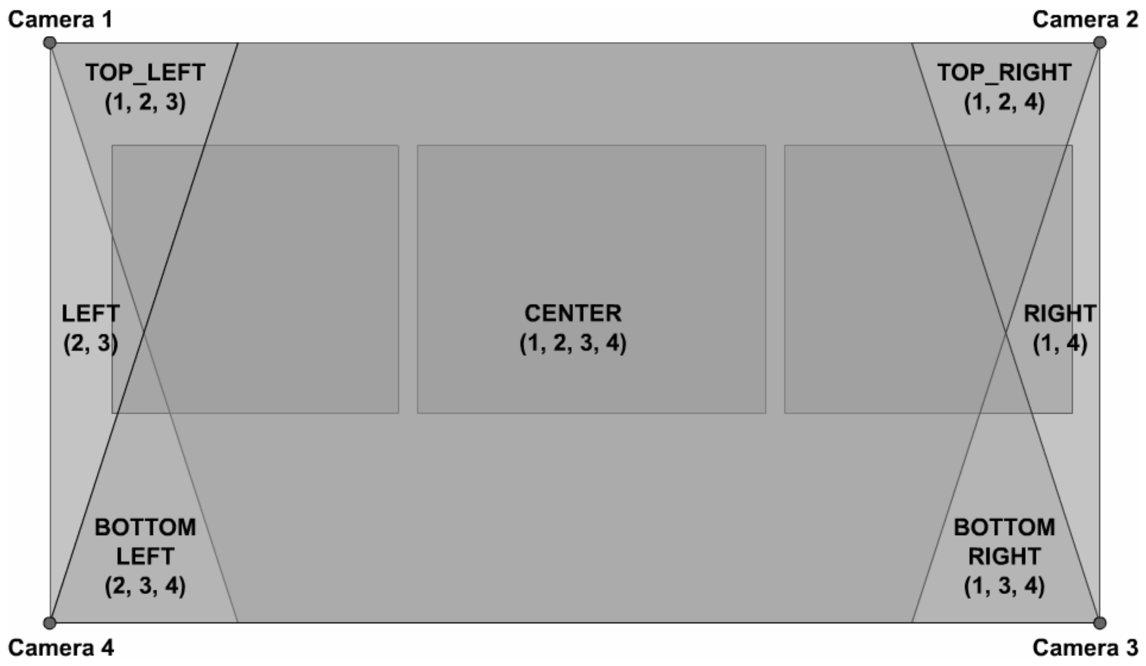


Figure 7.10: Sketch of the different regions and their observing cameras noted in parentheses.

Every calculated intersection will now be tested according to the methods that have been described in section 5.3. Depending on the region, an intersection needs to have two, three or even four lines from different capture devices that will be in short distance of it. After this is done, the system checks the number of intersections found. If still more than two positions exist, it needs to use a third filter step. This includes deleting positions on the far right side since only two cameras can detect these, leading to high inaccuracies. If still more than two points exist, the system simply takes the best matches (e.g. the two intersections with lowest inaccuracy) as final positions on the right side. Subsequently, the right side will be evaluated in analogy. Finally all detected intersections will be merged into one list which will be sent to the input layer running on a different machine. Now, the analyzer is in its beginning state again, and is thus able to calculate the next intersections. Figure 7.11 summarizes the basic steps in calculating positions on the wall's surface.

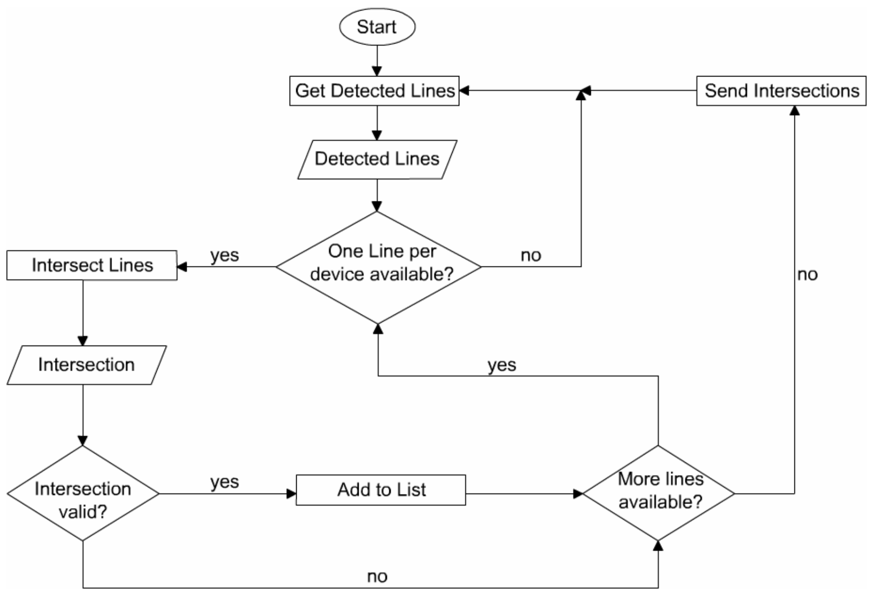


Figure 7.11: Flow chart illustrating the basic steps needed to be done for the calculation of intersections using detected lines from capture devices

All these steps are still done within the basic tracking system, without having other components involved. As I have described how the system calculates positions, I will continue with an explanation of how positions and fingers can be connected using the input layer in the following section.

7.4 Connecting Positions and Fingers

To connect calculated positions to their corresponding fingers on the wall by using a simple motion prediction, as described in section 5.4, the positions need to be sent to the input layer. This abstract layer does not only use the input from the wall's tracking system, but also takes the smartboard's input in the center region into consideration. The basic task is to combine inputs from both components and merge them together to give application developers a unique input structure. Figure 7.12 illustrates the design of the `InputLayer`.

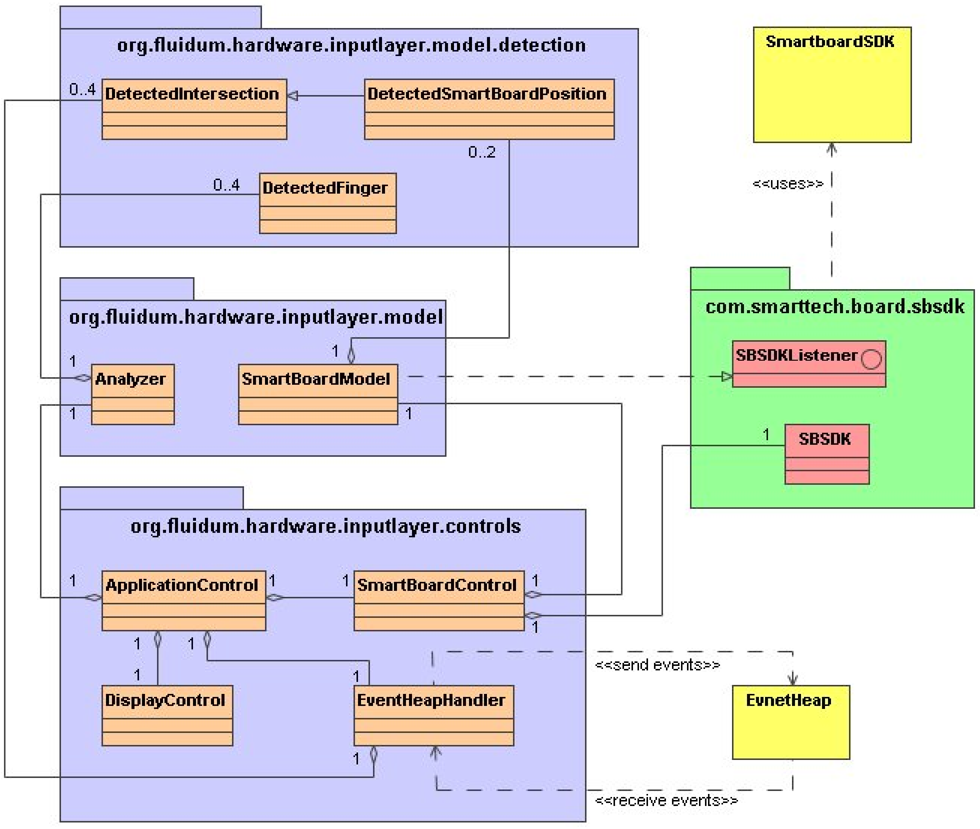


Figure 7.12: Basic design of the input layer component. Packages in green, classes in red and components in yellow are third party components used in the system.

The input layer permanently receives events from the wall's main tracking system through the Event Heap as described in section 7.1.2. Before the positions are sent to the Input Layer, the tracking system needs to convert them into screen coordinates. This is done by using the transformation matrices calculated during the interactive system calibration. As mentioned in section 5.5.2, the system first uses the lead matrix to determine the relative screen position of the detected intersection. In this case, the lead matrix is the transformation matrix calculated for the center screen. If the transformed position is in the left third of the complete screen, the system transforms the detected position again using the transformation matrix for the left display. If it is in the right third of the complete screen, it will use the matrix calibrated for the right display. If none of these two conditions are correct, the previously transformed point (with the transformation matrix of the center screen) will be taken. Figure 7.13 shows a flow chart describing the usage of the transformation matrices. The newly calculated coordinates will be added to the event for each point before it will be sent to the input layer.

