Anywhere Touchscreen with Head-mounted Displays

Virtuelle Touchscreens mit Head-Mounted Displays
Bachelor-Thesis von Anton Moritz Rohr aus Wiesbaden
Tag der Einreichung: 2. März 2015

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Telecooperation Group

Anywhere Touchscreen with Head-mounted Displays
Virtuelle Touchscreens mit Head-Mounted Displays

Vorgelegte Bachelor-Thesis von Anton Moritz Rohr aus Wiesbaden

Prüfer: Prof. Dr. rer. nat. Eberhard Max Mühlhäuser
Verantwortlicher Mitarbeiter: Florian Müller

Tag der Einreichung: 2. März 2015

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 2. März 2015

(Anton Moritz Rohr)

Abstract

Augmented Reality has lately become a popular topic among more or less futuristic concepts in computer science. 3D capable head-mounted displays like Microsofts *HoloLens* and Metas *Spaceglasses* are in active development. But there are still unsolved problems such as the interaction with content. No fully satisfying solution exists, even if there are already many approaches. Although 3D has also become very popular over the last few years, real 3D content is still rare. Most of the digital information today is in text, pictures and videos. All of these are 2D and meant to be displayed on a 2D screen.

In this work a solution for combining head-mounted displays with the still most used digital interface, the screen, is presented. The introduced system offers a new way of interaction that enables direct manipulation, through the use of touch input on a virtual screen. The software tries to provide a natural feeling to the user, by using a flat surface in the near environment. The virtual screen is perspectively transformed and anchored in the real world, hence it seems to be a conventional screen pinned to a surface.

To realize this project a time-of-flight camera and augmented reality glasses are used. Together they form a head-mounted device, which scans the user's field of view and is able to augment it with additional information. The video stream is used by several computer vision algorithms to track the user's head movement, detect fingers, touches and planar surfaces. For content generation a *WebKit* browser engine is used, allowing the display of types of *HTML*-based data and media.

Zusammenfassung

Augmented Reality ist in der letzten Zeit zu einem sehr populären Thema bei mehr oder weniger futuristischen Konzepten der Informatik geworden. 3D-fähige *Head-Mounted Displays* wie Microsofts *HoloLens* und Metas *Spaceglasses* sind in der aktiven Entwicklung. Aber es gibt immer noch ungelöste Probleme, wie zum Beispiel die Interaktion mit Inhalten. Obwohl 3D die letzten Jahre auch immer populärer wurde, gibt es noch immer wenig echte 3D Inhalte. Aktuell liegen die meisten digitalen Informationen in Texten, Bildern und Videos vor; das alles sind 2D Inhalte und dafür gemacht auf einem Bildschirm angezeigt, zu werden.

In dieser Arbeit wird eine Lösung präsentiert, die *Augmented Reality* mit der immer noch am meisten genutzten digitalen Schnittstelle, dem Bildschirm, verbindet. Das vorgestellte System bietet eine neue Art der Interaktion, die eine direkte Manipulation ermöglicht, durch den Einsatz von Touch-Eingabe auf einem virtuellen Bildschirm. Die Anwendung versucht, dem Nutzer ein möglichst gewohntes Gefühl zu bieten, indem es flache Oberflächen in der näheren Umgebung nutzt. Der virtuelle Bildschirm ist perspektivisch verformt und verankert in der echten Welt, sodass es scheint, er sei ein herkömmlicher Bildschirm, der an einer Fläche befestigt wurde. Zusätzlich dazu kann dieser Bildschirm auch als *Touchscreen* benutzt werden und ermöglicht somit Interaktion mit den angezeigten Inhalten.

Um dieses Projekt zu realisieren, wird eine *Time-of-flight* Kamera und eine *Augmented Reality* Brille benutzt. Zusammen bilden sie einen Apparat, der auf dem Kopf getragen wird, das Sichtfeld des Trägers filmt und es mit zusätzlichen digitalen Inhalten erweitern kann. Der Video-Datenstrom wird von mehreren *Computer Vision* Algorithmen verarbeitet, um des Nutzers Kopfbewegungen und die Position von Fingern, Berührungen und Flächen zu erfassen. Um Inhalte zu generieren, wird ein *WebKit* Browser verwendet, der die Möglichkeit bietet, jegliche *HTML*-basierten Daten und Medien darzustellen.

Contents

List of Figures

1 Introduction

1.1 Motivation

In the last years the topic of augmented reality has become more and more popular. As the name suggests, the concept is about enriching the real world with additional digital information. There are several ways to achieve this, but this work focuses on head-mounted displays with binocular transparent screens, that are capable of presenting three dimensional content, like the *Vuzix Star 1200*. The increasing importance of augmented reality is also shown by the emergence of several other devices, that are either in active development or already available. For example there is the Epsons *Moverio Smart Glasses*, Microsofts *HoloLens*, Googles *Glass* and Metas *SpaceGlasses*.

There is a wide range of applications for this form of augmented reality in our daily life, from navigation systems with virtual arrows on the street, to not using conventional screens anymore at all. But there are also unsolved problems such as the interaction with the presented content. No fully satisfying solution exists, even if there are already many approaches. Some approaches use additional devices, other are body-based, using voice, gesture or on-body touch. But all of them either need complex additional hardware or are not capable to provide direct manipulation.
Most of todays digital content is two dimensional and lies in text, images and videos. This raises several questions: How should these contents be presented in the three dimensional world? Where should it be placed in the field of view, so that it feels natural to the user and does not cover important parts of the real world?

Finding appropriate solutions to these problems is not trivial, but the technology of range measuring cameras offers new options. The main motivation behind this work is to propose a solution, by providing a system to use a planar surface as virtual touch screen, that solely uses a head-mounted display (HMD) and a *time-of-flight* camera.

1.2 Research Questions and Contribution

As outlined above, the main question is how to create a virtual touch screen, that is capable of direct manipulation in an augmented reality scenario. This can be split into following detailed questions.

- How to integrate virtual displays seamlessly into the environment, so that it feels natural for the user?
- How to interact with the displayed content?
- Where to present digital information in the user's field of view?

This work proposes a solution to the previously stated questions in form of an prototype application. With the appropriate hardware, it offers a way to place a virtual touch screen on a surface in the user's close environment.
The touch interaction and the possibility of responsive content, offers the option to manipulate and perceive live feedback. Additionally it is very accustomed to the user.
The screen is perspectively transformed, meaning that if the user is directly in front of the surface and the viewing angle is orthogonal, the screen is rectangular. But if position or angle changes, the screens shape is transformed in such a way that it appears natural to the user. In addition the screen is anchored in the real world, meaning if the user turns his or her head, the screen stays at the same position. This provides a perception as if a conventional screen were pinned to a surface, therefore providing a familiar experience and a seamless transition between the real world and the digital content.

As a prototype application this software should show what is possible in principle. It can also be a basis for future augmented reality projects, for example to try out different touch-based keyboard types.

1.3 Structure

This work is structured as follows: After this short introductory section 1, section 2 discusses somehow similar projects, with a focus on the main differences and the novelties, which are introduced by this project. The following section 3 is all about relevent knowledge that was used, mainly from the areas of augmented reality and computer vision. Section 4 explains the system architecture of the developed software, in an top to bottom direction. The subsections contain thoughts on which other possibilities there were, and why a certain way was chosen. In section 5 important technical and implementation details are explained and how an user handles the software. Section 6 critically analyses the developed system, lists parts that do not work well yet, states alternatives and possible extensions. At the end, the section 7 concludes the work, reflects on some decisions and dares an outlook on the future of augmented reality.

2 Related Work

2.1 Overview

In this section some related work is discussed and novelties of the present approach are shown. At first the famous augmented reality experiment SixthSense of Pranav Mistry is explained and some of its limitations are connected to the motivation of this work. In the next subsection, several other content interaction approaches in scenarios with head-mounted displays are discussed and explained why an additional approach is needed. In the last subsection several approaches to computer vision based finger and touch detection are discussed without the focus of head-mounted displays.

2.2 Sixth Sense

Pranav Mistry and Pattie Maes built an augmented reality device in 2009 which shares some features with the present setup and work [19, 20]. The device called *SixthSense* gained a lot of attention, because of a corresponding TED talk[1], which is one of the most watched to this date.

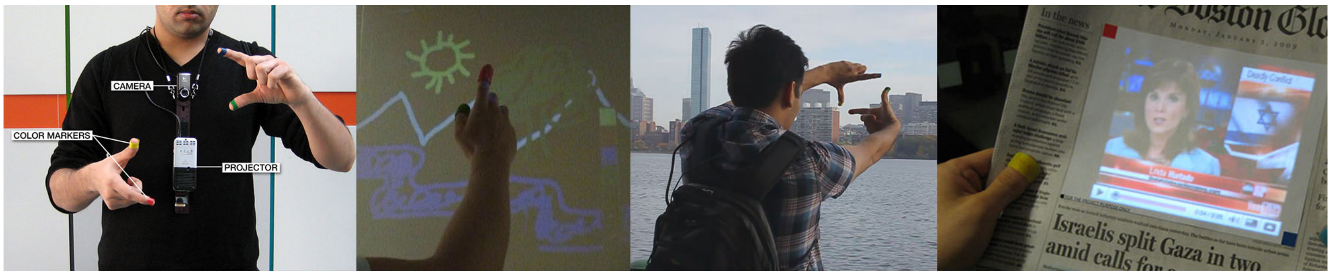


Figure 1: SixthSense (source:[19])

2.2.1 Setup

The *SixthSense* is a wearable device worn on your chest and attached to a strap around your neck. As you can see in Figure 1 the main hardware parts are a camera and a small projector. The camera and the projector face nearly in the same direction, orthogonal to the chest. This position was chosen because the projector's content can then be seen clearly and the camera sees the users hands if the arms are stretched out. In addition a mobile computing device is needed, which can be in the user's pocket or e.g. in a backpack. To recognize the user's fingers as input devices, color marker are used.

2.2.2 Handling

The user uses his hands to give some kind of input to the device. Several different actions are possible. For example, in a drawing mode, you can draw with the index finger and the projector displays the drawing in real-time. Also gestures can be recognized by the system and it is possible to shoot a photo by forming a frame with your fingers. If you face an appropriate surface, you can scroll through your photos with flick gestures. The last scenario from Figure 1 shows a newspaper with an augmented digital video in it. Mistry's work explains that displaying videos on paper is also possible, but does not state any details.

