A Low-cost Tracking Solution For VR Headsets

BSc Thesis

T. A. AMENT (4080300)
M. I. KROES (4112075)

June 19, 2017

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE

ELECTRICAL ENGINEERING PROGRAMME

Supervisors: Prof. Dr. E. Eisemann
Dr. M. Stengel

Thesis committee: Prof. Dr. E. Eisemann, TU Delft, supervisor
Dr. ir. I. E. Lager, TU Delft, chair
Dr. A. Endo, TU Delft, jury

Abstract

Inaccurate and slow positional tracking are fundamental problems of low-cost VR solutions. These factors can induce motion sickness and therefore significantly deteriorate the user experience. There are already a variety of solutions available to track the position of a VR HMD, some of which can be made robust and low-latency by inexpensive means.

This thesis proposes a low-cost, outside-in, optical tracking solution. As optical sensor a stationary camera with an IR pass filter is used to capture an array of IR LEDs, which are integrated at strategic locations within the frame of the HMD. By using the OpenCV library the 3D pose of the LEDs can be estimated, and used to change the perspective of the user within the virtual world.

In particular, this tracking solution incorporates a fast, cost-effective, modified camera, capable of capturing over 180 frames per second. With this camera and a custom designed polyhedral structure to incorporate the LEDs, the LED array can be calibrated quickly and tracked accurately even when oriented at an obtuse angle with the camera. The high frame rate also allows for a redundancy in pose estimations, obtained by a mix of algebraic and iterative PnP solvers to further improve numerical stability and accuracy.

The tracking algorithm accurately tracks the developed prototype at short distances, even when placed in a noisy environment. The calibration procedure can also be performed fast, so that tracking can be resumed within a second when tracking has been lost due to occlusion. Recommendations for further improvements would be to integrate an IMU to provide tracking information in case that the HMD becomes occluded, and to overall improve tracking of rotational movement. Also, a higher resolution optical sensor should be used to improve the range of this tracking solution.

Preface

This thesis was written as part of the BSc Graduation Project of Electrical Engineering at the faculty of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology. In this document the research and the design process for developing a low-cost and low latency tracking solution for virtual reality headsets will be described.

We would like to thank Dr. Michael Stengel and Prof. Elmar Eisemann for their guidance and support throughout the project, and for sharing their knowledge on computer vision. We would also like to thank Dr. Jaap Hoekstra for providing us with the tools and electronics to build our test setups.

Tjalling and Mairin
Delft, June 2017

Contents

Chapter 1

Introduction

For a deeply immersive VR (Virtual Reality) experience, fast and accurate positional tracking is essential. Many different tracking systems and algorithms exist, but the technology is still in its infancy. Moreover, many of the more robust solutions are available only at a high cost, which prevents market penetration. VR is therefore often relegated to being merely novelty technology [1]. The use cases for virtual reality are numerous however, spanning from entertainment purposes, like virtual tourism and movies, to professional applications like architecture, automotive design and surgery training.

The main technique to track the position of a user wearing an HMD (Head Mounted Display) relies on optically sensing a set of markers and deriving a so called 3D pose out of the captured 2D images. See figure 1.1 for an image of the VR HMD, designed by Dr. Michael Stengel [2], that will incorporate the finalized low-cost tracking solution.

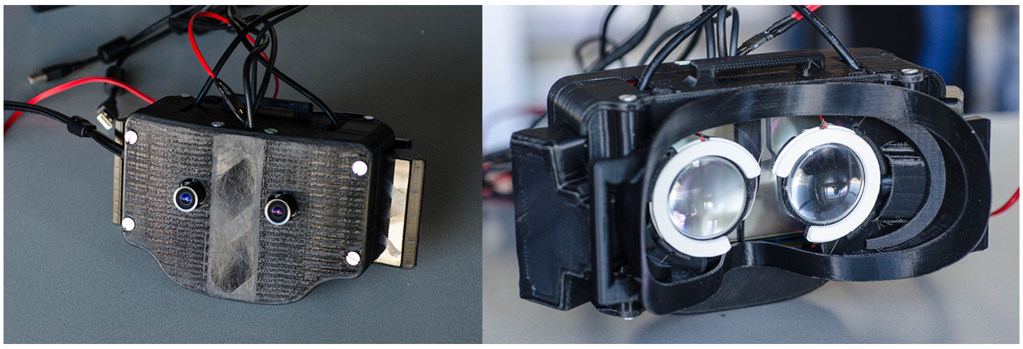


Figure 1.1: A custom designed and 3D printed HMD, viewed from the front (left) and from the back (right)

For optical positional tracking there are two main options available. There is inside-out tracking, where a camera is located on the HMD and visible markers are distributed across a room. The other option is outside-in tracking where the camera is stationary and the markers are located on the HMD. Seeing that the primary objective is to lower costs, latency and increase accessibility, the latter option is more viable since it allows for faster communication between the camera and stationary processing unit and is more flexible to configure.