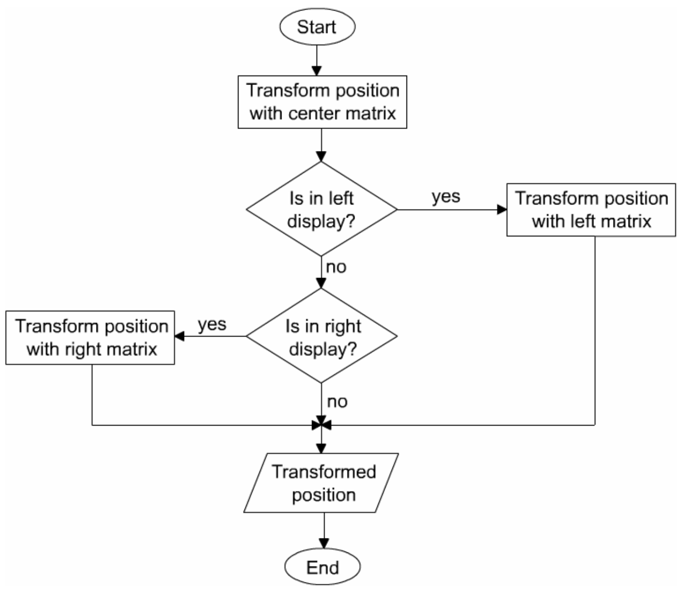

Start
Transform position with center matrix
Is in left display?
yes
Transform position with left matrix
no
Transform position with right matrix
yes
Is in right display?
no
Transformed position
End

Figure 7.13: Flow chart illustrating the transformation of raw coordinates with the correct transformation matrix that has been calibrated previously

After receiving a new position event, each of the contained positions are immediately stored as a `DetectedIntersection` in a vector. Additionally, the input layer obtains positions from the smartboard embedded in the wall as center display. The smartboard is able to track two positions simultaneously on its surface resulting in more points that need to be evaluated. For this reason, a class named `SmartBoardModel` has been created implementing the `SBSDKListener` interface provided by the smartboard's SDK. The listener provides several methods for positioning events. I will briefly describe the most important ones:

- `onXYDown(int x, int y, int z, int iPointerID)`
  This method will be called when the first touch on the smartboard occurs. The parameters `x` and `y` indicate the position (in the smartboard's coordinate system) and `z` gives the pressure (only for non camera-based boards). The `iPointerID` is an identifier whether it is the first or second pointer found on the board. These parameters are the same for every positioning method.

- `onXYMove(int x, int y, int z, int iPointerID)`
  This method will be called when a previously recognized pointer moves on the board without releasing it from the board.

- `onXYUp(int x, int y, int z, int iPointerID)`
  When a pointer has been released from the board this method will be called. The pointer usually caused a "down" and optionally several "move" events before.

Additionally, three further methods for the non-projected mode of the smartboard are given. These methods are named `onXYNonProjectedDown`, `onXYNonProjectedMove` and `onXYNonProjectedDown`. It depends on the use of the smartboard whether the projected methods or the non-projected methods will be called. The projected mode is the projection of a computer display onto the smartboard while manipulating the operating system's mouse pointer directly by touching the screen. In non-projected mode, the smartboard serves as projection plane where the mouse pointer is not manipulated through the generated position events. In this implementation I decided to use the non-projected mode since it is not necessary to affect the system's mouse pointer.

Whenever a position event arrives from the smartboard, the `SmartBoardModel` constructs a `DetectedSmartBoardPosition` and adds it to its queue of detected fingers. Due to restrictions in the smartboard SDK the queue's length is limited to two. The pointer identifier of an event helps the model to associate consecutively detected positions. Additionally, the action of the finger will be set according to the type of event occurred from the smartboard.

Now that I have described how the different input events are gathered by the input layer, I will now explain how the major task – the connection of positions to fingers – is accomplished. In analogy to the tracking system, I created a separate thread called `Analyzer`. This class holds an array of four detected fingers. Each finger contains its current position, the amount of time it has been inactive, its movement vector (see section 5.4), an integer representing its current action and another integer indicating the finger's unique number. The reason for using a value of inactive time is that a finger might not be detected at all during one cycle of the tracking system although it is still on the wall. This happens if the finger moves too fast or is occluded by another one in certain situations. If a finger is inactive for a predefined amount of time, it gets cleared out of the array indicating that the finger has truly left the wall. In the following, I will describe in detail which steps need to be done within one cycle of the analyzing thread.

In the beginning state, the thread updates all previously recognized fingers. This is done by increasing the inactive time indicator by one. If the finger's value equals the predefined value, its action will be automatically set to `FINGER_UP` indicating that the finger might leave the wall in the next cycle. If the value is above the threshold, its position will be set to `null` telling the application that the finger finally has left the wall. After this is done for each finger of the system, the thread collects all detected intersections including the detected intersections from the wall and the detected positions from the smartboard. This list will then be evaluated as described in section 5.4. If the system associates a position to a finger, the finger's movement vector and its position will be updated. Furthermore, the finger's value of inactive time will be set to -1 indicating that this is a newly detected position. Additionally, the action will be set either to `FINGER_DOWN` (finger has been detected for the first time) or `FINGER_MOVE` (finger has been on the wall and is moving).

Finally, the system takes the list of fingers and encapsulates them into an event. This event will be sent to the Event Heap again, where applications running on the display wall are able to receive and process the finger positions to use them as input events. The structure of an event generated by the input layer component is shown in Table 7.2.

Event type	**Fields**	**Class types**	**Time to live [ms]**
NewPositions	FingerID	String	100
	Action	String	
	PositionX	String	
	PositionY	String	
	WallPositionX	String	
	WallPositionY	String	

Table 7.2: Structure of an event sent by the input layer to listening applications

With the knowledge of this event structure, a programmer has the ability to write an application which connects to the Event Heap and registers for these events. Thus, s/he can use the inputs from the wall as well as from the smartboard as input for the application. An example of how these input events can be used will be given in the next chapter.

7.5 The Steerable Projector

The last component of the display wall is the steerable projector mounted on the ceiling in front of the wall. It uses the events generated by the input layer component to change the position of the projected display. The projector can be operated with a DMX[8] controller connected to a computer using a USB to DMX interface. This comes off-the-shelf with a precompiled C++ DLL which takes DMX commands (see Table 7.3) to steer the projector. Thus, I needed to write a JNI interface to connect the projector to a Java application.

Channel	Name	DMX	Function
0	Pan (rough)	0 127 255	Horizontal -170° Horizontal 0° Horizontal +170°
1	Pan (fine)	0 127 255	Horizontal -0.5° Horizontal 0° Horizontal +0.5°
2	Tilt (Rough)	0 127 255	Vertical -135° Vertical 0° Vertical +135°
3	Tilt (Fine)	0 127 255	Vertical -0.5° Vertical 0° Vertical +0.5°
4	Focus	0 127 255	0 Volt +5 Volt +10 Volt
5	Zoom	0 127 255	0 Volt +5 Volt +10 Volt
6	Signal source	0…84 85…169 170…255	Slot 1 select Slot 2 select Slot 3 select
7	Image reversal / rear projection	0…63 64…127 128…191 192…255	off / off off / on on / off on / on
8	Trapeze adjustment up / down	0 1…30 31 32…62 63 64…255	100% Linear change 0% Linear change -100% No change
9	Trapeze adjustment left / right	0 1…13 14	100% Linear change 0%

[8] DMX (Digital Multiplex): Communications protocol used to controls stage lighting [52]

		15…27 28 29…255	Linear change -100% No change
10	Picture mute / shutter	0…127 128…255	Video mute ON Video mute OFF
11	Hyperband channel	0…99 100 101…199 200 201 202 203 204…255	No change Power on No change Power off No change RESET (pan / tilt) RESET (trapeze) No change

Table 7.3: DMX values used to control the steerable projector from an application [34]

Due to time constraints, the projector has not been calibrated in detail. Instead, I measured points on the desired positions on the wall's surface with the tracking system and put them into the projector's control. If the control receives a position event, it interpolates between those points to position itself using pan and tilt values.

For this prototype I only have two positions the projector can display at. Thus, I have predefined pairs of pan, tilt, focus and zoom values. I used the projected images using these values to gather the corners' positions of it to determine whether the recognized finger is inside or outside a projected image. A detailed description of the projector's usage will be given in the next chapter, where I will demonstrate the system's operation in a demo application called *WallDraw*.

Chapter 8

WallDraw: A Demo Application

In this chapter, I will describe a small demo that has been implemented to demonstrate the capabilities of the system. This demo uses all components: The tracking system, the display wall (including input from the smartboard) and the steerable projector (see Figure 8.1). First, I will describe a simple task the user is able to accomplish with the system. After that, I will explain which steps are needed to be done in the system to perform the user's input. I will also show how the system will give visual feedback to the user.

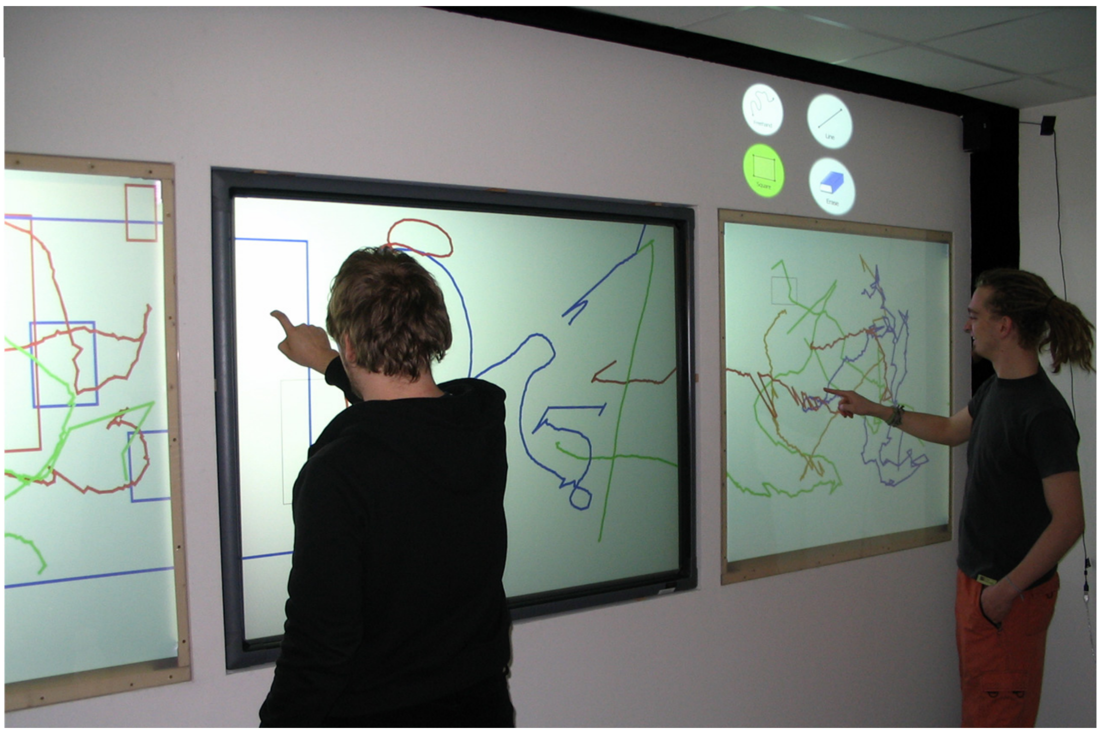


Figure 8.1: Two persons using the *WallDraw* demo application simultaneously

8.1 The User's Perspective

The demo application provides a simple drawing tool to the user. S/he is able to draw on each of the three displays as well as drawing strokes across all of them. There is also a tool palette displayed onto the wall's surface next to the displays using the steerable projector. This allows selecting different tools such as drawing freehand, drawing a line or a rectangle as well as an eraser to delete previous drawings (see Figure 8.2). Different colors for each stroke result from different fingers detected on the wall. Drawing lines and rectangles is done by giving the user a preview function (e.g. a thin black line or rectangle) to let him or her be aware of the shape's final size.