2.2.3 Comparison

The comparison between our hardware setup and the *SixthSense* is the following. Instead of a projector, glasses as head-mounted display device are used and the camera is in addition capable of distance measurements. The main idea behind both projects are very similar as both aim to augment the real world with digital content.
SixthSense has one main limitation which leads directly to the main motivation of the present work. The user is confined to use a free wall to use the projector of the device. The position of the projector does only allow the use of a vertical surface and the setup is meant to be used in a standing position. If the projector is not orthogonal to the surface, the

[1] http://www.ted.com/talks/pranav_mistry_the_thrilling_potential_of_sixthsense_technology

displayed screen is deformed, which leads to an unpleasant experience. In addition the user has to keep his or her upper body in the same position, otherwise the projected image starts to move as well.
These limitations raised the question of how to properly display content, which led directly to one motivation of this work. The present work proposes a solution that is capable of using all kinds of flat surfaces. Furthermore it is usable even if the user moves and it generates a pleasant and natural experience.

2.3 Content Interaction with Head-mounted Displays

The following subsection discusses some related work from the perspective of content interaction with head-mounted displays. Several interaction approaches are discussed, from either commercial products or research. The focus lies on limitations of these in relation to the solution of this work. The last part explains in more detail, why and how the proposed system is a novelty.

2.3.1 Voice- and Device-based Interaction

The device Googles Glass[2]attracted some attention in the media recently and is an example for voice and device interaction. Several voice inputs like "OK Glass, take a photo" are available as well as a touchable area on the side of the device. Epsons Moverio BT 200[3] even has an external touchable device which is connected with a cable. Both approaches are quite limited, as voice input is restricted on a set of commands that can be understood. Direct manipulation is not possible, neither with voice input nor through such a touch device. Furthermore, a second touch input device always means carrying additional hardware.

2.3.2 Mid-Air Gesture-based Interaction

The SixthSense Device [19], which is in the next section discussed in more detail, uses marker on finger tips to recognize gestures like flicking and pinching. FingARTips[5] works also with markers trough a glove and addresses the direct manipulation problem. Both are limited by the number of implemented gestures and do not work without the mentioned marker. The difference of these two, to the proposed solution from this work, is mainly the setup without markers.

2.3.3 Touch On-Body Interaction

Some approaches use the palm or forearm as touch area, which has the main advantage that they are always available. Skinput [11] makes use of a pico projector instead of an HMD and senses sound waves that travel through the skin after a touch. It needs a special band with sensors, which is wrapped around the upper arm.
Another project from the same research team, OmniTouch [10], uses also a pico projector, but this time a Kinect depth camera is mounted on the shoulder.
Despite both are capable of direct manipulation, the main disadvantage of both setups is the inflexibility. The arm has to be bent and held in a special position to be in the projecting and detecting area.

Imaginary Phone [8] and PalmRC [7] are both systems working with depth cameras and the palm as a touchable surface. Both do not use do not use a visual interface, but rely on the human sense of proprioception, that allows finger movements without any visual attendance. Both are stationary setups, that are limited to the area the cameras scan.

2.3.4 Novelties

The proposed solution of this work offers a way to interact with augmented content through an unprecedented manner. The novelty is a combination of touch interaction on flat surfaces with virtual screens. This is solely done trough a single RGB-D camera and without the need of additional hardware. The solution offers a way to perform direct manipulation and can be used everywhere where surfaces are found. It is mobile with the meaning that no stationary hardware is needed. The system would even be able to perform multi-touch if an appropriate 2D gesture recognition is used.

2.4 Touch Detection

This subsection focuses on related work about touch detection in all kinds of scenarios independently from head-mounted displays. There have been several other attempts of developing touch detection through hand and finger detection.

[2] `http://www.google.com/glass/start/`
[3] `www.epson.com/MoverioBT200`

Wilson [26] uses cameras behind a semi transparent surface. A touch on the surface causes a reflection that is detected and tracked with the camera.
Another approach from Niikura [21] tried recognizing touches with the help of a wristband that includes a camera and a microphone. The camera was mounted on the inner side of the wrist to film the space between hand and a surface.
A possibility for touch input in 3D space was also developed by Niikura [22]. He designed a small device with a camera and lights that is meant to be placed below the bottom end of a smart phone, facing the camera upwards. Input can then be generated by placing the finger above the camera and performing movements. The finger detection results generate feedback for the user in form of a cursor on the smart phone screen. A movement towards the screen increases the area of the finger in the perceived image and this is used to detect a click or touch event.
These three approaches try to detect touches with the help of camera and therefore computer vision. The main difference to the principle of the present work is the position of the camera. A more similar setup was used by both Letessier [14] and Agarwal [1]. They try to solve the finger detection problem with cameras mounted above the scene. Letessier uses a conventional color camera and works with background extraction and shape fitting. His approach only supports finger detection and tracking but no touch detection. Agarwal uses a two camera setup to create stereo vision and addresses the touch detection problem with his work. His algorithms use machine learning techniques and he achieves very detailed results. His system is capable of detecting a touch event with an accuracy of up to 90%, even with a minimal finger to surface distance of three millimeter.
The novelty and difference of the approach described in the present paper is the use of a mobile and moving head-mounted camera. On one hand a moving camera introduces new difficulties, for example it is not easily possible to do a background subtraction. On the other hand it offers the new possibility to use the system everywhere and not just in a prepared stationary setup. Furthermore the use of a depth camera introduces new options for algorithms, leading to the extensive use of distance differences.

3 Background

3.1 Overview

This section is about relevant knowledge mainly in the areas of augmented reality and computer vision. Furthermore the capabilities, functionality and limitations of RGB-D cameras and augmented reality glasses are described.

3.2 Virtual and Augmented Reality

Virtual Reality (VR) is the concept of a simulated world or environment, in which an user can immerse himself. As an interface to this world, special VR technology is used, which stimulates different senses. Most of them focus on sight and hearing, but among others haptic perception is also imaginable. While immersed in a virtual world, the user does not perceive the real world with the blocked senses. A recently developed example for such technology is the Oculus Rift[4].
Augmented Reality (AR) is similar to Virtual Reality (VR) but with one main difference: AR technology does not block the perception of the real world, but supplements it with additional virtual elements. [2].

In the following, there are two out of many other approaches explained that show how to display altered reality. The first one is sometimes called *window-on-the-world* and uses a conventional display and a camera. The camera captures the real world and an application alters the image and then shows it on the display. Because most of current smartphones fulfill the requirements, today several augmented reality apps are available. Often additional input data, like the compass or GPS position, is used to derive information about the smartphone user's current situation. An example app is SkyView[5], which adds the constellations to the night sky in real time, based on GPS position and camera direction.
The second approach is similar to the first one but needs additional hardware. Transparent displays offer the possibility to place digital content directly into the user's field of view. This is most of the time realized by integrating these displays in eyeglasses or the visor of a helmet. These systems are called *Head-mounted Displays* (HMD). Like in the smartphone scenario additional data sources like sensors are often used. A camera is not necessary but often used, to apply computer vision. [18]

3.3 3D World Model

It is important to model the surrounding real world in an appropriate way. First of all, it must be done in a way the computer can understand. On one hand it should not be too complex to avoid unnecessary memory consumption and computational effort, especially if software has to run in real time. On the other hand, to get the most precise results it is reasonable to use all input data. Here are two world modeling methods explained, which both directly derive from given input data.

3.3.1 Pointcloud

The first one of these modeling methods is often called *Pointcloud* and is a representation of measured depth data. A *Pointcloud* is in general a two dimensional array, with a three dimensional vector as every entry. The array represents the grid of points where the camera measures the depth. One dimension is represented by the columns and the other dimension by the rows of the grid. The vector at every grid point represents a real point in the world. The three values of the vector are the distance in three directions (X,Y,Z) from the origin, which is most of the time the camera sensor. [24]

3.3.2 Depth Image

The second model is called *Depth Image* and is a simplified version of *Pointcloud*. A *Depth Image* is also a two dimensional array, but with only one value for every entry. Like before, the array represents the pixels at which the camera senses and has therefore the same width (columns) and height (rows) as the *Pointcloud*. Also the value at each pixel is the same as the Z component of a *Pointcloud*. The naming depth in *Depth Image* comes from calling the distance between the real world object and the camera *depth*. And it is considered an image, as it has much in common with normal images, besides the pixel values are not color but distance values. In fact a depth image can be displayed like a normal image, if the distance values are treated as greyscale color values.

[4] `https://www.oculus.com/rift/`
[5] `https://itunes.apple.com/en/app/skyview-free-explore-universe/id413936865?mt=8`

Even when this representation lacks some accuracy in relation to the *Poincloud* it has one big advantage: The possibility to treat a *Depth Image* like a normal image offers the opportunity to use all conventional computer vision algorithms.

3.4 Real-time Computing

As mentioned in subsection 1.2 providing a live experience is one of the main goals of the current project. To achieve this the application has to work smoothly and may not stutter. The eyes in combination with the brain interface is able to see moving motion above a rate of about 24 pictures per second. The decision was taken to adopt the frame rate of the camera 3.6.1 which is 30 frames per second (FPS). This leads to the following calculation:

$$T = \frac{1}{f} \Rightarrow \frac{1}{30Hz} = 33.\bar{3}ms$$

This simple calculation shows that computation time for each frame must not exceed 33 ms. There are several ways to shorten computation time. The most important is to use an efficient programming language and well developed algorithms. Another way to make efficient use of the given resources is multi threading, especially with current multi core processors.

3.5 Computer Vision

3.5.1 Overview

Computer Vision (CV) is a field of computer science which tries to gather additional information from image data. Very different kinds of information can be used with this approach, depending on the particular problems addressed. Detecting and tracking of objects, movements or surfaces as well as enhancing and filtering are some of them.
Besides augmented reality CV has also many other fields of application, for example in robotics. For some robots it is crucial to understand their environment, especially if they are meant to operate in unpredictable changing environments like a household. Other examples for daily used computer vision would be bar and QR code scanning.
Most of the CV concepts use normal color images as input, because depth sensing cameras have become available only during the last few years. But as stated in subsection 3.3.2 these algorithms can also deal with 3D data, if an appropriate representation like a *Depth Image* is used.

In the following section several concepts and algorithms are presented and explained. The next subsection, explains what filtering an image means and uses the median filter as example. Several concepts around feature tracking and optical flow are shown leading to two algorithms for the newly developed software.