1.1 Problem Definition

There are still quite some challenges when it comes to optical tracking. As mentioned before, the technique relies on tracking visual markers in a captured 2D image and reconstructing a 3D pose out of it. These markers need to possess certain distinguishable features compared to other visual elements in the image, and its position needs to be tracked frame-by-frame. The following problems arise from this.

The system may not be able to keep track of the markers, and lose the position when the markers move too much in between frames. It might also stop detecting certain markers completely if they are not within LoS (Line of Sight), or the system might track noise if the features are not sufficiently distinct from other elements or artefacts within the image. In the case where the amount of LEDs that are still being accurately tracked are insufficient to be able to derive a pose, some LEDs will need to be found again with a calibration procedure that is both reliable and fast. Finally, distortions caused by imperfections in the camera lens and misalignment of the lens relative to the image sensor are in general also not negligible and will be detrimental to tracking accuracy.

Even when the markers are found and are tracked correctly, there still remains the fact that the algorithm to convert these 2D positions to 3D poses can be inaccurate. These outliers will need to be corrected and preferably prevented where possible, since inaccurate pose reconstruction will lead to a very unpleasant and disturbing VR experience.

To correct for the problems related to optical tracking described above, we will study how to properly calibrate the camera to correct for these lens distortions, how to design a system of visual markers that can be reliably and quickly calibrated, while still being capable of being properly integrated within an HMD, and lastly how to use this array of markers to eventually build an accurate algorithm to reconstruct the 3D pose.

1.2 Thesis Structure

This thesis is structured as follows:

1. Program of Requirements
2. Background Theory
3. Design
4. Validation
5. Conclusions & Recommendations

First, we will list a set of essential requirements the designed product must adhere to. In the following chapter we will provide the reader with some theoretical background that will be necessary to obtain a profound understanding of some of the concepts and principles behind the technology incorporated in the product, as well as to understand why certain design choices were made. After discussing some theory we will extensively describe our design and elaborate on some of the most important design choices. In the subsequent chapter we will describe the tests we conducted and their results to verify whether the system meets the program of requirements. Finally, we will conclude our thesis with a reflection on the overall design and provide some recommendations for future work.

Chapter 2

Program of Requirements

This developed tracking solution is intended to be used as a cost-effective, and open-source means for developers to create their own VR HMDs. The typical use case would be indoors, connected to a well equipped desktop PC. The main PC specifications for the tracking algorithm are based on the recommended specifications for the Oculus Rift [3] and can be found in table 2.1.

CPU:	Intel i5-4590 / AMD Ryzen 5 1500X or greater
Graphics Card:	NVIDIA GTX 970 / AMD Radeon R9 290 or greater
RAM:	8GB+
OS:	Windows 7 64-bit or newer

Table 2.1: Recommended PC specifications for achieving optimal tracking performance

1. **Functional requirements**

   Since the system should track an end user who is wearing an HMD, the following requirements are listed to provide a comfortable user experience.

   1.1. *The system must be capable of tracking a single HMD*

   The system is only intended for the use of a single person. As of such, only a single worn HMD needs to be tracked.

   1.2. *The system must be capable of tracking user movement in 6 Degrees of Freedom*

   To allow for a natural VR experience, the end user should be able to move their head in a way that is intuitive without losing connection to the virtual world.

   1.3. *The LED array must be within Line-of-Sight of the camera*

   Since this tracking solution relies on optically sensing a set of markers, no position can be tracked if these markers are occluded or simply turned away from the camera. This requirement also restricts user movement to the viewing frustum of the camera.

2. **Ecological embedding in the environment**

   While functional requirements are of the essence for a good design, it is important to also prevent negative impact on the environment. To comply with this, the necessary requirements need to be drawn.

   2.1. *The LED array must not transmit light within the visible spectrum*

   To be less sensitive to background noise, as well as to provide a solution that is less intrusive on its direct environment, it is important to limit emission of light to wavelengths outside of the visible spectrum.

   2.2. *The system must indicate when tracking is lost, so that the user can be disconnected from the virtual world gracefully*

   Since the loss of positional tracking will result in the user being unable to change perspective regardless of movement of the head, the user will be subject to unpleasant dissociative sensations. To prevent this, the system should send a signal that tracking is lost, so that the developer of the rendering framework can anticipate on this (i.e. fade out the image projected to the HMD).

3. **System requirements**

   To make it possible for the entire design to meet the projected demands, the system needs to meet a set of specific requirements.

   3.1. *The system must be capable of correcting for distortions caused by the camera lens*

   Lenses have certain imperfections that will be detrimental to the overall performance of the system. A reliable calibration method will need to be implemented to correct these distortions to improve tracking accuracy.