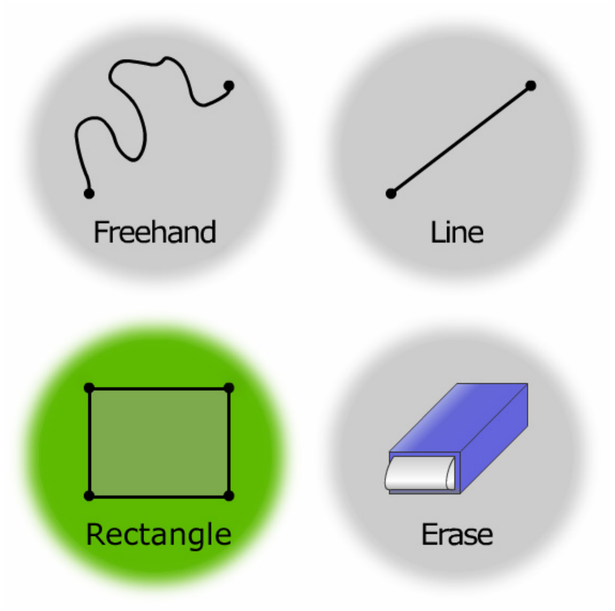


Figure 8.2: Schematic view of the tool palette with the rectangle tool being selected. This palette will be projected onto the wall's surface with black background

I will explain a simple task of the user to demonstrate how the system works. First the user needs to start all components of the system. The order is not important, since all components will communicate over the Event Heap and not directly. An example of starting the system would be to first run the tracking system. After this, the input layer can be started. Finally, the WallDraw application needs to be launched on two machines – one on the display wall and one on the computer controlling the steerable projector. The tool palette is not shown directly after startup but a black image (invisible) will be projected above the center screen.

The user now is able to draw on the wall. To show a more difficult task, I will explain this with having two users using the system. Each of them will start on one side, which means that user *A* will be on the left and user *B* on the right. The task for user *A* is to draw an arbitrary shape across the three displays, while user *B* only needs to draw a rectangle onto the display on the right side of the wall. User *A* first selects the tool palette by "clicking" on the wall's surface above the left display with his or her hand. The tool palette then will be displayed immediately after the touch at the point the user hits the wall. S/he is then able to select the desired tool which is the line tool in this case. This can be done by simply "pushing" the "Line" button displayed on the wall. In the beginning state of the system, the freehand tool is selected for both users and is thus highlighted in green color. Once user *A* has selected the line tool, it gets highlighted to indicate the newly chosen tool. S/he is now able to draw the line with which s/he will immediately start with. In the mean time, user *B* wants to select another tool as well. S/he calls the tool palette in the same way user *A* did by touching the wall above the right display. Once the tool palette displayed at the desired place, user *B* selects the rectangle tool by "clicking" on the displayed "Rectangle" symbol, which automatically gets highlighted. Now s/he is also able to start drawing. Meanwhile, user *A* has already started drawing the line and reached the center screen with the more accurate smartboard, where s/he can continue drawing as if it was the same input device. User *B* has also started to draw the rectangle on the right screen, which will have another color than the line drawn by user *A*. S/he draws a square on the display and releases the finger once the rectangle has the desired dimension. User *A* is still drawing his or her line entering the right display at this time. Once s/he is satisfied with the line drawn, s/he will release his or her finger which finally draws the line on the screen. Figure 8.3 shows important snapshots over time of the user's different steps.

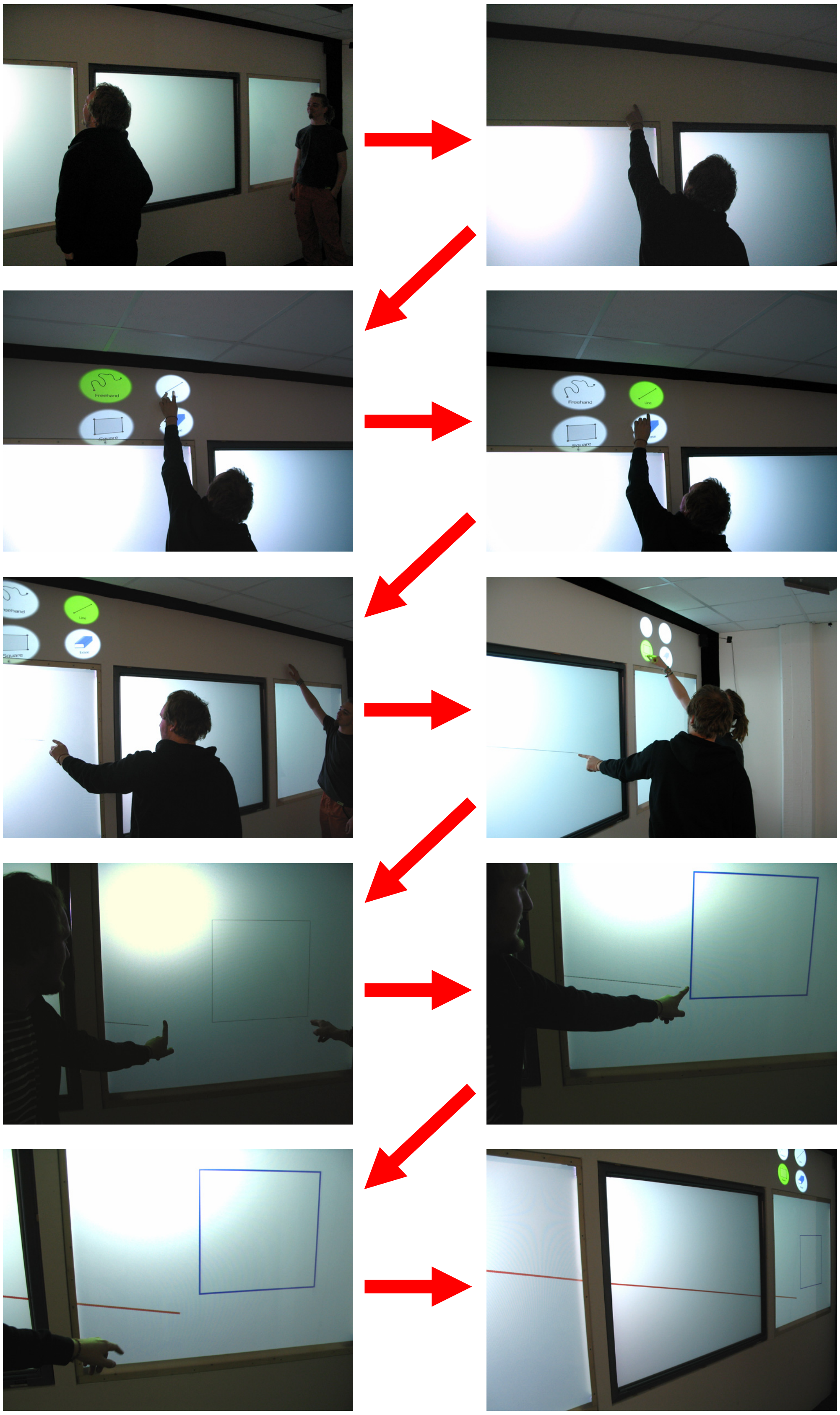


Figure 8.3: Important steps in a dual user scenario using the *WallDraw* demo application with the implemented tracking system and all of its components

Assuming that the system already had been started for the users they will not be aware of the different technologies and machines used. The display wall appears as one large display with a tracking system as input device. Having described the user's perspective of this system, I will continue with the system's perspective to explain what the system needs to accomplish to perform the users' tasks.

8.2 The System's Perspective

In this section, the system's performance to accomplish the users' task will be described. Before I can start with this, I need to explain how the demo application has been embedded in the infrastructure of the tracking system. Figure 8.4 shows the basic design of *WallDraw*.