3.5.2 Filtering

The process of filtering an image tries either for enhancing the quality in some kind of way or to highlight a special property of the image. One example to enhance an image is noise reduction. Noise reduction aims to smooth image data by removing artifacts which often occur because of non perfect camera sensors. For classic color sensors scenes with insufficient light are hard to capture. Therefore the image brightness is digitally amplified, causing noise in the data. For the present work, noise reduction was particularly important, because, as mentioned in subsection 3.6.1, the depth sensor in the *time-of-flight* camera measures not very accurately, resulting in an unsteady, noisy image.

Median Filter

Median filtering is a common filtering technique in computer vision, but besides image processing, it is also used in other kinds of signal processing. Applying a median filter reduces statistical outliers and therefore smoothes the image and reduces noise. In comparison to other smoothing filters, like for example a Gaussian blur, the main advantage is that median filtering also preserves edges. This is helpful in all scenarios where further computations are based on these edges, like the algorithms described in subsection 3.5.3 and subsection 3.5.5. The main disadvantage of median filtering is that it uses sorting and is therefore computationally expensive, meaning that it is in general more time consuming.
A median filter is defined as follows: let $[I_{xy}]$ be a matrix which describes an image and every matrix entry describes a pixel value. Median filtering with a window of size $n \times m$ results in an image matrix $[J_{xy}]$. Every pixel value j_{xy} of $[J_{xy}]$ is equal to the median of all pixel values lying in a window of size $n \times m$ with its center in j_{xy}. [13]

3.5.3 Canny Edge Detector

John F. Canny has developed an edge detection algorithm [6] in 1986, that is still often used in computer vision. An edge in an image is a region where the brightness changes sharply, or more formally, a region with significant differences between adjacent pixels.
Canny states the following three performance criteria as motivation for edge detection in his work:

1. *Good detection.*
2. *Good localization.*
3. *Only one response to a single edge.*

The first criterion means to achieve a low probability of not marking a real edge and a low probability of marking a non edge point. The second one describes the importance of an exact location and the last one forbids a duplicate detection of a single edge.
To achieve these three criteria, Canny developed an algorithm which has the following structure:

1. Blur: To remove noise the image is blurred with a Gaussian filter.
2. Intensity Gradients: The gradient of intensity from one pixel to all adjacent pixels is computed. This results already in a rough edge image.
3. Non-maximum suppression: A Non-maximum suppression filter is used to thin edges. Only the maximal parts of thick edges remain. This is done to provide a good localization of the edge (criterion 2).
4. Hysteresis: Apply two thresholds to the remaining gradients. If above the upper threshold, consider it as edge. If below the lower threshold, reject. If between the thresholds, consider it as edge, only if connected to a pixel already considered as edge.

The two mentioned thresholds can be chosen freely and specify how strong an edge has to be. The choice of these values increases or decreases the likelihood of identifying edges.

3.5.4 RANSAC Plane Detection

RANSAC was first introduced by Robert Bolles in 1981 [3] and is an abbreviation for random model sample consensus. RANSAC is an iterative method that tries to fit a mathematical model into a given dataset and to estimate the parameters of the model. It assumes that the input data is divided into inliers, that fit into the model and outliers, that do not fit. RANSAC has one requirement: for every model that should be found, it needs a way to estimate the parameters from a given dataset. The algorithm is structured as follows:

1. Choose random subset of the input dataset.
2. Estimate model parameters from the picked subset.
3. Test all other data against the found model.
4. Check the model error, as the relation of found inliers to the complete dataset.
5. If it is not good enough, reject the model.
6. If it is good enough, accept the model. Then refine it by recompute steps 2, 3 and 4 but with the found inliers instead of a random subset.

This is repeated a fixed number of times, returning every time a rejection or a found model. The model with the lowest error is considered as the best result.

3.5.5 Feature Detecting & Tracking with Optical Flow

This section describes the concepts of *Optical Flow* (OF), Features, Feature Detector and Feature Tracker. In addition some linked algorithms are briefly described.

Optical Flow

Berthold Horn and Brian Schunk's defined OF in their work *Determining Optical Flow* from 1981 as follows:

Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image. Optical flow can arise from relative motion of objects and the viewer. [12]

This quote describes a phenomenon which is normally easily detectable by humans, if they observe two similar images, which only differ a little. That could be for example a slightly different viewpoint, or the same viewpoint and movements of objects in the image. Then the human brain can easily detect which movement happened, by comparing these images, but for a computer this is more difficult.

Feature

Features, by definition, are locations in the image that are perceptually interesting [17]

This quote from Manjunath describes a feature as *interesting* point in an image. And with interesting he means it differs from other parts in the image, or in other words, its more or less unique. Algorithms which try to find such *unique* features are called *Feature Detectors*.

Feature Detector

A Feature Detector tries to find one or more features in a given image. Often a rating function is implemented to determine how good a found feature is. There are several ideas how to compute if a location is interesting and therefore a good feature or not. Good Features to Track is one possible algorithm to achieve this. It is described and explained later.

Feature Tracker

Let $I(x)$ and $J(x)$ be two image functions which assign a pixel value to every pixel x. Additionally we assume that we have an already found feature called u in image I which describes a point in the real world. The goal of a feature tracker is to find a corresponding feature v in J. By corresponding a feature which describes the same point of the real world is meant. This works only if the images are somehow similar, and the point which is described by u also exists in the second image J. The connection between the two features can then be described with a distance d.

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

An example for similar images are pictures of the same thing from slightly different view points, a situation which is often found in stereoscopic camera scenarios. Another example is a video. While a scene is filmed, two consecutive frames often differ just a little.

Good Features to Track

The algorithm *Good Feature to Track* was developed by Jianbo Shi and Carlo Tomasi in 1994 [25], but is a further development of the often called *Harris Detector* from Chris Harris and Mike Stephens [9] developed in 1988.
The basis of the *Harris Detector* is the concept of how to rate a feature as good or not good. The idea is quite simple but has a great effect. A feature location is good, if a small move in all directions has great effect. And the opposite: A feature location is bad, if a small move in all directions has very little to no effect. Or condensed one can say: the higher the effect, the higher the rating.

To explain the idea we take a look at simplistic example Figure 2 with its four possible features A, B, C and D. If one moves feature A, there is literally no change in its content. Therefore this is not a good feature. The content of feature B changes if you move it up or down, but not if you move it left or right. Feature B is therefore better than A but still not perfect. Feature C and D are very similar, as they change both a lot when moved in every direction, they are therefore the best features in our example.
What we observe here, in more general terms:. With these feature detectors, corners and blobs are classified as the best features. Edges are still good but are just used if no corners, intersections or similar are available.
This behavior seems also very intuitional. If one tries to solve a classic jigsaw puzzle, often the edges and especially the corners are the easiest parts. In contrast, in jigsaw puzzles parts which are nearly the same color are very hard to solve.

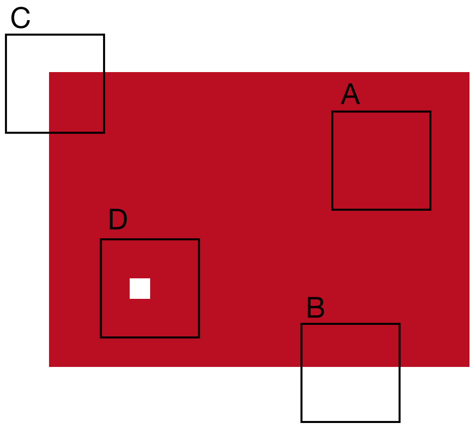


Figure 2: Example Features

Lucas Kanade Method

This algorithm was developed by Bruce Lucas and Takeo Kanade in 1981 [16]. It solves the problem of image registration, i.e. matching one image to another similar image.
To simplify the explanations, this algorithmis presented in the first dimension only, but can be generalized to multiple dimensions.
Let $F(x)$ and $G(x)$ be two functions which describe images that should be matched. The connection of these two functions, is described with the distance h_x at every point x like this:

$$G(x) = F(x + h_x) \tag{1}$$

The mean distance is then computed with (1) and:

$$h = \frac{\sum_x h_x}{\sum_x 1} \tag{2}$$

The goal of the algorithm is to find an optimal h so that $F(x)$ and $G(x)$ are as *similar* as possible. To compute an approximation of this h, *Newton's difference quotient* is used:

$$F'(x) \approx \frac{F(x + h_x) - F(x)}{h_x} \tag{3}$$

With the use of (1) and (2) this equation follows:

$$h \approx \frac{\sum_x \frac{G(x) - F(x)}{F'(x)}}{\sum_x 1} \tag{4}$$

To refine the found h a *Newton-Raphson* iteration is used with the following equation:

$$\begin{aligned} h_0 &= 0, \\ h_{k+1} &= h_k + \frac{\sum_x \frac{G(x) - F(x + h_k)}{F'(x + h_k)}}{\sum_x 1} \end{aligned} \tag{5}$$

The iteration repeats until an appropriate termination criterion is fulfilled. For example that is a maximal number of iterations or if the distance between h_{k+1} and h_k is below a certain threshold.
In the final implementation the computation of (5) is additionally enhanced with weighting and an alternative approximated derivation. For more details see [16].

Image Pyramid

Let $I([x,y]^T)$ be a function which describes the image with width n_x and height n_y. This function assigns a pixel value to every $x \in [0, n_x - 1]$ and $y \in [0, n_y - 1]$. Then a pyramidal representation L with depth n of this image is defined as follows:

$$\begin{aligned} L_0 &: \begin{pmatrix} [0, n_x - 1] \\ [0, n_y - 1] \end{pmatrix} \rightarrow \mathbb{R} = I(\begin{pmatrix} x \\ y \end{pmatrix}) \\ L_{n+1} &: \begin{pmatrix} [0, \frac{n_x}{2^{n+1}} - 1] \\ [0, \frac{n_y}{2^{n+1}} - 1] \end{pmatrix} \rightarrow \mathbb{R} = L_n(\begin{pmatrix} 2x \\ 2y \end{pmatrix}) \end{aligned} \tag{6}$$

This results in an image function at every level of the pyramid, with the original image I as basis. The image at a higher level is basically a representation of the last lower level with a reduced resolution and half the width and height. With this recursive definition one can for example build an image pyramid L with depth 3, by computing L_1, L_2, L_3. If an image I has a resolution of 320×240, then this results in L_1 having 160×120, L_2 80×60 and L_3 40×30.

Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker

The algorithm developed by Jean-Yves Bouguet in 2001 [4] combines the Lucas Kanade method with image pyramids to build a robust optical flow detector, which is highly suitable for feature tracking. The algorithm works in principle as the above mentioned general feature tracker. For a given feature u in a given image I the feature tracker tries to find a corresponding feature v in another image J. A simplified version of the algorithm looks as follows:

```
Initialization
    Create image pyramid with desired size $m$ image $J$.
    Initialize $v \leftarrow u/2^{m+1}$.
Repeat $L = L_m$ down to $L_0$ with step $-1$
    Initialize guess from higher pyramid level  $v \leftarrow 2v$
    Repeat until termination criteria
        Lucas Kanade  refine $v$ in image $L$ with the above mentioned Lucas Kanade iteration.
Result  return $v$
```

There are two main advantages of combining the Lucas Kanade method with the image pyramids. Firstly, the Lucas Kanade converges notably faster because of the approximated position of v from the last higher pyramid level. And secondly, even a large distance between the feature positions in the different images can be detected, because a distance in L_0 is 2^m times shorter in L_m. For additional details see [4].

3.6 Hardware

In this subsection the functionality, capabilities and limitations of the required hardware are descibed and explained.

3.6.1 Color and Range Camera

A color and range camera, is effectively a normal color camera with an additional range sensor. One example is the *DepthSense 325*[6] camera from *SoftKinetic*, which works with the time-of-flight (TOF) principle. This camera was used to develop and test the software described in the following sections. (Note: Instead of *range*, the term *depth* is often used. Therefore these cameras are sometimes abbreviated as *RGB-D* camera)

Capabilities

A range sensor works like a normal color sensor, but instead of color values it measures the distance between itself and the scanned area at every pixel. The result is an image (often called *depth map*) with distance values at every pixel. With

[6] http://www.softkinetic.com/Store/ProductID/6

these values and the information of the angles of the field of view, it is possible to calculate X, Y and Z coordinates of the corresponding real world points in a coordinate system, in which the camera sensor is the origin. Figure 4b shows this behavior, where R is a point in the depth map, and O an object point with three coordinates.

Functionality

There are different kinds of depth cameras, which function based on different principles. The focus here is on the functionality of TOF cameras. A TOF camera consists of two parts, an infrared light source and an imaging sensor which responds to infrared light. The light source emits a short flash, which is then captured by the sensor. As the name suggests, the camera measures the elapsed time between flashing and detection. This measurement is done separately for every pixel of the imaging sensor. [15]

Limitations

Although range measurement offers great new options, there are several limitations that should be considered. A camera can only measure the position of objects that are in the line of sight. If one object covers another, just the one in front will be scanned. The next important limitation is accuracy. No camera is perfect, but TOF cameras in particular are lacking in precision. The uncertainties in measurement occur also in adjacent pixels, resulting in a noisy image.
Another limitation has to do with light absorbing surfaces. Every system that works with detecting emitted light has problems with dark and therefore more absorbing surfaces. If no or just little light is reflected, the image sensor is not able to detect it. Therefore the darker the surface the less accurate is the measured distance. An example for this behavior can be seen in Figure 3. The region in the middle of the scene is a dark surface. The color sensor can easily capture the colors as it can be seen on the left side. The result of the depth sensor is shown on the right side. At every white pixel a measurement was possible. As is can be seen because of the large black reason, the depth sensor was not capable to capture the dark surface because of to low reflection.
Another limitation that has to be considered but can be handled is the different perspective of color sensor and depth

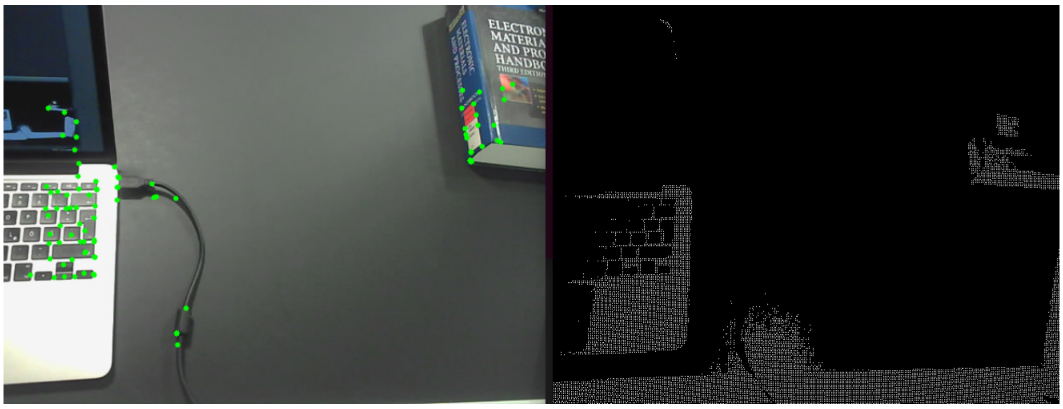


Figure 3: Example for a Dark Surface

sensor. The color sensor might capture a bigger or smaller area of the world. This has to be kept in mind every time a transition between the two systems is made.

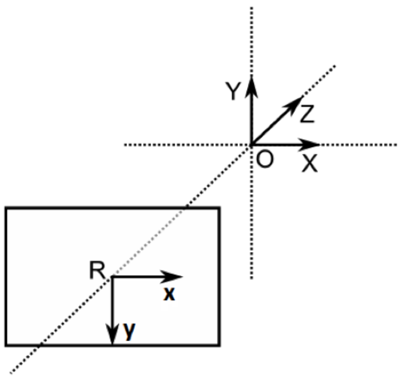
3.6.2 Augmented Reality Glasses

Augmented reality glasses are a head-mounted device, which are able to add something into the users field of view without completely obscuring it. To develop and test the new software presented here, augmented reality glasses of the type Vuzix Star 1200 shown in Figure 5 were used.

Capabilities

Augmented reality glasses offer the possibility to display information directly into the user's field of view. If done correctly, and no important part of the real world is obscured, these glasses could be worn everywhere, for example in the office or even while hiking or cycling.

(a) DepthSense 325, time-of-flight camera



(b) Camera coordinate systems

Figure 4: Range Camera

Figure 5: Vuzix 1200 Star Augmented Reality Glasses

Functionality

In each glass of the device there is an area where digital content can be displayed. This is done either with transparent displays, or with reflection and lenses, so that the effect of an transparent display is achieved, without really using one.

Limitations

The displaying area of the glasses is rarely as big as the users field of view. As a consequence they might reduce the field of view. This can be dangerous in unsafe environments because incoming danger might be detected later than usual.

3.7 Stereoscopic Computer Graphics

John Roese discovered in 1979 [23] that stereoscopic vision is well suited for many applications involving three dimensional computer graphics. The human vision system is capable of extracting depth information using solely the small differences between the visual informations of both eyes, which occurs because of the different position of the two eyes. Stereoscopy uses this to generate the illusion of perceiving depth from two dimensional images. All current rendering engines for three dimensional graphics like OpenGL or DirectX are using 3D world models and virtual cameras. A virtual camera represents the viewpoint of an user. The rendered image is then an excerpt from the modeled 3D world, as it is seen from the position of the camera.
It is easily possible to generate a stereoscopic effect with computer graphics by using two virtual cameras. The cameras have to look slightly in the same direction from similar positions. If users then perceive the different images, one on the left eye and the other on the right eye, they are able to perceive the stereoscopic illusion.

4 System Architecture Design

4.1 Overview

In this section the general structure of this application will be explained in a top to bottom direction. I start at a general overview and then break the separate modules into smaller parts with more details. If there were important design decisions they are stated there as well, together with reasons why the decision was made. The last subsection 4.6 descibes which other libraries are needed and how the system can be used by third party developers.

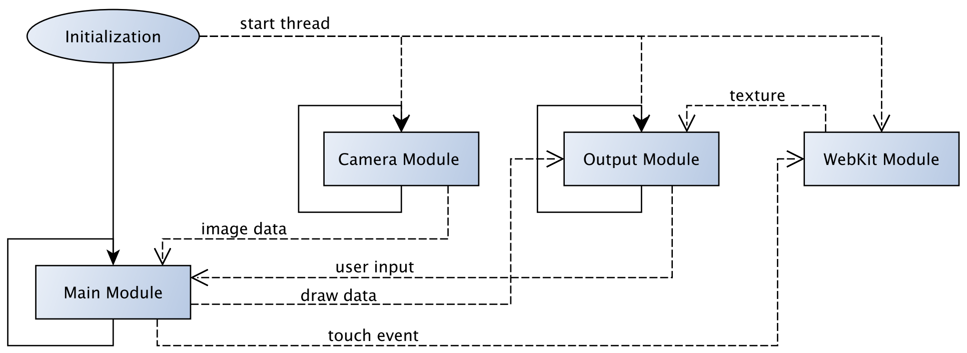


Figure 6: Overview of the Four Main Modules

After the application starts, some initialization is carried out first (Figure 6). Different threads for camera input and glasses output and the WebKit engine are started. Also some configuration takes place in this phase. For example the camera is told which data in which format is needed. And other modules are told which image sizes should be expected. After the initialization, the main module starts and is waiting for the first incoming data from the camera.

The general work flow is as follows: After the camera captured a frame the camera module sets a flag that fresh data is available. Next the main module recognizes this flag and processes the newly arrived data. This processing is the phase with the most computations and will be discussed in detail later. After the processing is carried out the output module is informed and starts to render the new data. The used output engine has also a built-in capability to capture user input in form of keystrokes. This input is either handled instantly if it concerns the output directly or it is passed on to the main module. After one of this iterations it starts again as expected.

The WebKit module is inactive most of the time. It only becomes active if requested and then reacts on the request, which means loading and rendering a new *HTML* page.

4.2 Camera Module

The module responsible for retrieving the image data from the camera is the camera module. This module works hand in hand with the *DepthSenseSDK*[7] to communicate with the *DS325* camera. This module can be understood as a wrapper for the *DepthSenseSDK*, or as interface between the main module and the camera. This module is designed to be as uncoupled as possible from the remaining application. The main motivation behind this, was to make it easy to replace, to easily support other cameras.