   3.2. *The system must be capable of distinguishing the LED array from ambient light and internal lens reflections*

   The tracking algorithm needs to estimate the head pose of the connected user. If it is sensitive to artefacts within the image, like noise and reflections, an incorrect head pose will be estimated. The identification process of the LEDs should therefore be built robust enough to prevent unintended labeling of image artefacts.

   3.3. *The tracking algorithm must not exceed 5ms of runtime per frame on PCs that meet the recommended specifications in table 2.1*

   In order to enable the tracking algorithm to derive a pose out of each frame captured by the camera, the algorithm should be fast enough to process the frame before the next one is captured. Since the sustained maximum frame rate of the camera is 180 frames per second, the algorithm can only be allowed to have a runtime of up to $\frac{1}{180Hz} \approx$ 5.6 ms.

4. **Development of manufacturing methodologies**

   The following requirements were imposed on the development of the design to ensure a fully open-source foundation.

   4.1. *The optical tracking will be implemented in C++ with the open-source OpenCV 3.2 library*

   4.2. *The LED array will be controlled by a Teensy micro-controller programmed in C*

5. **Business strategies, marketing and sales opportunities**

   This single criterion is necessary to protect the economic value component of the design

   5.1. *The total sum of the costs of the components used in the tracking solution should not exceed €50,-*

   In order to be a viable alternative to existing technology, the low cost of this particular solution is an important requirement. As mentioned in the introduction, the cost of positional tracking solutions is a fundamental problem for the adoption of VR technology. €50,- is well below the price of existing solutions [4], especially considering the fact that this price includes the LED structure rather than exclusively the optical sensor.

Chapter 3

Background Theory

In this chapter we will provide a theoretic background to get a solid understanding of some of the fundamental concepts and principles we have to consider within the context of this product. In the first paragraph we will investigate how we can obtain an accurate mathematical model for the camera. Finally, we will look into the concept of 3D Pose reconstruction.

3.1 Camera Calibration

The images captured with the camera contain information regarding the position of the LEDs measured in pixels. To translate this data into real-world coordinates, it is necessary to obtain a model for the camera that maps the 3D world space to the 2D image space. It would be naive, however, to extract the 2D coordinates of the LEDs directly from the captured image. Imperfections in the camera lens and misalignment of the imaging sensor introduce non-negligible distortions within certain areas of the image. The retrieved positions of the LEDs will therefore be inaccurate if these aren't corrected.

First, the camera model with the so called camera matrix containing the intrinsic camera parameters will be discussed. This camera matrix mathematically describes how the coordinates of a 3D object are mapped onto a 2D plane (camera sensor). Next, we will inspect sensor misalignment and the various kinds of lens distortions that are common within most cameras and elaborate on how we can model these phenomena. Finally, the concept of planar homography will be introduced, which is another mathematical tool that will enable us to derive these camera parameters.

3.1.1 Camera Model

The most simplified model for projecting a 3D object to a 2D plane is the pinhole camera model and is depicted in figure 3.1. In this model no lens is used to bend light rays, and all rays that hit the surface of the image plane pass through the focal point. The resulting image is one that is inverted, but for mathematical modeling the plane can be placed in front of the focal point, which simplifies the model so that the projected image doesn't need to be inverted.

With this adjusted pinhole camera model the projection of a point $\vec{P} = < X, Y, Z >$ of a 3D object, to the homogeneous coordinates of a point $\vec{q} = < u, v, w >$ on the projected image can be described by equation 3.1.