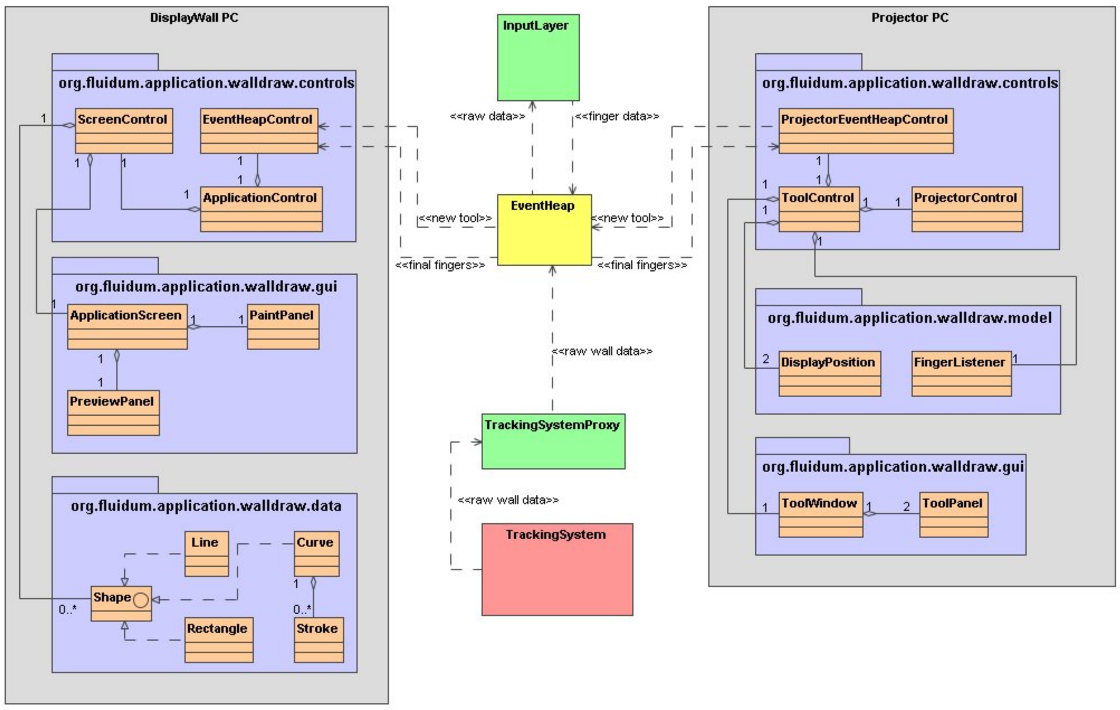


Figure 8.4: Basic design of the *WallDraw* demo showing how it has been embedded in the system.

This demo application needs to be distributed on two machines - one for the wall's displays and one for the steerable projector's control. These parts will also use the Event Heap to send and/or receive events. As shown in this figure, the system needs to use another event indicating a change of a selected tool by the user. Thus, I created a new event called "NewDrawingTool", which is shown in detail in Table 8.1.

Event type	Fields	Class types	Time to live [ms]
NewDrawingTool	ToolID WallSide	String String	100

Table 8.1: Structure of an event sent by the projector control to listening applications (e.g. the display wall)

Assuming that the Event Heap, the tracking system and the input layer are already running, the following steps are done while launching up the demo application. I will start with the part on the display wall. After execution, the application immediately connects to the Event Heap. This also includes a registration for two events the application wants to receive – the

`NewPositions` event from the input layer and the `NewDrawingTool` event sent by the projector's control. Once this is done, the application initializes its drawing surface. This window will use the full-screen exclusive mode and has a white background resulting in a completely white window covering the three displays. At this stage, *WallDraw* is fully functional and able to receive events and process the input commands.

The only component left is the projector's control. This is an optional task since the application does not need to have different tools and will initially use the freehand tool. Once the `ToolControl` has been executed, it connects to the Event Heap letting it know that it also wants to receive the `NewPositions` events from the input layer. After this is done, the application will initialize its tool window which is also running in full-screen exclusive mode. In the beginning state, no tool palette is shown while the projector points on the surface above the center screen. Finally both applications are running and the users are able to make use of them to accomplish their tasks described in the previous section.

As described above, user *A* first wants to have another tool. The tap performed above the left display is recognized by the tracking system and the calculated position is sent to the input layer component. This transforms the position into a detected finger by adding a finger identifier and sends it to all listening applications. In this demo, there are only two of them – the projector's control and the display wall's drawing surface. Both will evaluate the received position and decide whether this is a valid one for them or not. There are three types of possible positions the system needs to distinguish:

- The position is **inside one of the screens** of the display wall, or
- the position is **inside one of the two possible positions** of the tool palette, or
- the position is **somewhere else** on the wall's surface.

Since the received position is not inside one of the screens, the display wall's application will not perform any further steps. At the same time, the projector's control will evaluate the received position. As mentioned in section 7.5 the corner points of the tool palette have been entered directly into the projector's control. Figure 8.5 shows the determination whether a point is inside the tool palette's position or not.

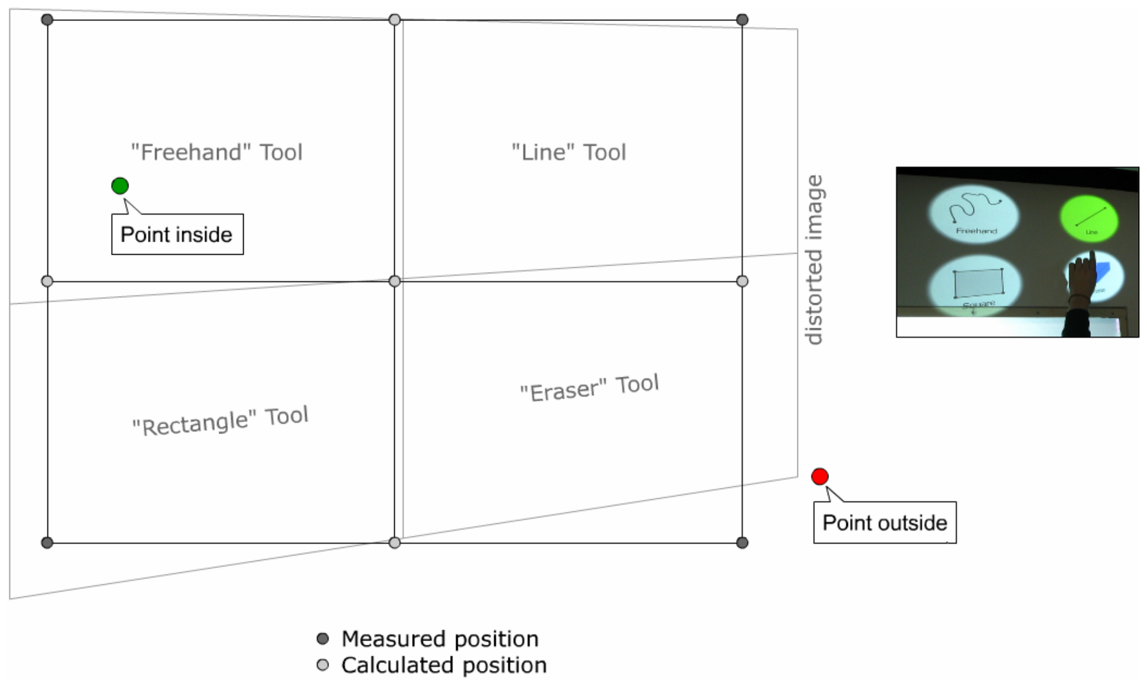


Figure 8.5: Distorted image (above left display) and the corresponding measured positions on the wall.

This evaluation needs to be done for both, the left and the right tool palette's position. Additionally, the application needs to check whether the tool palette is already being displayed at this side or not. If so, the projector does not need to be moved and the system needs to figure out which of the four tools have been touched by the user's finger. If the projector needs to be moved, it will also exchange the visible tool palette since it has a unique tool palette for every side of the wall. As mentioned before, the tool panel is not visible in the beginning and it is "projected" above the center screen. Thus, the projector moves and displays the left side's panel above the left display immediately after recognizing a touch on the wall's surface.

After moving to the left position, user *A* touches the wall again inside the line tool icon this time. The process in determining the action is the same as described above. Since the projector is above the left display, it knows that the user selected the line tool by evaluating the newly received event from the input layer. Immediately, the tool palette changes the active tool to the selected one by highlighting it with the green color. Additionally the old selected tool will be deactivated. Internally, a `NewDrawingTool` event will be sent to the wall's drawing surface application to let it know that the user has selected a new tool. The drawing application will then know that it has to start drawing a line, when the user touches the wall inside the displays (on the left side).

User *A* now touches the wall inside the left display to start drawing the line. The system recognizes the new position and again sends it to the two applications. This time the position is not of interest for the projector's control and thus will be ignored. The drawing application also receives the `NewPositions` event. This includes the action `FINGER_DOWN`, indicating the start of a drawing function. The application now creates a new `Line` object using this position as starting point. As the user starts moving the finger, new events will arrive from the input layer including the action `FINGER_MOVE`. They also contain a new position and the same finger identifier as the first event, letting the application know that the new position belongs to the started line. After every event, the preview will be updated to show the user how the line will look like if s/he releases the finger.

Although user *A* is currently drawing the line with his or her finger on the wall's surface, user *B* is still able to select the tool palette. Internally, two events will be sent from the input layer to the two processes. Because both will have unique finger identifiers, the system will not get confused. The line of user *A* can still be drawn using positions with the identifier used at the start point. The second position event in the beginning is outside the displays again starting the procedure for selecting a new tool as described above.

Now user *B* also has selected his or her desired tool (the rectangle tool) s/he is also able to start drawing. With the `NewDrawingTool` event sent by the projector's control, the drawing application now knows that every shape started on the right side of the wall will be a rectangle. User *B* now touches the wall inside the right display to start drawing the rectangle. As noted before, the system can clearly distinguish between both recognized positions on the wall by comparing the unique finger identifiers. Thus, the system knows whether a received position belongs to the line of user *A* or the rectangle of user *B*.

Even when user *A* enters the smartboard region, resulting in a high accuracy of tracking, the system will not end the started line and start a new one. Since it still receives the input from the input layer (which also controls the smartboard), it is not aware of the change of the tracking system. Also, the user does not notice that s/he is tracked by another component. The same happens if the user gets on one of the side displays.

User *A* has now finished drawing the line on the right display. Thus, s/he releases the finger which results in a `NewPositions` event with the action `FINGER_UP`. Now the system knows that the user has stopped drawing his or her shape and immediately draws it in the finger's associated color. The same will happen as soon as user *B* releases his or her finger. Figure 8.6 will summarize the users' interaction with the system, including the tasks the application needs to perform.