As already mentioned in the last subsection 4.1, this module is initialized and then works as a thread on its own. After the configuration is passed to the SDK in the initialization phase, nothing else but passing SDK events is done. For example the SDK provides a *onNewDepthSample* event. This triggers the execution of a function in the camera module, to save the pointer to the raw data. In addition the flag *newDepthData* is set true. This flag as well the pointer are global variables shared by the main modules. All further computations are then handled by the main module.

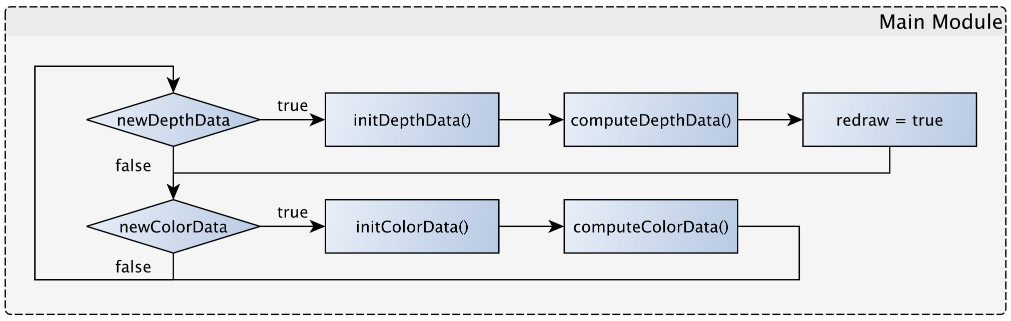


Figure 7: Flow Diagram of Main Module

4.3 Main Module

As seen in Figure 7, after new captured color or depth data arrives, corresponding initialization is done. This is mainly class instantiation and preprocessing in form of filtering to provide enhanced data to the following computation submodules. After the initialization, the main computations like feature tracking and surface detecting are executed.
As it can be seen in Figure 7, the depth and color initialization and computations are separated and there is a reason behind it. The camera might be able to provide different frame rates at color and depth capturing. Using the given setup, the software is not limited to the lower one, but can operate with both rates and use the given resources optimal. This is only possible because depth and color computations are separated as well.

4.3.1 Initialization of Depth Classes

The following classes are responsible for storing and preprocessing. As mentioned in subsection 3.3 the depth data is stored in a *Pointcloud* and in *Depth Images* using data types from the Point Cloud Library and OpenCV. One reason behind this classes is the encapsulating these data types. This provides a lower coupling of other modules, because they do not depend directly on the libraries. The classes could also be seen as black boxes, which hide their logic.

Cloud : This is a class directly representing a *PointCloud*.

ConfidenceImage : This class derives directly from camera input and calculates a confidence mask, to indicate depth pixel which hold valuable data, and pixel which are too noisy.

DepthImage : Here the confidence mask and median filtering is used to enhance the depth data. To speed up further computations the representation of depth values is narrowed down from 16 to 8 bit.

EdgeImage : This class uses the just generated 8 bit depth representation and the canny edge detection algorithm. The generated mask indicates if an edge exists at a given point on the image.

UVImage : As mentioned in subsection 3.6.1 the perspective of color and depth images are different. This image holds normalized color coordinates of depth pixels and is used by the class *ColorDepthConversionHelper* to display depth coordinates in the color image and vice versa.

4.3.2 Depth Computations

The computations on depth data can be separated in two groups, the first one is responsible for touch event detection and also contains finger detection and tracking. These are all executed for every frame. The second group is about detection surfaces and fitting a screen into them. As you can see in Figure 8 this is only triggered after an user input and not every frame. The reason for that, is not only that it is not necessary every frame, but also that this takes more than 33ms. And as explained in subsection 3.4 this is not fast enough, if 30 frames per second are desired.

[7] `http://www.softkinetic.com/Support/Download`

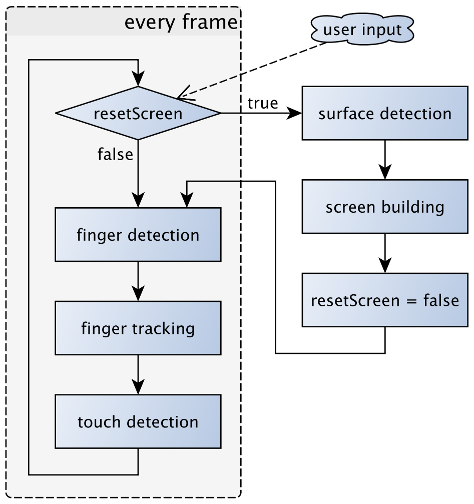


Figure 8: Flow Diagram for Depth Computations

Finger Detection

This component works mainly with the *EdgeImage* and *DepthImage* and detects fingers based on edge pairs. The direction is also detected, meaning which end belongs to the hand and which is the fingertip. The implementation in detail is quite complex and uses the property of a finger that it has edges on both sides. And also that the distance from finger to camera is approximately as far as the distance from camera to the edges. Therefore in principle everything that is shaped like a tube and has about the size of a finger is detected. Big pencils for example could also be used for interaction. A detailed explanation can be found in subsection 5.4.

Finger Tracking

Once a finger is detected the finger tracker starts to track it. The module basically tries to match all detected fingers in the current frame with those from the last frame. The measure if two fingers are similar is the distance of their start and end points. The finger tracker is necessary, because sometimes the finger detector detects false fingers in background noise.

Touch Detection

The Touch Detection component tries to detect when and where the user touches a surface with his finger. This works on every surface, but is enabled only on virtual screens. For every tracked Finger, the component looks at the square shaped area around the fingertip. If the depth values in this area are similar to the depth value of the fingertip, a touch event is detected. The lack of precision which was mentioned in subsection 3.6.1 results in the behavior that a touch event is detected not only if the user really touches a surface, but at any point where the finger is about 1 cm away from a surface.

Surface Detection

As the name suggests, the Surface Detection component detects surfaces in the view of the user. The component only detects planar surfaces, to use them as flat displays. It is powered by an algorithm from the Point Cloud Library and uses a random sample consensus (*RANSAC*) approach to fit a plane in the sensed Pointcloud. The algorithm is explained in subsection 3.5.4. As a result an instance of *Surface* is returned, which holds the indices of the points that belong to the surface.
Besides the *RANSAC* approach, several other surface detection algorithms where tried, but *RANSAC* led to the best results in the shortest time.

Screen Building

Once a surface is found, the class *RectangleMaximizer* tries to find a rectangle that fits into the surface. The class uses the inliers to create a mask, which is then dilated to erase holes in the surface. After that a rectangle is fitted into this mask, with the help of a custom algorithm which starts in the center of the user's view and expands the rectangle in four directions. The detailed algorithm is explained later in subsection 5.4.

4.3.3 Initialization of Color Classes

As color data arrives, a new instance of *ColorImage* is created. *ColorImage* uses *OpenCV* data types and algorithms to store data and also creates a converted greyscale image.

4.3.4 Color Computations

All computations that are done on color images, are solely used to estimate the movement of the camera, thus the movement of the user's head. This is done in an separate subsystem explained in subsection 5.5.

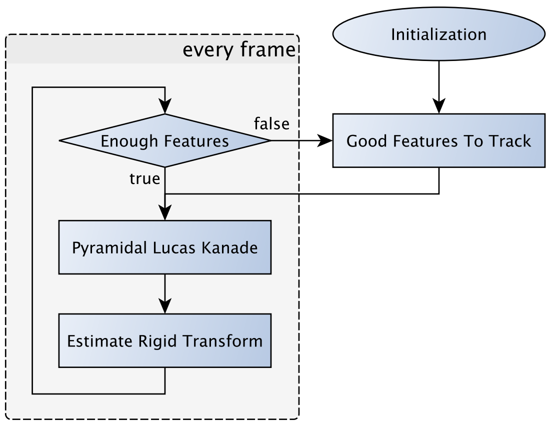


Figure 9: Flow Diagram for Color Computations

As you can see in Figure 9, the subsystem uses mainly the algorithms *Good Features To Track* and *Pyramidal Lucas Kanade Feature Tracker*, both explained in detail in subsection 3.5.5.
There are several reasons for the use of optical flow and especially the latter of the both algorithms. The *Pyramidal Lucas Kanade Feature Tracker* is fast and performs solid if the movement between to frames is not too big. There are other feature tracking approaches that are just fast enough to perform real time, but the algorithm has to share the given time span with the depth computations. In fact this was the only tested feature tracking algorithm capable of holding the needed time requirements.

4.4 Output Module

The output module is responsible for bringing content to the glasses. The module is built on OpenGL and with the help of GLUT, which is the OpenGL Utility Toolkit. This module creates an output window, which contains the output of the system in an side by side view. This output is then send to the glasses, where each of the side by side images are displayed on one eye-glass. In addition to the presenting content, this module also captures input from the user through the keyboard. Some of the input is directly handled inside the module, others are sent to the main module.
OpenGL was chosen over other rendering engines, because it provides a very simple and easy to learn interface. Furthermore, it is platform independent and already pre-installed on Linux, PC and Mac systems. In addition it is easily capable to produce side-by-side or other 3D content. Moreover, emphasis was placed on only using simple OpenGL directives, to make it available for mobile systems in the future.

Work Flow

As mentioned in subsection 4.1, the output module works in his own thread. After some initialization the *glutMainLoop* is started. This is an internal looping function which manages several events, e.g. when something is drawn, when the windows size changes. It also captures the user input. The scene is drawn again if an internal flag is set, the setting of this flag is achieved by calling the glut function *glutPostRedisplay*. The *idleFunc* is executed when nothing else has to be done.