$$\vec{q} = \mathbf{K}\vec{P} \tag{3.1}$$

Where **K** is the intrinsic camera matrix given by

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

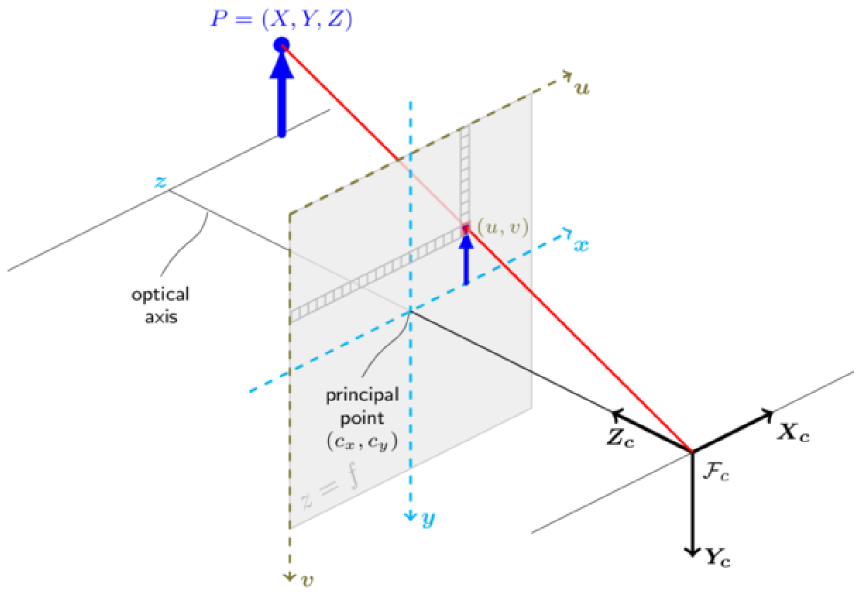


Figure 3.1: Illustration of the pinhole camera model
Source: `docs.opencv.org`

In this camera matrix the parameters f_x and f_y are the focal length of the camera lens in the x and y dimension respectively, and describe how strong the magnification is in either direction. c_x and c_y are the x and y coordinates of the principal point of the camera. The principal point is the point on the image sensor that intersects the optical axis, which aligns with the centre of the aperture of the camera. Ideally this would be right in the middle of the sensor, but due to imperfections in the fabrication process this will typically be off by a certain amount of pixels.

After applying matrix multiplication, equation 3.2 is obtained for the projective transform of points on a 3D object to the pixels on the image.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} f_x X + c_x Z \\ f_y Y + c_y Z \\ Z \end{bmatrix}$$

$$\begin{aligned} u &= f_x \frac{X}{Z} + c_x \\ v &= f_y \frac{Y}{Z} + c_y \end{aligned} \tag{3.2}$$

In the last step the homogenizing transformation $< u, v, w > = < \frac{u}{w}, \frac{v}{w}, 1 >$ is applied [5]. As will be seen in the next section, this model for the camera projection is not complete for our purposes, since it models a perfect ”lens” (pinhole) without distortions.

3.1.2 Image Distortions

In contrast to our virtual camera model, real cameras need lenses to collect more light by bending the rays in order to increase the exposure of the image sensor. Real lenses are bound by physical limitations and manufacturing errors. Due to non-ideal lens shaping some rays will not be bent uniformly, which will also result in a variance in magnification within different sections of the image.

Typically for small, wide angle lenses, light is bent more strongly towards the fringes of the lens. This will result in a disproportional decrease in focal length, and thus lower magnification near the edges of the image. Distortion of this type is referred to as radial distortion, since the magnification of the lens falls off proportionally to the radial distance to the optical centre of the lens. Radial distortion can be modelled by the Taylor expansion around r = 0 as shown in equation 3.3 [6].

$$\varepsilon(r) = \sum_{n=0}^{\infty} \alpha_n r^n = \alpha_0 + \alpha_1 r + \alpha_2 r^2 + \dots \tag{3.3}$$

For most lenses, the first two even-order terms of this polynomial can be used to correct for radial distortion, with the third term for additional precision [7]. Now the corrected positions can be found as described in equation 3.4.

$$\begin{aligned} u' &= u\varepsilon(r) + u \approx u(1 + \alpha_2 r^2 + \alpha_4 r^4 + \alpha_6 r^6) \\ v' &= v\varepsilon(r) + v \approx v(1 + \alpha_2 r^2 + \alpha_4 r^4 + \alpha_6 r^6) \end{aligned} \tag{3.4}$$

Another type of distortion, that we need to correct for in our image, is caused by sensors that may be assembled at a skewed angle relative to the lens, rather than being perfectly parallel. This will cause so called tangential distortions. The corrected coordinates for this type of distortion can be found by equation 3.5.

$$\begin{aligned} u' &\approx u + [2\beta_1 uv + \beta_2(r^2 + 2u^2)] \\ v' &\approx v + [\beta_1(r^2 + 2v^2) + 2\beta_2 uv] \end{aligned} \tag{3.5}$$

As can be seen from the descriptions for the corrected image coordinates, there are 5 distortion coefficients we need to solve for: α_2, α_4, α_6, β_1 and β_2. In literature these are commonly referred to as k_1, k_2, k_3, p_1 and p_2, and we will also continue to do so from this point on. The next section will describe how we can set up a system of equations to solve for these distortion parameters, as well as the camera intrinsic parameters.

3.1.3 Planar Homography

In sections 3.1.1 and 3.1.2 we gathered that in order to get an accurate mathematical description for our camera, we need to solve for 4 camera intrinsic parameters: f_x, f_y, c_x and c_y, and at least 5 lens distortion parameters: k_1, k_2, k_3, p_1 and p_2. One method for setting up a system of equations that can solve for these parameters is called planar homography.