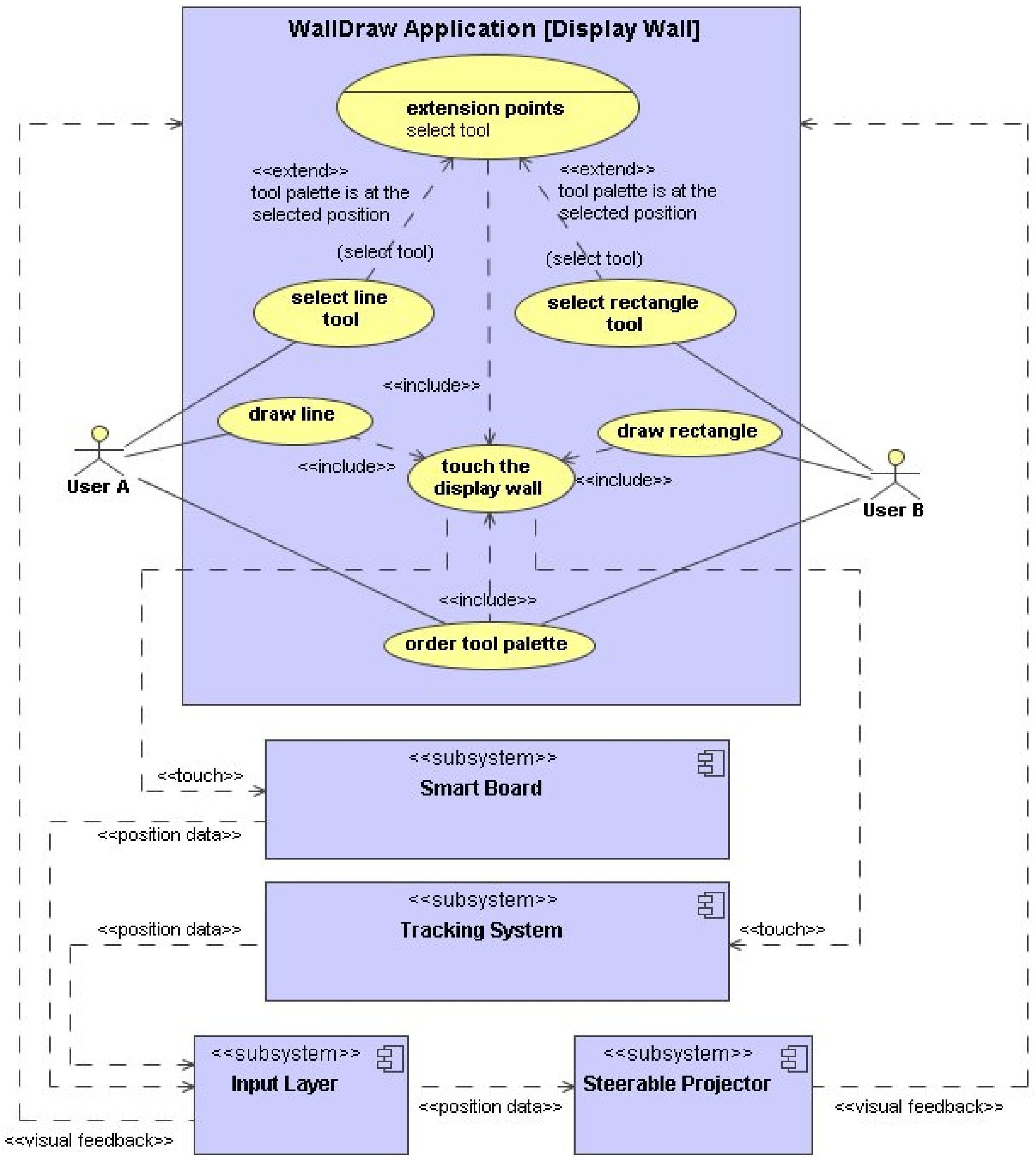


Figure 8.6: Shows a use-case diagram illustrating the interaction a user can have with the system and what steps are done within the application if one of the users touches the wall

Now that I have described a first demo application using the components currently available in the system, I have shown that the wall can serve as one large display in the user's perspective. Unfortunately, there are several problems with having multiple positions on the wall. Those will be discussed and evaluated in the next chapter.

Chapter 9

Conclusions

The implementation of this system was more difficult than I first thought. Handling four capture devices to build an application running nearly in real-time is a complicated task. Additionally, there were several hardware and software problems I encountered during the phase of implementation. In this chapter I will give a summary of the problems I had building this system. I will start with the time management of the project continued by a description of technical lessons I have learned. After that, I will illustrate the performance the system currently has. Additionally, I will show possibilities to fix some of those.

9.1 Time Flow of the Project

One of the most important problems was the time management. Managing a project as large as this one by oneself is rather difficult. This was a totally new experience for me. Additionally, the technical problems encountered during the work were discouraging from time to time. I ran into several dead ends during the implementation phase, which might have been avoided using a more theoretical approach to the problem.

Phase	Planned	Realized
Study of related work	One month	Three weeks
Planning the system	Three weeks	Two weeks
Hardware setup	Two weeks	Five weeks
Implementation (System & demo)	One month	One and a half months
Writing	Four weeks	Four weeks

Table 9.1: Summary of the time planned and actually used to build the system

The planning phase was accomplished in a short amount of time, which led to the mentioned implementation problems. Rearranging time resources might have solved this issue, but since I have never realized such a project, I did not know this in the beginning. During the implementation phase I was often frustrated due to unsolvable technical issues. It was also important to deal with these problems and to figure out solutions quickly to avoid being stuck. At some points, this problem led to a high amount of work.

Another part of the social lessons I have learned has been my own autonomy. While studying at a university still gives a high amount of supervision, I had to create ideas on my own to solve the problem stated in the beginning. However, as soon as I came up with a new idea, I discussed this with other people in the project which helped modify or discard it.

Finally, Table 9.1 gives an insight of how I distributed the time before the project. It also shows how I tried to keep those milestones during the thesis. In the end I have to say that it was a great time as well as a great experience for me.

9.2 Technical Lessons

During this work, I learned a lot regarding technical issues as well as social impacts. I had to deal with many new situations and problems resulting from hardware and software issues. Additionally, I had to manage the planning and implementation in a short period of time which has probably been the most difficult task throughout the whole project. Nevertheless, I was impressed with the results I achieved with relatively inexpensive hardware.

Immediately after starting my work on this system, I encountered several technical issues. This included dealing with multiple capture devices, which is not possible with Java applications. Thus, I had to learn a new programming language (C#) as well as the DirectX framework. I enjoyed learning this because it was a challenge and a chance to see how concepts from one programming language can be transferred or modified into another. Additionally, it was the first time I had to manage with multiple capture devices. Since I like to deal with video data, this was an exciting task to accomplish during this thesis.

There were many more difficulties regarding the technical aspect. First, I had to adopt several input techniques such as the smartboard. Furthermore, I needed to create a concept for the steerable projector and how it might be used in a demo application. This also included connecting this new product to my own system using additional technologies such as the DMX to USB interface.

Other lessons included the construction of the fixation for the cameras. The first approach was the design in 3D Studio Max. This design was sent to an industrial manufacturer who was able to print the fixations three-dimensionally. Unfortunately, those printouts had one tenth of the desired size and were thus not applicable for the system. After this disappointment and several costs, I decided to produce those fixations with simple wooden boards, resulting in sufficient products. Additionally, this method was far more inexpensive, compared to the three-dimensional printouts.

Furthermore, the evaluations and decisions regarding the used hardware were very time-consuming. This particularly included the choice of the cameras and their extension cables to connect them to a single computer. In the beginning I had to deal with erroneous cables as they were designed for USB 2.0 but only transferred data with USB 1.1. Finally, those cables did not work at all at some point and thus needed to be replaced by others.

All in all, this work has been a big challenge for me and other people involved in the technical point of view. It turned out to be extremely interesting because of all the new and advanced technologies that have been assembled together in this system.

9.3 Performance and Possible Improvements of the System

In this section I will describe the performance the system has after implementation. It also includes solutions for some of the failures, which have not been implemented due to time constraints but are applicable after completing this work.

One major problem has been the amount of jitter produced by the tracking system. This results from the sequential capturing of frames, which means that the simultaneously evaluated four frames have not been captured at exactly the same time. Figure 9.1 exemplifies the jitter occurring from the system using the *WallDraw* application.

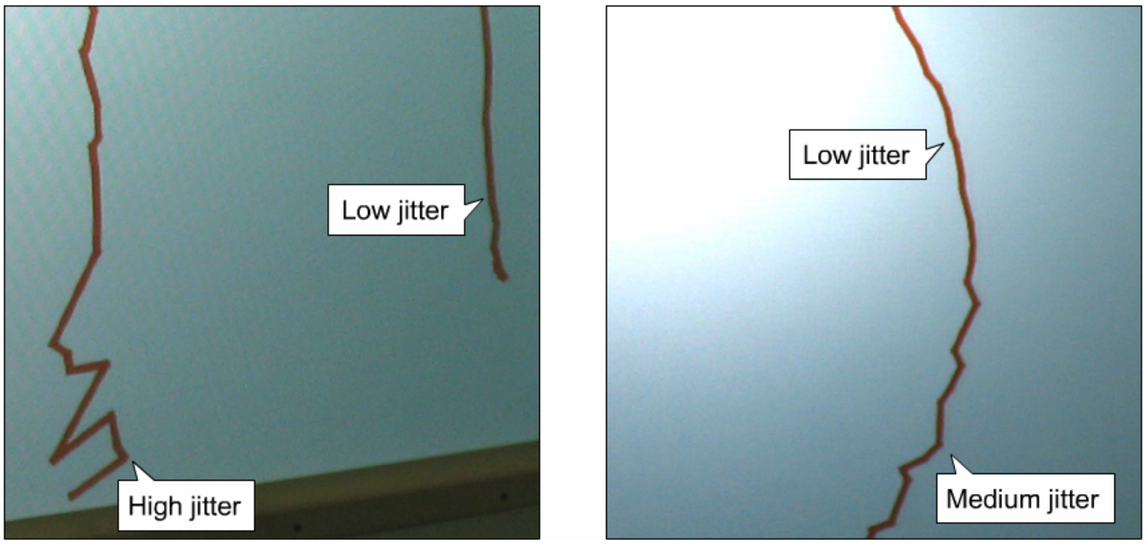


Figure 9.1: Jitter occurring while running the *WallDraw* demo. Left shows two line ends with high jitter for one. Right shows a circle's segment increasing its jitter at the bottom.