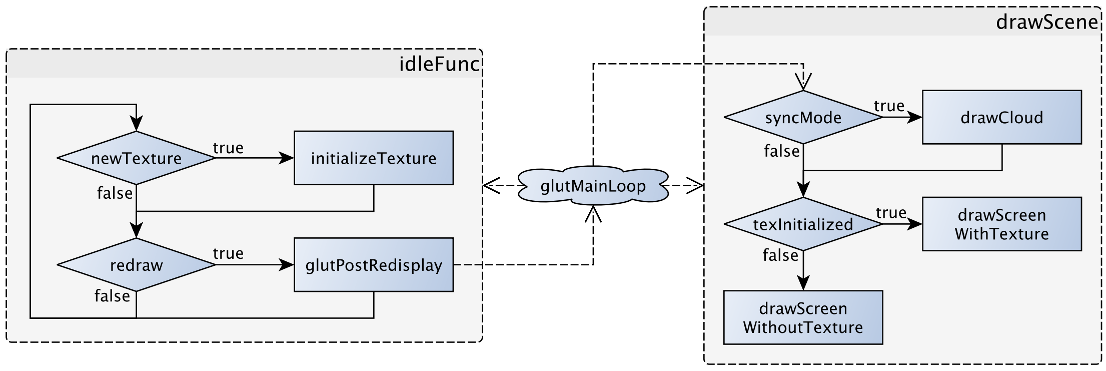


Figure 10: Flow Diagram for the Output Module

The general work flow of the OpenGL output thread is shown in Figure 10. The *idleFunc* checks if a new texture was computed by the *WebKit* module. If this happened, the new texture is initialized, meaning it is packed into the correct data type and stored into the video memory. Then it checks if the main module has finished computing new data, this happens through the redraw flag. If that's the case, the redrawing is triggered through *glutPostRedisplay*.
The drawing is done by the function *drawScene*, as shown in Figure 10. Which components are drawn depends on the synchronization mode and the availability of textures.

4.5 WebKit Module

The Webkit module is mostly powered by Awesomium, a lightweight and simple HTML rendering engine. Awesomium has an API for C++ and C#, and is capable of loading local web pages as well as retrieving data over the network. It manages all underlying http traffic on its own and is able to render a Bitmap which is then used as OpenGL texture.
The reason behind choosing HTML as basis for content generation is because it is very easy to display text, images and other media. In addition the Internet is accessible with this solution, so own content generation is not needed at all.

In the main initialization phase, the module starts the Awesomium engine, loads the first page and then renders it. The resulting Bitmap is then saved to memory and the output module is informed that a new texture is available. Then it waits for further requests from the main module. After the main module successfully detected a touch event, it is passed to the WebKit module and triggers a mouse click on the current page. And then, of course, renders it again and passes the results to the output module.

4.6 Dependencies and Reusability

4.6.1 Dependencies

The provided application has some libraries on which it depends. The utility library Boost[8] is used throughout the whole application and manages things like threading and timers.
As already mentioned in subsection 4.3.1, the depth and color classes use data types from OpenCV[9] and Point Cloud Library[10]. Also different algorithms from these libraries are used at various locations. Therefore the main module depends on those two libraries.
As mentionend in subsection 4.2 the camera module depends on the DepthSenseSDK which manages the communication with the camera. If another camera should be used, this module is also exchanged.
Because the output is based on OpenGL, this module needs the corresponding OpenGL[11] library and additionally the functionalities of GLUT which are provided by the freeGLUT[12] libraries.

[8] http://www.boost.org
[9] http://opencv.org
[10] http://www.pointclouds.org
[11] http://www.opengl.org
[12] http://freeglut.sourceforge.net

As already mentioned in subsection 4.5, the WebKit module is mainly based on Awesomium[13] and therefore needs the corresponding libraries.

4.6.2 Reusability

Considered as a whole, the application does not communicate with the external world in form of an programmable interface. But the main components are designed to be exchangeable, what is mainly achieved through loose coupling.

Accessing Touch Events

The resulting touch events can be easily accessed through the WebKit engine. The possibility to display every needed web page, offers a lot of options. For example, if the system should be used to evaluate different touch keyboard types, probably the best idea would be to just model it in HTML. But it would be also possible to display a simple image, and capture the touch events through mouse click events with Javascript.

Content Generation

The WebKit module offers a lot flexibility through HTML, CSS and Javascript, but if this is not enough, a different content generation module could be used. The whole module is easily exchangeable, the only constraint that has to be fulfilled is that the new module also generates something that can be loaded as OpenGL texture.

Camera

Similar to the WebKit module, the camera module can be exchanged. The camera module can be seen as an interface to the camera, therefore if another camera should be used, this module needs to be adapted or rewritten.

Output

If any other displaying hardware should be used, for example single display glasses or even a smart phone, this module has to be changed or exchanged.

Small Submodules

All other and smaller submodules that are mainly contained in the main module are also designed with aspect to low coupling. But they have some kind of hierarchical dependency, for example, *FingerDetector* relies on *Finger*, *EdgeImage* and *DepthImage*. *FingerTracker* uses only itself and *FingerDetector*, but to use it you need, of course, also all dependencies of *FingerDetector*. With this respect, all modules can be used independently.

[13] `http://www.awesomium.com`

5 Implementation

5.1 Overview

In this section, some information about the technical background of the software the finger detection algorithm and the handling of the software is exmplained.

5.2 Technical Background

The whole system is written in C++, for several reasons. The main advantages of C++ are on one hand execution speed, which is a central aspect for real time applications and on the other hand, it is object oriented and therefore supports the modularity of the system. An additional aspect of choosing C++ is about the depending libraries. The DepthSenseSDK and the Point Cloud Library are not only available in another language. Despite OpenCV has several wrappers for Java, Python and C# it is natively written C++.
The build process is guided by CMake, which gives the possibility to easily link libraries and headers from different sources. Furthermore CMake works with a lot of different compilers and is platform independent. The System is developed, built and tested under Linux, but it could also be built for Windows or Mac OS X.

5.3 Finger Detector

In this subsection the finger detection algorithm is explained in more detail. As already mentioned in subsection 4.3.2 the finger detection uses instances of the *DepthImage* and *EdgeImage* classes, as well as the class *Finger* with its subclasses *FingerSliceTrail* and *FingerSlice*.

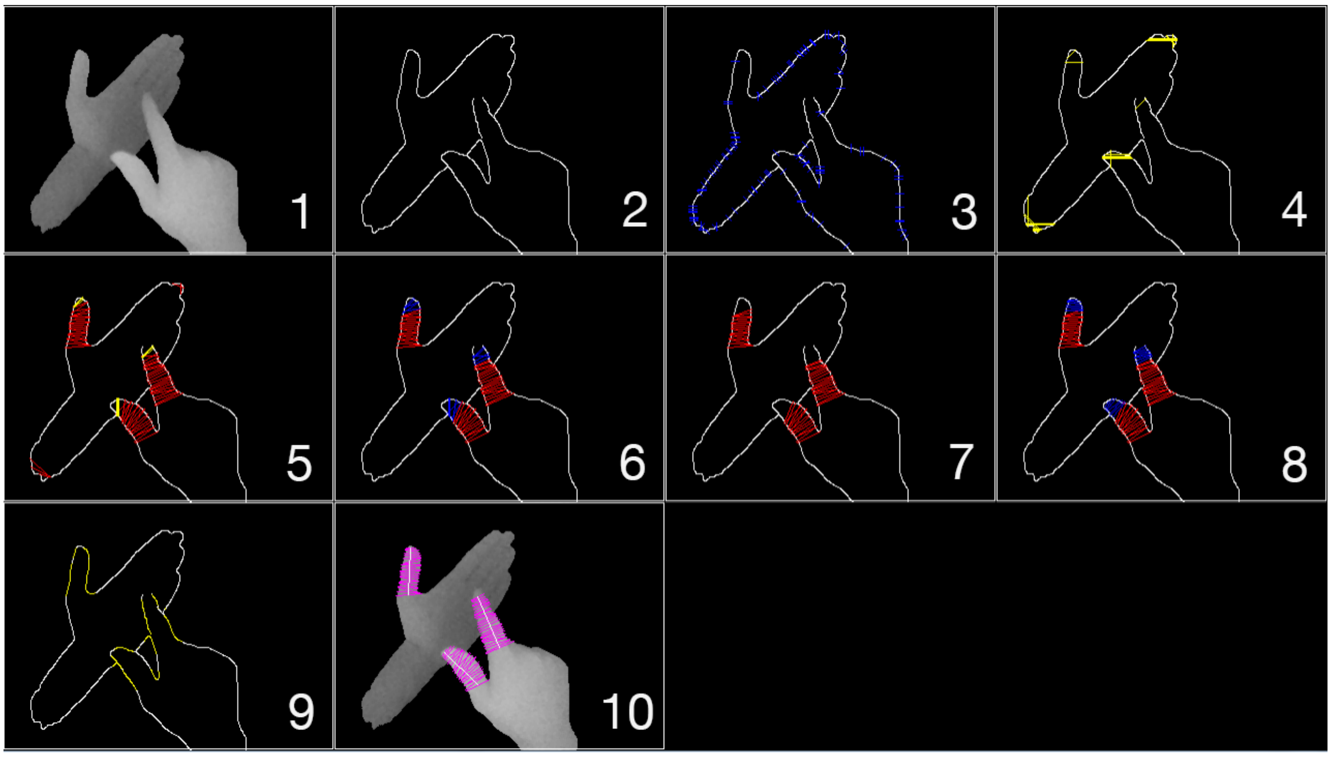


Figure 11: Process of Finger Detection
1. depth image 2. edge image 3. edge directions 4. initial finger slices 5. start slice (yellow) + first expansion (red) 6. inaccurate start slices (blue) + lower slices (red) 7. lower slices (red) 8. lower slices (red) + second expansion (blue) 9. finger edges 10. final finger

The different steps for the finger detection are shown in Figure 11, the following part references to the different frames of this figure. Frame 1 is the perceived depth image, and frame 2 the corresponding edge image. To compute frame 3,

the algorithm iterates over the edge image and calculates the edge directions. Next the algorithm tries to find orthogonal finger slices at the found edges, by trying to find another edge along this orthogonal direction. A valid finger slice is found if the slice forms a hill, meaning the edges have a higher distance to the camera than the point between them. Without this last constraint, also free spaces between two fingers would be mistakenly found. In frame 4, found finger slices are marked.
Then the algorithm starts to expand a slice into a finger slice trail, as shown in image 5. To find a trail of slices along a finger the algorithm iteratively takes the mid points of the previous couple of slices and calculates an average direction. The next slices are then searched along this direction, until the finger ends. The algorithm needs some iterations to make a good estimate of the fingers direction. Because of that, the aligning of the first few slices is often not very accurate, shown with the blue slices in frame 6. To correct this, the wrong aligned slices are deleted (frame 7), and then recalculated by expanding the trail in the reverse direction than before. The result in image 8 seems similar to image 6, but on closer inspection, the top blue slices are much better aligned.
After all slices of a finger are found, the corresponding edges are removed from edge image, and the next start slice is expanded. The removing is important to ensure that no duplicates are found. Frame 10 shows the end result with a line segment between tip and hand point.