Planar homography can be described as the projective operation that relates points of one plane to those of another. In our case, it would be the mapping of measured positions on a planar object, to the pixels in the image sensor of our camera. This projection has an extra operation compared to the projective transform in 3.1 however. The object points first need to be rotated and translated to the reference frame of our camera, before we can apply the projection from the lens to the sensor. This operation is modelled by the extrinsic camera matrix:

$$\mathbf{W} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix}$$

The elements r_{ij} are the rotation parameters, and t_x, t_y and t_z are the translations in the x, y and z directions respectively. The full equation for the planar homography can then be formulated as:

$$\vec{q} = s\mathbf{KW}\vec{P} \tag{3.6}$$

Where $\vec{q}$ is once again the position of a point on the image sensor, $\mathbf{K}$ the intrinsic camera matrix, $\mathbf{W}$ the extrinsic camera matrix and $\vec{P}$ the position of a point on the captured planar object. The homography matrix $\mathbf{H}$ can therefore be described by equation 3.7. The variable s is a scalar that has intentionally been factored out of the homography matrix to explicitly denote the scale up to which the homography is defined.

$$\mathbf{H} = \mathbf{KW} = \mathbf{K}\begin{bmatrix}\vec{r_1} & \vec{r_2} & \vec{t}\end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \tag{3.7}$$

As can be seen in equation 3.7, the resulting homography matrix will be a 3×3 matrix. Since the last element of the homography matrix contains the homogeneous coordinate, there remain 8 free parameters to solve for. Fortunately, each of the four vertices of the plane consist out of two coordinates, giving us 8 equations to obtain these parameters.

These 8, linearly independent equations, allow us to determine the 3 Euler angles described by the rotation vector, the 3 translations in the x, y and z dimensions, as well as 2 of the intrinsic parameters in the camera matrix. An additional frame capture from a different view can be used to retrieve the other 2 intrinsic parameters, since these remain static across frames. The new frame again introduces 6 unknown extrinsic parameters, so in theory this should be exactly enough to solve for the last 2 intrinsic parameters. However, to correct for noisy measurements and numerical instabilities, multiple images can be captured to iteratively prime the intrinsic parameters.

Moreover, we initially still assume the lens to be perfect when deriving these parameters. By capturing multiple frames, we can also derive the distortion parameters by correcting the errors caused by the distortion iteratively, and adjusting the other parameters accordingly.

Per frame the homography matrix can be constructed by finding a matrix **H** that satisfies the following:

$$\begin{aligned} \vec{q} &= s\mathbf{H}\vec{P} \\ \vec{P} &= \frac{1}{s}\mathbf{H}^{-1}\vec{q} \end{aligned}$$

Then, by applying equation 3.7, we obtain an expression for each of the columns of the homography matrix.

$$\begin{aligned} \vec{h_1} &= s\mathbf{K}\vec{r_1} \\ \vec{h_2} &= s\mathbf{K}\vec{r_2} \\ \vec{h_3} &= s\mathbf{K}\vec{t} \end{aligned}$$

Which when rewritten provides us with the descriptions for the extrinsic parameters given by 3.8.

$$\begin{aligned} \vec{r_1} &= \frac{1}{s}\mathbf{K}^{-1}\vec{h_1} \\ \vec{r_2} &= \frac{1}{s}\mathbf{K}^{-1}\vec{h_2} \\ \vec{r_3} &= \vec{r_1} \times \vec{r_2} \\ \vec{t} &= \frac{1}{s}\mathbf{K}^{-1}\vec{h_3} \end{aligned} \tag{3.8}$$

Since $\vec{r_1}$ and $\vec{r_2}$ describe rotations around perpendicular axes, we find that these vectors are indeed orthogonal and that equation 3.9 must hold.

$$\vec{r_1} \cdot \vec{r_2} = \vec{r_2}^T \vec{r_1} = 0 \tag{3.9}$$

Furthermore, since we factored the scaling factor out of the homography matrix, we find that $\vec{r_1}$ and $\vec{r_2}$ are of unit length and therefore in fact also orthonormal. This means that equation 3.10 must also hold.

$$\begin{aligned} ||\vec{r_1}|| &= ||\vec{r_2}|| = 1 \\ \vec{r_1}^T\vec{r_1} &= \vec{r_2}^T\vec{r_2} \end{aligned} \tag{3.10}$$

Substituting 3.8 in 3.9 & 3.10 and omitting the scaling factor for now, we get:

$$\vec{h_1}^T(\mathbf{K}^{-1})^T\mathbf{K}^{-1}\vec{h_2} = \vec{0}$$

$$\vec{h_1}^T(\mathbf{K}^{-1})^T\mathbf{K}^{-1}\vec{h_1} - \vec{h_2}^T(\mathbf{K}^{-1})^T\mathbf{K}^{-1}\vec{h_2} = \vec{0}$$

If we refer to the recurring product $(\mathbf{K}^{-1})^T\mathbf{K}^{-1}$ as $\mathbf{B}$ and inspect it, we find that the resulting matrix is given by:

$$\mathbf{B} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{f_x^2} & 0 & \frac{-c_x}{f_x^2} \\ 0 & \frac{1}{f_y^2} & \frac{-c_y}{f_y^2} \\ \frac{-c_x}{f_x^2} & \frac{-c_y}{f_y^2} & \left(\frac{-c_x}{f_x^2} + \frac{-c_y}{f_y^2} + 1\right) \end{bmatrix}$$