As mentioned above, the reason for getting this jitter is the subsequent capturing of frames by the cameras and the application. With this knowledge, it is possible to reduce the amount of jitter by capturing multiple frames with their corresponding time stamps. Whenever the `Analyzer` evaluates a set of frames, it will first create a virtual time stamp which lies in between two captured frames of each camera. The system then interpolates between a pair of frames to get the estimated position of the recognized object at the given virtual time stamp. This should give positions as close as possible to the correct ones assuming that the cameras do not need different periods of time transferring the data to the computer. Figure 9.2 shows an example of the jitter occurrence before and after the possible modification.

Another problem is, that the cameras are not able to cover the whole wall due to their angle of aperture. This results in having regions where only two or three cameras are able to observe objects. As mentioned in section 5.3, having one object in a region with two cameras does not affect the process of triangulation. Unfortunately, the case that two fingers are in such an area might occur, which results in only one finger being tracked. This problem can be solved using either more cameras or devices with a higher angle of aperture (at least 90 degrees). Both methods would be sufficient, but using more than four capture devices results in a higher amount of processing power, and is thus not the appropriate solution.

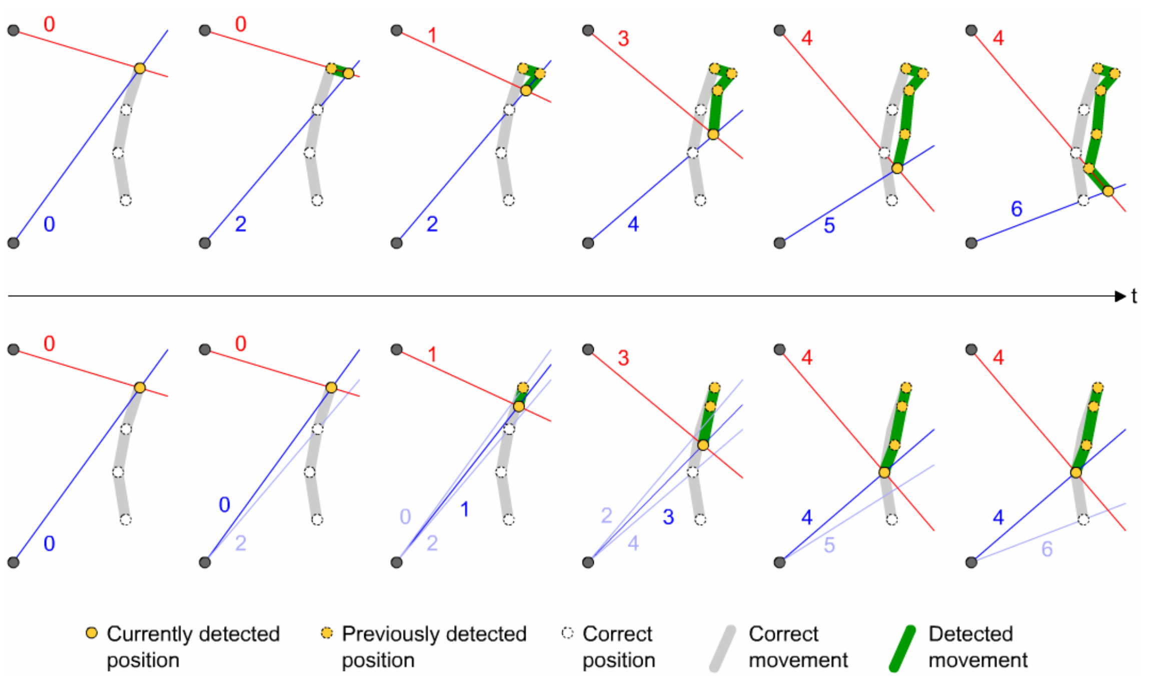


Figure 9.2: Example of a detection process. Top shows the currently used method whereas bottom is showing the interpolated method. The numbers represent the time stamps of each line.

Furthermore, the system suffers from using visual tracking, as it is not able to track occluded fingers. This also results in detection errors as shown in section 5.3. There are lots of cases where occluding is a big problem. Most of them are only solvable using more capture devices in a specific setup on the wall. As mentioned above, this would result in a higher latency of the system which is not applicable at all as it has to give immediate feedback to the user. Even industrial solutions such as the smartboard have these occlusion problems.

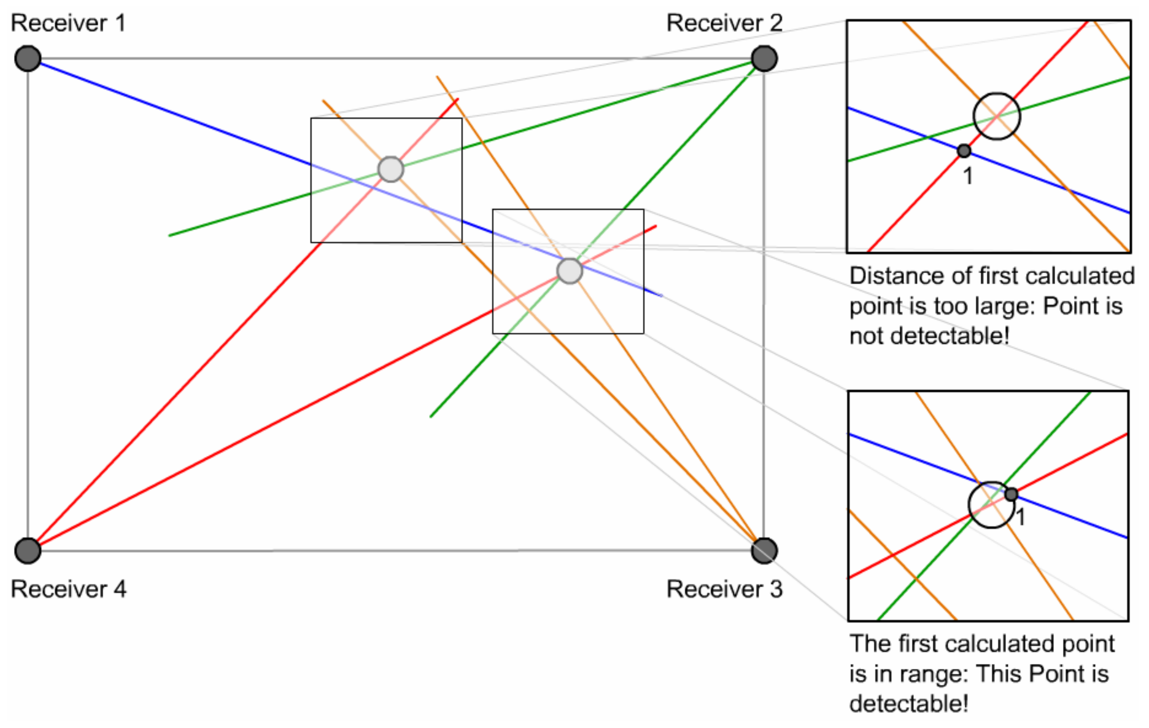

Receiver 1
Receiver 2
Receiver 4
Receiver 3
1
Distance of first calculated point is too large: Point is not detectable!
1
The first calculated point is in range: This Point is detectable!

Figure 9.3: This shows two positions on the wall where only one will be detected. The upper magnification shows the detection error due to a too large distance of the calculated position from other lines. The lower one shows the correct detection of the second point on the wall.

The next error the system produces is, if two fingers get too close to each other. This is mainly caused by the resolution of the cameras. Each camera captures an image which has a width of 400 pixels due to performance reasons. Capturing larger images slows down the tracking system a lot and is thus not applicable. Whenever the distance in the captured image of two fingers is small, the cameras will detect only one line. This has multiple impacts on the tracking system. For example, the system might not detect a position on the wall due to measurement errors. Figure 9.3 shows one example of the described problem.

Besides leaving a valid intersection undetected, the system can also merge multiple intersections into one intersection. This leads to inaccuracies of the detected position as well as a position not being detected at all (see Figure 9.4). This case is basically a special case of the first issue. If the system detects both intersections, there are still problems if they are close too each other. Those include inaccuracies for both positions as well as complications in determining which position belongs to which finger (see Figure 9.5). In some cases this may lead to wrong associations of intersections to fingers, leading to points that hop between different positions. This problem is not of interest as long as the application is designed for clicking operations. As soon as drag and drop needs to be used, it might frustrate the user because of some non-comprehensible actions the system executes. This is only solvable for some cases by using cameras with a much higher resolution but it will still fail for some situations.

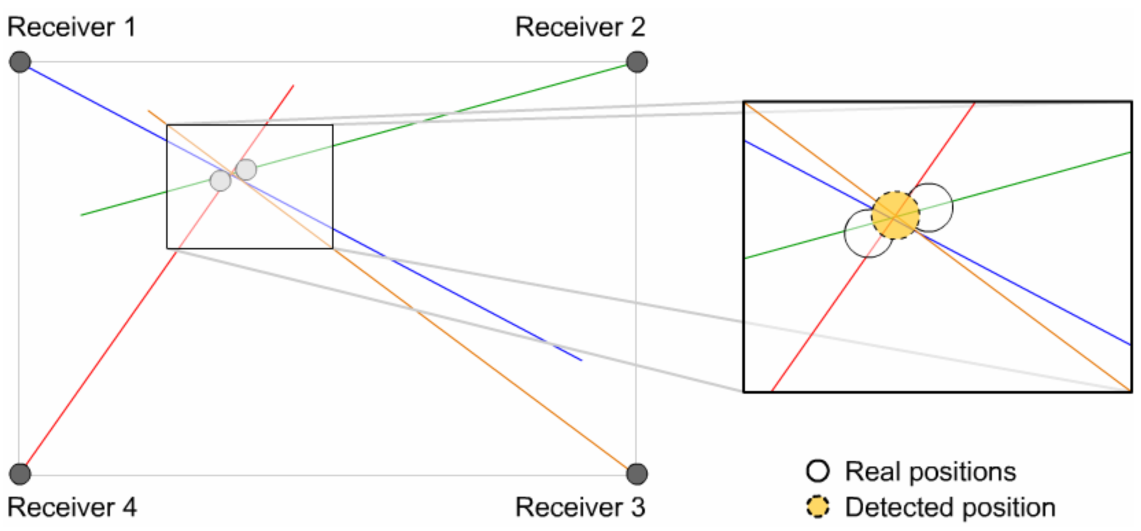


Figure 9.4: This shows the inaccuracy as two intersections existing on the wall are merged into one detected intersection, causing jitter over a given time span.