5.4 Surface Detection and Screen Building

This subsection explains the algorithms that work together to build the virtual screen. This part of the software needs solely the Pointcloud representation of the measured depth data as input, but uses algorithms from both, the Point Cloud Library and OpenCV.
Before the surface detection starts, the Pointcloud is preprocessed with a median filter to smoothen the image. Then the next step is to find a plane surface of the current frame in the Pointcloud. This task is done by an instance of *SurfaceDetector*, that uses a RANSAC algorithm from the Point Cloud Library which works as explained in subsection 3.5.4. The result is an instance of *Surface* that wraps a list of indexes that belong to the found plane surface. An example for an found surface can be seen on the left of Figure 12.

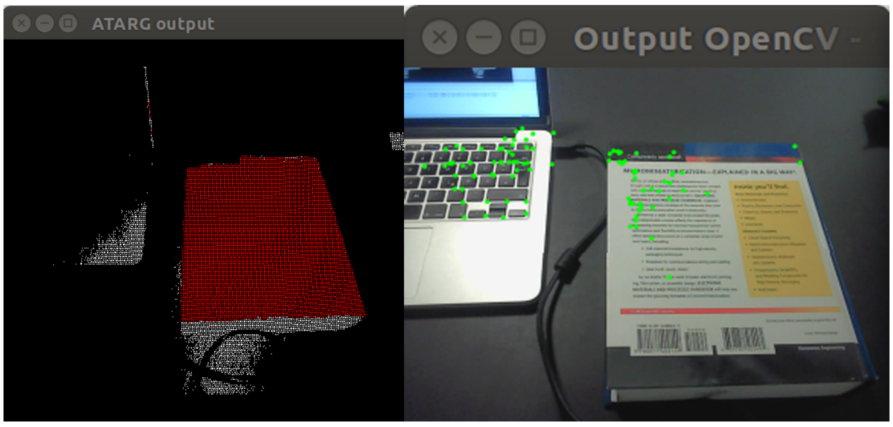


Figure 12: Detected Surface

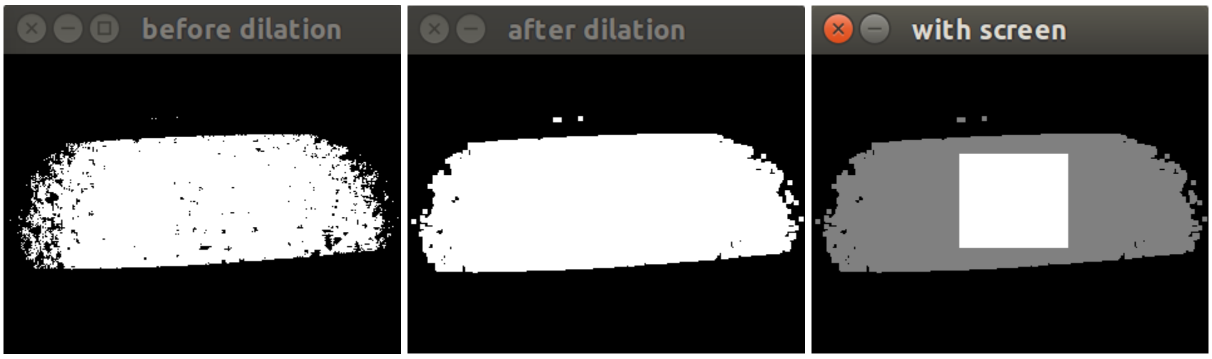
The building of a rectangular screen from the irregular shaped surface is done by the class *RectangleMaximizer*. The class maximizes a rectangle into a surface, by expanding the rectangle as far as possible in all four directions. A rectangle is defined by the top-left and bottom-right corner. The initial values for the both corners are here chosen as the center of the Pointcloud.

The algorithm works as follows:

1. Surface preprocessing Create an image mask by using an empty OpenCV image and set every point, that belongs to the surface, to value 1 (Figure 13a). Dilate the mask with size 4 to erase remaining gaps (Figure 13b).

2. Rectangle expansion Repeat until all four sides are outside the surface:

   For all four sides: Check if the side is still completely inside the surface. If its true, expand it by one pixel.



(a) Mask before Dilation **(b)** Mask after Dilation **(c)** Mask with found screen

Figure 13: Example Screen Building

3. Postprocessing Reduce the size of the rectangle by a threshold, to compensate the dilation and ensure that it fits into the surface. The result can be seen in Figure 13b

The result of the above algorithm is than used to initialize a new instance of *Screen*.

5.5 Screen Position Tracking with Optical Flow

The subsystem explained in this subsection is responsible for keeping the screen at a certain place in the real world. It uses *ColorImage* and the fast implementations of the two algorithms *Good Features To Track* and *Pyramidal Lucas Kanade Feature Tracker* both explained in subsection 3.5.5.

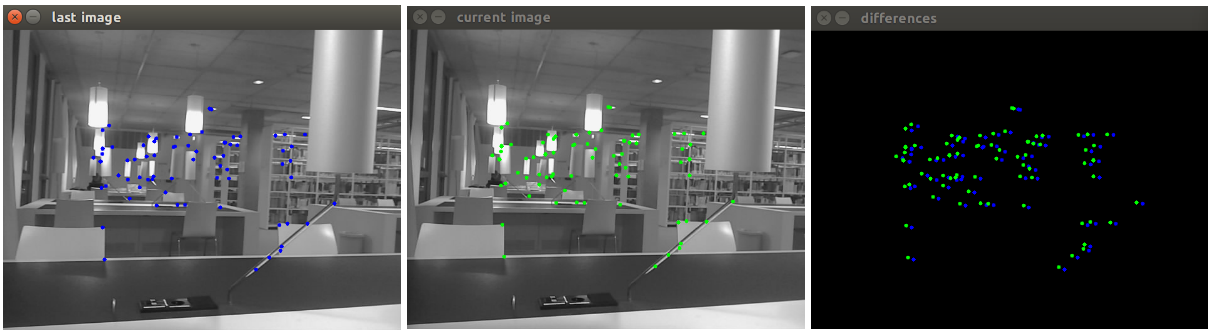
All used algorithms from the OpenCV library are briefly explained here:

goodFeaturesToTrack : Finds features in the image which are optimal for tracking. For details see subsection 3.5.5.

calcOpticalFlowPyrLK : Uses a list of features from the last frame and tries to find the corresponding features in the current frame. It is possible that some features are not found, therefore the number of features may decrease. This is mainly done by a pyramidal implementation of the affine Lucas Kanade feature tracker, described in subsection 3.5.5.

estimateRigidTransform : Uses two lists of features and estimates a rigid transformation matrix, which describes the mean transformation between the particular features.

cornerSubPix : Uses a list of features and refines them. Refining means enhancing the location of the feature with more detail than the pixel grid. The resulting location can then even be between pixels.



(a) Features from last frame **(b)** Features from current frame **(c)** Features from two frames

Figure 14: Example Features

The subsystem finds features, tracks their position and estimates an camera movement. One iteration of this subsystem is described with Figure 14. The subsystem starts with the image and the features from the last frame (Figure 14a). Then it computes the corresponding features in the current image (Figure 14b). The difference between the features (Figure 14c) is used to compute the head movement. In detail this works as follows:

1. Check if features from the last frame exist. If not jump to step 2
   a) Compute the optical flow between the last image and the current image with the algorithm *calcOpticalFlow-PyrLK*. This algorithm returns a list of corresponding new features, and the information which feature was found and which not.
   b) Erase features that were not found in the current image, from both the list of last features and the list of current features.
   c) Check if features still exist in the two lists. If there are, compute a motion transformation matrix with *estimatRigidTransform*
2. Check the number of features in the feature-list. If the number is too low, compute additional features of the current image with *goodFeaturesToTrack* and refine them with *cornerSubPix*.
3. Save the current feature and the current image as the last feature list and the last image, to use it in the next frame.
4. Update the screen by multiplying the corner of the rectangle with the computed motion transformation matrix

5.6 Handling

The software is started from the command line. There is only one option that can be specified and that is the URL of the web page that is displayed on the screen. After starting the application, the synchronizing mode is active, where all input from the depth camera is displayed on the screens. In this mode the user can adjust the perspective and viewing point with the arrow keys, *W*, *A*, *S*, *D*, +and -. The goal of this procedure is that measured depth points are at the same place as the real corresponding objects. This is done to synchronize the position of the user's eyes and the cameras position and can also be called calibration mode. The synchronizing mode can be turned off with *R*. Then the user can create a new screen by looking straight to a surface and pressing the button *E*. As soon as a screen is visible, the user can start using the screen as touch screen.

6 Limitations and Possible Extensions

6.1 Hardware

The hardware represented the main limitation in the current project. Both the TOF camera and augmented reality glasses are sold commercially but are very new technologies. They functioned but probably need another few years of development before they will provide fully satisfactory results and a good experience. In the following, a more detailed description of these limitations is given.

6.1.1 Camera

The *DS325* TOF camera used has a depth resolution of 320×240 pixels and a precision below 1.4cm. This results in a very noisy image. A human can easily recognize what the camera films, but for computers this is more difficult. For example feature detecting and tracking on these depth images was practically impossible.
This behavior affects mostly the surface detection, the finger detection, tracking and touch detection, because they rely solely on depth data.
A better camera with more precise images and a higher resolution would definitely increase the overall experience of this software.

6.1.2 Glasses

The used augmented reality glasses *Vusix Star 1200* has one main limitation that makes their usability suboptimal. The area in which the field of view can be augmented is not big, and it is just a small part of the complete field of view of the user. For example, the glasses are capable of projecting a maximal 43 inch screen at a wall in 3m distance. Although 43 inches do not seem bad, they are not enough for environments with close ranges, because it means a screen in 30cm distance is only 4,3 inches (about 11cm) big. This limitation does not affect the correctness or performance of the software in any form but it negatively affects the experience of the user.

6.2 Touch Detection

Touch detection solely based on computer vision with the camera near the eyes remains a difficult topic. The quality of the custom created algorithms is sufficiently high. The main limitations lie in the physical setup of the device available for this study. To improve the user input, the system could be extended with completely different approaches. Two conceivable extensions would be, for example, touch detection through gloves with sensors, or gesture based interaction. More detailed aspects will be considered in the following subsections.