Substituting $\mathbf{B}$ gives us equations 3.11 & 3.12.

$$\vec{h_1}^T\mathbf{B}\vec{h_2} = \vec{0} \tag{3.11}$$

$$\vec{h_1}^T\mathbf{B}\vec{h_1} - \vec{h_2}^T\mathbf{B}\vec{h_2} = \vec{0} \tag{3.12}$$

Since we obtain a pair of these equations for each derived homography, we indeed need at least two captured views to solve for all four intrinsic parameters. When we have at least two frames, we can obtain the intrinsic parameters given by formulae 3.13, 3.14, 3.15 & 3.16.

$$f_x = \sqrt{\frac{1}{sB_{11}}} \tag{3.13}$$

$$f_y = \sqrt{\frac{B_{11}}{sB_{11}B_{22} - sB_{12}^2}} \tag{3.14}$$

$$c_x = sB_{13}f_x^2 \tag{3.15}$$

$$c_y = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \tag{3.16}$$

Where s is the scaling factor that was extracted out when applying the orthonormality properties. Its value is given by equation 3.17.

$$s = \frac{B_{11}}{B_{11}B_{33} - B_{13}^2 + c_y(B_{12}B_{13} - B_{11} - B_{23})} \tag{3.17}$$

In section 3.1.2 we derived models for tangential and radial lens distortion. The complete description for the undistorted coordinates u' and v' are given by equation 3.18

$$\begin{aligned} u' &\approx u(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1uv + p_2(r^2 + 2u^2) \\ v' &\approx v(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2v^2) + 2p_2uv \end{aligned} \tag{3.18}$$

After all of the 10 intrinsic and extrinsic parameters have been derived, the distortion coefficients in 3.18 can be obtained and applied to the image. Since the projection has now been altered, the intrinsic and extrinsic parameters will need to be readjusted iteratively to minimize the re-projection error again.

This is the very reason why, while in theory 2 images of a planar object should be sufficient to build a basic mathematical model of the camera, we require many more in practice. In the documentation of the OpenCV library in [8], a minimum of 10 captures is even recommended. The planar object that will be captured also needs to have a multitude of easily identified vertices at precisely measured locations, since there can be outliers in the corner detection process.

3.2 3D Pose Estimation

After the camera has been calibrated and a mathematical model has been derived, it is possible to derive a so called pose of a captured object towards the camera. This pose reconstruction forms the backbone of optical positional tracking. In this section we will discuss what this pose is, and how it can be reconstructed using perspective-n-point estimation.

3.2.1 Perspective-n-point estimation

Consider the camera model that was derived for planar homography in equation 3.7. The pose of an object can be defined as the translation and rotation operations described by matrix **W**, that are necessary to bring an object to the reference frame of the camera. An illustration of this operation can be found in figure 3.2. This pose then provides all the necessary information for the orientation and position of an object relative to the camera.

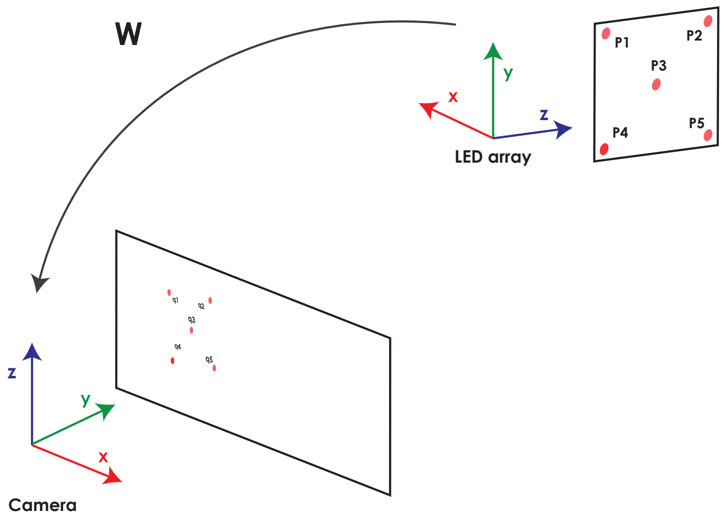


Figure 3.2: The pose **W** of a structure of points relative to the camera

The next step is to actually derive this pose by using data that is readily available. It turns out that if there is a set of n visual markers, that correspond to a certain set of n coordinates in the coordinate system of the object, and these coordinates are compared to the n positions on the image of the camera, a system of equations can be obtained that has in fact one unique solution, if certain conditions are met. First, according to [9] the condition $n > 3$ should be met to get an unambiguous pose. That is to say, the position of more than 3 visual markers need be known and visible on the image to obtain a distinct pose. Finally, according to [10] the feature points should not be collinear, since this would lead to a degenerate case where the line connecting these points would have rotational ambiguity.