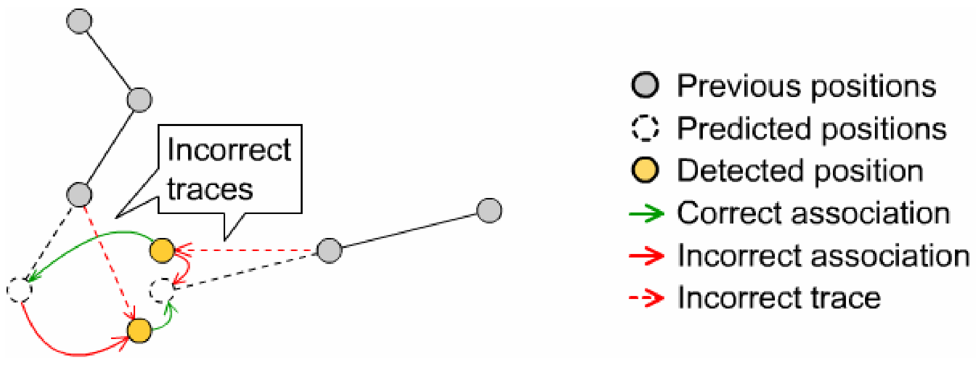


Figure 9.5: This shows an incorrect association of detected intersections to points if they are close to each other. The dashed red lines show that the positions will be associated to the wrong finger resulting in incorrect movement detected on the wall.

Regarding the performance aspect of execution speed, the system is acceptable as it takes about 100 milliseconds to calculate the positions and the association to fingers. Also, the system is robust and will only fail in the case a camera gets detached from the system. This would be solvable by listening to the USB attach and detach events produced by the operating system. If a detach event occurs, the tracking system could, for example, stop and give a warning to the user that s/he needs to reconnect the device.

Overall, this system shows that it is possible to implement an inexpensive wall-sized focus and context display, with a tracking system providing the user with the ability to interact with it. This also involves having a multi-user bimodal interaction space which might be extended with further displays in the instrumented room. While it is fast enough for interactions that require clicking, it still needs to get faster for moving and/or dragging operations as shown in the *WallDraw* demo application. To solve this, the system could be equipped with more cameras or with cameras that have a higher resolution. If this is to be done, the processing power of the computer needs to be increased to ensure a fast detection.

Additionally, with this system, it has been shown that it is possible to combine more input and output technologies to have an interactive display across the complete wall. This display is a single one to the user, who does not need to know what components and input and output technologies are involved, while s/he interacts with it. In the next chapter, I will give an overview of the steps that will be done to improve the system.

Chapter 10

Future Work

In this chapter I will discuss parts of the system that have been designed in theory and practice but have not been implemented yet. This also includes several components which were not part of the basic system described in this work. I will start with the improvements regarding the tracking system. Subsequently, I will discuss ideas to improve the image equalization of projected images being distorted by the steerable projector. Finally, the demo application *Brainstorm* will be introduced as it is a first demo using the instrumented room.

10.1 Improvement of the Tracking System

One of the most important parts that need to be done is the improvement of the tracking system, which includes several steps. Besides the avoidance of jitter as shown in section 9.3, the system needs to get faster and more accurate. This affects the main tracking system running locally on the machine with the attached cameras.

Additionally, it is desirable to avoid using black velvet on the wall. This may be accomplished by using other image analysis techniques such as improved HSL filtering, edge detection and/or region growing. For this task, it may be necessary to change the imaging library from AForge [48] to OpenCV [17]. Besides using other computer vision techniques, the cameras' settings for captured images can be adjusted in different ways using different values for saturation, brightness and contrast.

Furthermore, the input layer needs to be updated. The most important part of this will be the association of fingers. This includes using another motion prediction as the currently implemented one does not perform well enough for this system. As mentioned in section 5.4, the application of a Kalman filter is highly desired, because it will give an accurate motion prediction, which avoids wrong associations in most cases.

Another point I want to update is the limitation of four fingers being detected by the system. This needs to be changed to at least six positions, so there can be a person in front of each screen. This results in three persons and – if they are all working with both hands – in six detectable positions.

Finally, the system needs to match real-world coordinates as the room will be extended to a three-dimensional information space where information can be displayed not just on the large display wall. In the next section I will discuss this more detailed, as I will describe image equalization techniques for the steerable projector.

10.2 Image Equalization for the Steerable Projector

One major problem shown in the *WallDraw* demo has been the distortion of the projected image (e.g. the tool palette in the demo's case). This basically has been described and done in [44] and may be improved in the future for real-time applications.

The basic idea is, that the room will be given as virtual space within the system including the projectors position. This enables the projector to display a non-distorted image everywhere in the room on every wall with the usage of the method shown in Figure 10.1.

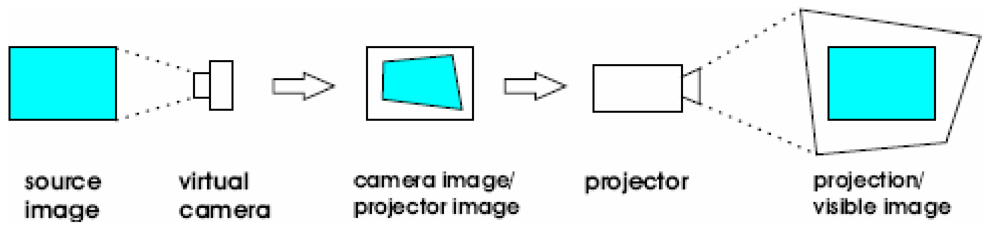


Figure 10.1: Mutual obliteration of projection distortion and camera distortion [44]

This solution demonstrates a virtual camera mounted in the virtual room at the projectors position and with the projector's orientation. It captures the projected image on the wall. If this captured image is projected using the same orientation and distance again with the projector in the real world, the image will be undistorted. Another possibility is to calculate transformation matrices for each pixel. These need to be evaluated first especially regarding the execution speed of the calculation and transformation.

With the possibility to display everywhere, the complete system needs to be extended with a display component. This component should also handle the equalization described above. Furthermore, it should be possible to add new displays into the environment such as interactive tables, PDAs and cell phones, which means that the component needs a simple calibration and tracking procedure to retrieve the displays' initial and current positions. Visual markers might be one solution for this problem, using the steerable projector as calibration device.

All improvements regarding the steerable projector are of high interest to FLUIDUM and me personally. For this reason, parts of my research will focus on this component in the instrumented room to design and implement the steps described above.

10.3 Brainstorm: Demo of the Instrumented Room

The project FLUIDUM started to design a demo application using the display wall as well as an interactive table in the instrumented room. The goal of this application is to support the collaborative creation of mind maps. In this section, I will discuss the demo's status at this point of time regarding the design.

First of all, I will identify the components being used within the demo application. Besides the display wall (including its tracking system as well as the smartboard), an interactive table, which is being produced at the moment, will be used in *Brainstorm*. This table is an LCD TV equipped with a DViT overlay as used with the wall's smartboard. Thus, it is an interactive screen supporting two fingers simultaneously.

The demo is divided into three general phases. The first phase is the creation of notes (e.g. Post-Its) on the interactive table. The reason for accomplishing this assignment on the table is that users can sit comfortably around it. For this task, the users can touch an empty space on the surface which results in the creation of a new note. In there, they can write down their ideas like it is done on Post-its in the physical world. Every created note has one raised corner indicating that the note may be flipped. Once it has been reversed, the user is able to edit, to copy or to delete the note. After the creation of basic ideas has been done, the notes may be moved to the wall either by wiping them or by putting them into a proxy for the wall.

The second phase is the generation of groups and clusters on the display wall. This is the first step where the display wall resulting from this work will be used. The reason for grouping and clustering them on the wall is that it has a much larger screen size and resolution. Post-its can be moved on the wall to put them into virtual groups. Once they have been grouped spatially, the users can draw a line around the post-its using their fingers on the wall. This indicates a cluster. These clusters can be merged by dragging one cluster onto another one. Two further methods regarding clusters have been designed. These are dissolving one by ripping apart the surrounding line of the cluster and deleting one by crossing it out. Once the clusters are created, they can be moved the same way as the notes to arrange them spatially.

The third phase is the consolidation into a mind map. Thus, the users need to decide on a root for it. Once finished, they can draw lines to logically connect the clusters into the desired mind map. The only step left is the ordering inside the clusters on the wall. This will be done on the smartboard for one cluster, whereas all other (inactive) clusters will be displayed on the two side displays as contextual information. Whenever the users want to activate another cluster, they are able to drag the desired one onto the center screen.

This demo is currently being implemented by the members of the FLUIDUM project. As shown, there is no thought for using the steerable projector in this demo, but this might be changed in the future.