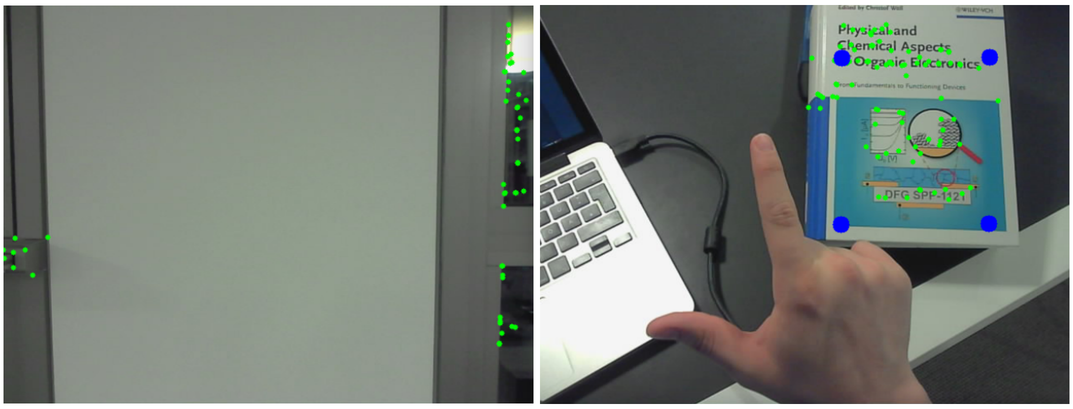
6.2.1 Covering

One problem encountered in the present project was the positioning of the camera near the eyes. As the touch detection is based on the detection of a finger covering, problems can occur. For example, if the finger is pointed straight and away from the user, it might be partially covered by the hand. As a consequence the best results are achieved if the finger is stretched out and can be seen by the camera.
This is a limiting factor because the user cannot use the touch detection like normal touch screens. In addition, the position and shape of the hand might feel strange.
As this is a physical problem, there is no easy solution with the current hardware setup. One possible extension is a second camera which would only film the hands from a higher position.

6.2.2 Camera Precision

As already mentioned in the above subsection 6.1.1 the camera's low level of precision leads to a similar lack of precision of the touch detection component. For example, a touch event is detected even if the finger is about one centimeter away. As mentioned before, the only way to enhance this behavior would be a camera with more precision or a completely different setup.

As explained in subsection 5.5, the camera movement estimation, which is done through optical flow calculation, uses the Lucas Kanade feature tracking algorithm. There are some limitations that happen with all feature-based approaches as well as some individual problems particular for this approach.



(a) White Wall with not enough Features **(b)** Textured Surface with many Features

Figure 15: Examples for Poor and Good Surfaces

First of all, if something is based on features, features are needed. This means the system cannot be used in a setup without any features. For example, if the user is in front of a blank white wall (Figure 15a), the system will not find any good feature, because there are none. The feature tracker operates best if many features are available. When for example the user sits at a desk with a cup or pens in front of him, no problems occur.
There are some options the extend the feature tracking mechanism and improve the experience. A simple way is to provide more features through objects, or for example use a textured magazine or book cover as screen (Figure 15b). Another option would be to extend the whole system by additional hardware like a sensor that measures the user's head tilt and rotation.
One problem that especially occurs with this implementation, is that mistakes add up. This occurs because one head movement estimation is based on the previous one. If one estimation is not perfect, because of rounding errors or not enough features, this will also affect the following estimations. The result is that sometimes the anchoring is not perfect and the screen moves a little bit.
This problem could be addressed in several ways. Another algorithm might not have these problems. It might also be possible to improve the current algorithm by using the original frame, where the screen was detected.

7 Conclusion and Outlook

7.1 Conclusion

The main motivation behind this project was the improvement of the so far unsatisfactory ways of content interactions in set-ups with head-mounted displays and to solve the problem of how and where to present two-dimensional content in augmented reality scenarios.
The solution proposed in this thesis successfully addresses these problems in form of a prototype system. The newly developed system uses a head-mounted device containing a time-of-flight camera and augmented reality glasses together with especially developed software.
A new way of content interaction was developed through the use of a depth camera. This solution allows direct manipulation by detecting touch input on a virtual screen. The software uses only the distance measurements of the mentioned camera to successfully detect and track fingers, as well as detecting if a fingertip touches a surface.
The question about how and where to place two-dimensional content was solved by placing virtual screens in the user's field of view. A wide range of content is applicable through the use of HTML. The solution presented here offers the freedom to choose any plane surface in the close environment as a screen, simply by turning the head towards it and triggering the detection.
Additionally this work has achieved a seamless and natural integration of a screen into the real world, by attaching it to a surface. The position of the screen is tracked by estimating the head movement with the use of optical flow algorithms.

The software is meant to be a basis for further development and as prototype for testing augmented reality with touch based interaction. The four individual main modules work reliably on their own and can easily be exchanged or integrated into other projects.

The limitations mentioned in section 6 are mainly due the hardware, which is still in an early stage of development and far from ideal. Apart from the limitations caused by the hardware, the system presented in this thesis represents a major step forward. The way they are combined represent an important new contribution to this exiting new area of research.

7.2 Outlook

The opportunities for augmented reality seem endless. It is likely to improve the everyday lives of people in ways that are not imaginable yet. An impression of what can be expected allows the promotion movie of Microsofts HoloLens[14]. A deep integration of augmented reality into the user's life is shown. Some of the presented applications are very futuristic but others might be possible sooner as thought.

Near Future

At first, applications that are not coupled to the real world are likely to emerge. For example, content that is not anchored in the world, but stays at the same position on the screen. Some examples from the promotion movie are displaying the current weather, new messages and video calls.
Before a pleasant experience is possible and such applications might happen, the hardware has to be improved significantly. From the viewpoint of computer science, the needed content generation even from a smartphone presents no problem. But suitable ways of interaction are still needed, as the present work has illustrated.

Distant Future

Detecting and classifying objects as well as the current situation of the user are still major problems in computer science and actively researched by both academia and companies. Therefore applications that rely on the real world are not likely to emerge soon.
Although there are many concepts and futuristic ideas, for example in movies such as Minority Report, Iron Man or Terminator. Probably the best impression gives Sight[15], a short movie by Eran May Raz and Daniel Lazo. In this controversial piece of art, no conventional screens are used at all. Nevertheless, the world is packed with digital information that is solely perceived through augmented reality.
Even if it is a long way to go until such a scenario might become reality, the presented approach to use surfaces as screens together with its limitations and possible extensions paves the way for further research.

[14] http://www.microsoft.com/microsoft-hololens/en-us
[15] https://vimeo.com/46304267

References

[1] Ankur Agarwal, Shahram Izadi, Manmohan Chandraker, and Andrew Blake. High precision multi-touch sensing on surfaces using overhead cameras. *Tabletop 2007 - 2nd Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems*, pages 197–200, 2007.

[2] Ronald Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6:355–385, 1997.

[3] R C Bolles and M a Fischler. A RANSAC-based approach to model fitting and its application to finding cylinders in range data. *InternationalJoint Conference onArtificial Intelligence*, pages 637–643, 1981.

[4] Jean-Yves Bouguet. Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker—Description of the algorithm. *Intel Corporation*, 5:1–10, 2001.

[5] Volkert Buchmann, Stephen Violich, Mark Billinghurst, and Andy Cockburn. FingARtips: gesture based direct manipulation in Augmented Reality. *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 1(212):212–221, 2004.

[6] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 1986.

[7] Niloofar Dezfuli, Mohammadreza Khalilbeigi, Jochen Huber, Florian Müller, and Max Mühlhäuser. PalmRC: Imaginary Palm-based Remote Control for Eyes-free Television Interaction. *Proceedings of the 10th European conference on Interactive tv and video - EuroiTV '12*, pages 27–34, 2012.

[8] Sean Gustafson, Christian Holz, and Patrick Baudisch. Imaginary Phone: Learning Imaginary Interfaces by Transferring Spatial Memory from a Familiar Device. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, pages 283–292, 2011.

[9] Chris. Harris and Mike. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 147–151, 1988.

[10] Harrison, H Benko, and a Wilson. OmniTouch: wearable multitouch interaction everywhere. *Proceedings of UIST 2011*, pages 441–450, 2011.

[11] Chris Harrison, Desney Tan, and Dan Morris. Skinput: Appropriating the Body as an Input Surface. *Acm Sigchi*, 3:453–462, 2010.

[12] Berthold K.P. Horn and Brian.G. Schunck. Determining optical flow. In *Technical Symposium East*, pages 319–331, 1981.

[13] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(1), 1979.

[14] Julien Letessier and François Bérard. Visual tracking of bare fingers for interactive surfaces. *Proceedings of the 17th annual ACM symposium on User interface software and technology*, 6:119–122, 2004.

[15] Larry Li. Time-of-Flight Camera–An Introduction. 2014.

[16] Bruce D Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. *Imaging*, 130:674–679, 1981.

[17] B S Manjunath, C Shekhar, and R Chellappa. A new approach to image feature detection with applications. *Pattern Recognition*, 29(6989):627–640, 1996.

[18] Paul Milgram and Fumio Kishino. Taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, E77-D(12):1321–1329, 1994.

[19] Pranav Mistry and Pattie Maes. SixthSense: A Wearable Gestural Interface. In *ACM SIGGRAPH ASIA 2009 Sketches on - SIGGRAPH ASIA '09*, page 1, 2009.

[20] Pranav Mistry, Pattie Maes, and Liyan Chang. WUW - Wear Ur World - A Wearable Gestural Interface. *Association for Computing Machinery / Special Interest Group on Computer-Human Interaction (ACM SIGCHI)*, 2009.

[21] Takehiro Niikura, Y Watanabe, and M Ishikawa. Anywhere surface touch: utilizing any surface as an input area. *Proceedings of the 5th Augmented . . .*, 2014.

[22] Takehiro Niikura, Yoshihiro Watanabe, Takashi Komuro, and Masatoshi Ishikawa. In-Air finger motion interface for mobile devices with vibration feedback. *IEEJ Transactions on Electrical and Electronic Engineering*, page 4503, 2014.

[23] John A Roese and Lawrence E Mccleary. Stereoscopic Computer Graphics for Simulation and Modeling. In *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*, pages 41–47, 1979.

[24] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.

[25] Jianbo Shi and Carlo Tomasi. Good Features to Track. In *IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, 1994.

[26] AD Wilson. TouchLight: an imaging touch screen and display for gesture-based interaction. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76. ACM Press, 2004.