This problem of finding the pose of an object that corresponds to the projection of n points on the image, based on exactly known locations within their coordinate system, is called the perspective-n-point (PnP) problem [11]. There are different methods to solve this problem. Most of these solutions are iterative, since iterations typically result in a greater amount of accuracy and numerical stability. Then there are other methods like Efficient PnP (EPnP) [12] and Unified PnP [13], that attempt to reduce computational complexity while conserving numerical stability and accuracy, by applying a combination of algebraic methods paired with iterative refinement.

For the iterative algorithms the general concept to solve the PnP problem is similar. The first step consists out of making an initial estimate of the pose $\hat{\mathbf{W}}$ that corresponds to the found image locations. Based on $\hat{\mathbf{W}}$ the coordinates of the feature points will be projected to image coordinates by applying equation 3.6 from section 3.1.3.

$$\vec{q}_{n,estimated} = s\mathbf{K}\hat{\mathbf{W}}\vec{P}_n$$

Then $\hat{\mathbf{W}}$ will be evaluated based on how well the re-projected feature points $\vec{q}_{n,estimated}$ correspond to the actual positions $\vec{q}_{n,actual}$ of the visual markers in the screen. The next phase is where the individual iterative approaches diverge in their implementations. Most algorithms do at least implement a variation of the Least Squares approach to do curve fitting. This generally consists out of updating the estimate of the pose by computing the Jacobian $\mathbf{J}$ of the perspective projection equations for each of the feature points f_n.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_x} & \frac{\partial f_1}{\partial \theta_y} & \frac{\partial f_1}{\partial \theta_z} & \frac{\partial f_1}{\partial t_x} & \frac{\partial f_1}{\partial t_y} & \frac{\partial f_1}{\partial t_z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial \theta_x} & \frac{\partial f_n}{\partial \theta_y} & \frac{\partial f_n}{\partial \theta_z} & \frac{\partial f_n}{\partial t_x} & \frac{\partial f_n}{\partial t_y} & \frac{\partial f_n}{\partial t_z} \end{bmatrix}$$

The Jacobian expresses the sensitivity of the projection with respect to each of the 6 degrees of freedom of the pose [14]. In the Least Squares algorithm, an update to the pose $d\hat{\mathbf{W}}$ is calculated by computing the pseudoinverse of the Jacobian, and multiplying it with the re-projection error $d\vec{q} = \vec{q}_{estimated} - \vec{q}_{actual}$.

$$d\hat{\mathbf{W}} = \mathbf{J}^{\dagger} d\vec{q} \tag{3.19}$$

This update is then added to the estimate, and refined iteratively until this process converges to a point where the re-projection error drops below a certain predefined tolerance threshold. There can be convergence issues and singularities in the Jacobian for certain pose estimations, so algorithms like Levenberg-Marquardt, also referred to as Damped Least Squares [15], add a damping factor to the cost function to improve numerical stability [16].

Chapter 4

Design

In this chapter the design process for the developed prototype will be described. Also, the design choices that had to be made, as well as the constraints that were applicable will be elaborated. First, a brief overview of the system will be provided, followed by a couple of paragraphs, each dedicated to one of the designed subsystems.

4.1 System Setup

A schematic overview of the system is depicted in figure 4.1. The schematic is best read from right to left. The VR HMD has a couple of IR SMD (Surface-Mounted Device) LEDs integrated within the front cover. These LEDs are controlled by an integrated micro-controller capable of initiating a sequence of calibration patterns. The LEDs are then captured by a camera with an IR pass filter applied to the front element of the lens. The captured images of the camera are then transferred in real-time to the PC, which runs the tracking algorithm, and a rendering framework that provides the perspective corrected images to the HMD via HDMI. The rendering framework and the image capture framework were provided by Michael Stengel.

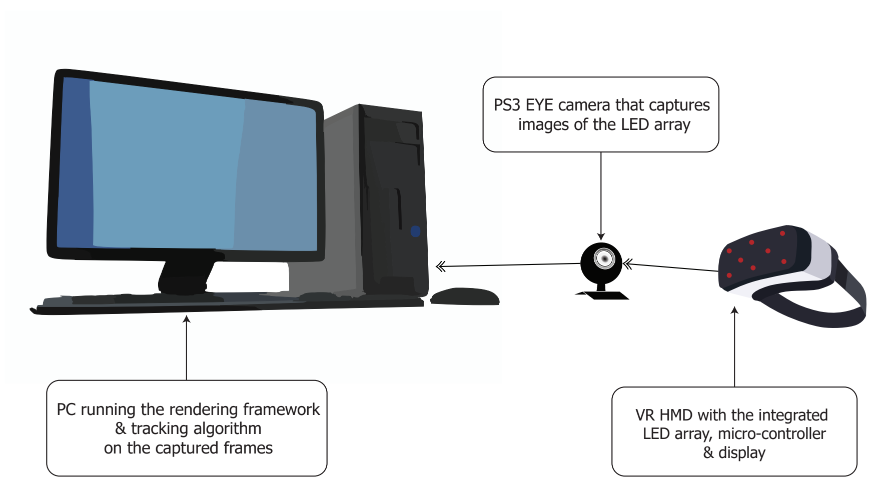

PS3 EYE camera that captures images of the LED array
PC running the rendering framework & tracking algorithm on the captured frames
VR HMD with the integrated LED array, micro-controller & display