List of Figures

List of Tables

Bibliography

[1] AUTODESC INC., *Autodesk – Autodesk 3ds Max – Product Information*, `http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112`

[2] AZUMA, R., *A Survey of Augmented Reality*, In: Teleoperators and Virtual Environments 6, 4, pp. 355-385, 1997

[3] BAUDISCH, P., GOOD, N., STEWART, P., *Focus Plus Context Screens: Combining Display Technology With Visualization Techniques*, In *Proceedings of UIST '01*, Orlando, FL, USA, November 2001, ACM, pp. 31-40

[4] BAUDISCH, P., GOOD, N., BELLOTTI, V., SCHRAEDLEY, P., *Keeping Things in Context: A Comparative Evaluation of Focus Plus Context Screens, Overviews, and Zooming*, In *Proceedings of CHI 2002*, Minneapolis, MN, USA, April 2002, ACM, pp. 259-266

[5] BEDERSON, B., HOLLAN, J., *Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics*, In *Proceedings of UIST '94*, Marina del Rey, CA, USA, November 1994, ACM, pp. 17-26

[6] BORCHERS, J., RINGEL, M., TYLER, J., FOX, A., *Stanford Interactive Workspaces: A Framework for Physical and Graphical User Interface Prototyping*, In *IEEE Wireless Communications, special issue on Smart Homes*, 2002, pp. 64-69

[7] BRÜGGE, B., DUTOIT, A., *Object-Oriented Software Engineering. Conquering Complex and Changing Systems*, Prentice Hall, Upper Saddle River, NJ, USA, 2000

[8] BUTZ, A., SCHNEIDER, M., SPASSOVA, M., *SearchLight – A Lightweight Search Function for Pervasive Environments*, In *Proceedings of Pervasive 2004*, Vienna, Austria, April 2004, Springer LNCS, pp. 351-356

[9] CREATIVE TECHNOLOGY LTD., *Creative WebCam Live! Ultra – Do more with USB 2.0, Smart Face Tracking, and CCD Sensor for exciting instant messaging*, `http://www.creative.com/products/product.asp?category=218&subcategory=219&product=10451`

[10] DAVIES, J., *An Ambient Computing System*, Master's Thesis, Department of Electrical Engineering and Computer Science at the University of Kansas, Kansas, United States of America, 1998

[11] DECATRON, *DECATRON, Magic Vision*, `http://www.decatron.ch/produktDetail.asp?Rubrik=men&URubrikID=118&ProduktID=258`

[12] DIETZ, P., LEIGH, D., *DiaomndTouch: A Multi-User Touch Technology*, In *Proceedings of UIST '01*, Orlando, FL, USA, November 2001, ACM, pp. 219-226

[13] FLUIDUM, *Flexible User Interfaces for Distributed Ubiquitous Machinery*, `http://www.fluidum.org`

[14] GARLAN, D., SIEWIOREK, D., MAILAGIC, A., STEENKISTE, P., *Project Aura: Toward Distraction-Free Pervasive Computing*, In *IEEE Pervasive Computing 1*, 2002, pp. 22-31

[15] GEORGIA INSTITUTE OF TECHNOLOGY, *[AHRI] – Aware Home Research Initiative*, `http://www.awarehome.gatech.edu/`

[16] HAN, J., *Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection*, In *Proceedings of UIST 2005*, Seattle, WA, USA, October 2005, ACM, pp. 105-108

[17] INTEL COOPERATION, *Open Source Computer Vision Library*, `http://www.intel.com/technology/computing/opencv/index.htm`

[18] JOHANSON, B., FOX, A., *The Event Heap: A Coordination Infrastructure For Interactive Workspaces*, In *Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02)*, Callicoon, NY, USA, June 2002, IEEE Computer Society, pp. 83-93

[19] JOHANSON, B., FOX, A., WINOGRAD, T., *The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms*, In *IEEE Pervasive Computing Magazine 1*, 2002, pp. 71-78

[20] KIDD, C., ORR, R., ABOWD, G., ATKESON, C., ESSA, I., MACINTYRE, B., MYNATT, E., STARNER, T. NEWSTETTER, W., *The Aware Home: A Living Laboratory for Ubiquitous Computing Research*, In *Proceedings of CoBuild '99*, Pittsburgh, VA, USA, October 1999, Springer LNCS, pp. 191-198

[21] KIENTZ, J., BORING, S., ABOWD, G., HAYES, G., *Abaris: Evaluating Automated Capture Applied to Structured Autism Interventions*, In *Proceedings of UbiComp 2005*, Tokyo, Japan, September 2005, Springer LNCS, pp.323-339

[22] KÜPPER, A., *Location-based Services: Fundamentals and Operation*, John Wiley & Sons Ltd., 1, Weinheim, 2002

[23] LEUNG, Y., APPERLEY M., *A Review and Taxonomy of Distortion-Oriented Presentation Techniques*, In *ACM Transactions on Computer-Human Interaction*, 1994, pp. 126-160

[24] LOGITECH EUROPE S.A., *Logitech – Leading web camera, wireless keyboard and mouse maker*, `http://www.logitech.com/index.cfm/DE/EN`

[25] LOGITECH EUROPE S.A., *Logitech Products > Webcams & Video Services > Webcams > Logitech® QuickCam® Fusion*^{TM}, `http://www.logitech.com/index.cfm?page=products/details&CRID=2204&CONTENTID=10562`

[26] LOGITECH EUROPE S.A., *Logitech QuickCam® Pro 4000*, `http://www.logitech.com/index.cfm?page=products/details&CRID=4&CONTENTID=5042`

[27] MATSUSHITA, N., REKIMOTO, J., *HoloWall: Designing a Finger, Hand, Body, and Object Sensitive Wall*, In *Proceedings of UIST '97*, Banff, Alberta, Canada, October 1997, ACM, pp. 209-210

[28] LFE MEDIENINFORMATIK, *Medieninformatik LMU*, `http://www.mimuc.de`

[29] MICROSOFT COOPERATION, *.NET Framework Developer Center*, `http://msdn.microsoft.com/netframework/`

[30] MICROSOFT COOPERATION, *DirectX Developer Center*, `http://msdn.microsoft.com/directx/`

[31] MITSUBISHI ELECTRIC RESEARCH LABORATORIES, *MERL - Mitsubishi Electric Research Laboratories*, `http://www.merl.com/`

[32] PINHANEZ, C., *Using a Steerable Projector and a Camera to Transform Surfaces into Interactive Displays*, In *Proceedings of CHI '01*, Seattle, WA, USA, March 2001, ACM, pp. 369-370

[33] PRANTE, T., STREITZ, N., TANDLER, P., *Roomware: Computers Disappear and Interaction Evolves*, In *IEEE Computer*, 2004, pp. 47-54

[34] PUBLITEC PRÄSENTATIONSSYSTEME & EVENTSERVICE GMBH, *publitec*, `http://www.publi.de/`

[35] REKIMOTO, J., *Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments*, In *Proceedings of UIST '97*, Banff, Alberta, Canada, October 1997, ACM, pp. 31-39

[36] REKIMOTO, J., SAITOH, M., *Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments*, In *Proceedings of CHI '99*, Pittsburgh, PA, USA, May 1999, ACM, pp. 378-385

[37] RUDDARRAJU, R., HARO, A., ESSA, I., *Fast Multiple Camera Head Pose Tracking*, In *Proceedings of Vision Interface 2003*, Halifax, Canada, June 2003

[38] RUDDARRAJU, R., HARO, A., NAGEL, K., TRAN, Q., ESSA, I., ABOWD, G., MYNATT, E., *Perceptual User Interfaces using Vision-based Eye Tracking*, In *Proceedings of ICMI 2003*, Vancouver, Canada, November 2003, ACM, pp. 227-233

[39] SANYO COOPERATION, *SANYO > Portable Projectors*, `http://www.sanyo.com/business/projectors/portable/index.cfm?productID=391`

[40] SHOPPING.COM, *Siemens Scenic X102 (FAPSCED10202GB) PC Desktop*, `http://img.shopping.com/cctool/PrdImg/images/pr/177X150/00/01/60/8c/79/23104633.JPG`

[41] SMART TECHNOLOGIES INC., *SMART Technologies Inc., industry leader in interactive whiteboard technology, the SMART Board*, `http://www.smarttech.com/`

[42] SMART TECHNOLOGIES INC., *DViT – Digital Vision Touch Technology (White Paper)*, `http://www.smarttech.com/dvit/DViT_white_paper.pdf`

[43] SOUNDLIGHT, *SOUNDLIGHT PC DMX Technik*, `http://www.soundlight.de/produkte/wg18/usbdmx/usbdmx.htm`

[44] SPASSOVA, L., *Fluid Beam – A Steerable Projector and Camera Unit*, Student and Newbie Colloquium at ISWC / ISMAR, Arlington, VA, USA, November 2004

[45] STANFORD UNIVERSITY, *Stanford Interactive Workspaces Project Overview*, `http://iwork.stanford.edu/`

[46] SUN MICROSYSTEMS INC., *Java 2 Platform, Standard Edition (Java SE)*, `http://java.sun.com/j2se/`

[47] TERRENGHI, L., *Design of Affordances for Direct Manipulation of Digital Information in Ubiquitous Computing Scenarios*, In *Proceedings of Smartgraphics 2005*, Frauenwoerth Cloister, near Munich, Germany, August 2005, Springer LNCS, pp. 198-206

[48] THE CODE PROJECT, *Image Processing Lab in C# - The Code Project*, `http://www.codeproject.com/cs/media/Image_Processing_Lab.asp`

[49] THE CODE PROJECT, *Motion Detection Algorithms - The Code Project*, `http://www.codeproject.com/cs/media/Motion_Detection.asp`

[50] THE UNIVERSITY OF QUEENSLAND, Australia, *Information Environment Equipments Listing and Support*, `http://www.itee.uq.edu.au/~mmds2802/EZIOTips/smart.html`

[51] UNIBRAIN INC., *Unibrain Fire-i digital camera*, `http://www.unibrain.com/Products/VisionImg/Fire_i_DC.htm`

[52] UNITED STATES INSITUTE FOR THEATRE TECHNOLOGY INC., *USITT DMX512*, `http://www.usitt.org/standards/DMX512.html`

[53] VISION ACADEMY DEUTSCHLAND, *Lexikon der Bildverarbeitung – Reflexionsgrad*, `http://www.atm-marketing.de/mv_wbuch/np/8709C3A21267B103C1256D57002EE4C7.htm`

[54] WEISER, M., *The Computer for the 21st Century*, In *Scientific American*, 1991, pp. 94-104

[55] WEISER, M., *Hot Topics: Ubiquitous Computing*, In *IEEE Computer*, 1993, pp. 71-72

[56] WELCH, G., BISHOP, G., *An Introduction to the Kalman Filter*, Technical Report 95-041, Department of Computer Science, University of Chapel Hill, NC, USA, 1995

[57] WU, M., BALAKRISHNAN, R., *Multi-Finger and Whole Hand Gestural Interaction Techniques for Multi-User Tabletop Displays*, In *Proceedings of UIST '03*, Vancouver, Canada, November 2003, ACM, pp. 193-202