Figure 4.1: Total setup of the developed system

4.2 Subsystems

As discussed in section 4.1, now a detailed description of the design process per subsystem will be provided. The design of the entire system was split into three subsystems. The first paragraph will describe the calibration process of the camera, the second paragraph will describe the creation of the LED array and its calibration procedure, and the last paragraph will describe the construction of the implemented tracking algorithm based on 3D pose reconstruction.

4.2.1 PS3 Eye camera

The camera that's used in this design is a PlayStation 3 Eye camera. The sensor has a resolution of 640×480 pixels (QVGA), and is capable of capturing 60 FPS (frames per second) at this resolution.

Important criteria for choosing this camera were the low cost (this camera was acquired for €7,-), and the fact that the sensor can also capture images at a resolution of 320×240 at frame rates over 180 FPS. This frame rate provides 3 times the amount of samples within the same time span, and is higher than any optical positional tracking system currently on the market [17]. The increased sampling rate also provides the means to make the tracking algorithm more robust, by being capable of detecting faster movements of the LED array, and thus losing track less often, as well as enabling an overall decrease in latency, since pose estimations can be updated more frequently.

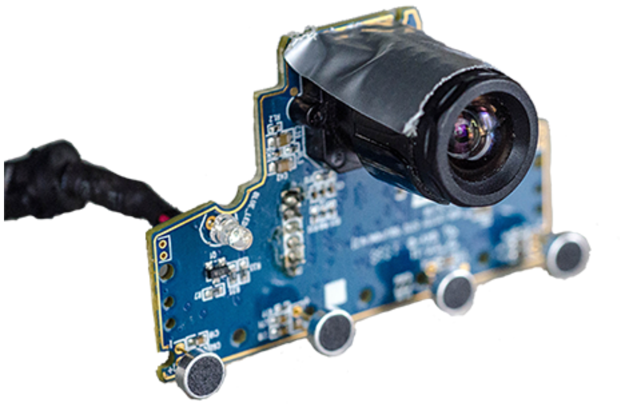


Figure 4.2: PS3 Eye camera with an IR pass filter applied to it

The camera has been modified by Dr. Michael Stengel, since by default this device, like most cameras, comes with an IR filter applied to the front element. As we are only interested in tracking the LED array, we chose to limit the camera by capturing IR light only, and thus removing the IR filter and replacing it with an IR pass filter. By only sensing light with wavelengths higher than the visible spectrum, we decrease the amount of potential sources within a room that can interfere with the tracking system significantly.

As mentioned in section 3.1, calibration of the camera is essential to obtain accurate results. The camera was calibrated following the procedure described in [18] based on OpenCV's *cv::calibrateCamera()* function. The pattern we used for calibration was a 9×6 checkerboard with 25mm×25mm squares. Checkerboard patterns provide a more robust and accurate calibration result than asymmetric circles, since the corners can be calculated by line intersections, which provides accuracy into the sub-pixel level rather than pixel level. The OpenCV function *cv::findChessboardCorners()* can then be used to detect the position of all 54 corners within the image per view. Since this is a regular pattern with exactly measured corner locations, we can solve for the intrinsic, extrinsic and distortion parameters with the use of planar homography. In total 15 captures of the checkerboard object were taken to minimize the re-projection error.

The resulting matrix containing the intrinsic camera matrix can then be used to estimate the pose, while the distortion parameters can be used to undistort the image. In order to not waste computational power, we only correct for the position of the LEDs rather than the entire image.

4.2.2 LED Array

The next subsystem in the design is the LED array. As mentioned in previous sections, the LEDs transmit IR light and must be programmable to support calibration routines. The device that was used to control the LEDs is an Arduino compatible Teensy 3.6 micro-controller [19]. This controller has a couple of advantages over the Arduino: it's faster and has a much smaller footprint, although the cost is higher than that of the Arduino Uno (€33,50 vs €20,-). The prototype was developed with an Arduino Uno however.

The LEDs used in this design are Harvatek HT-191IRAJ SMD LEDs that emit light at 940nm [20]. The main features that we based our choice on, is their low cost (€0,20) and small size. The designed prototype has 16 of these LEDs integrated at predefined locations in subgroups of 4 LEDs per tetrahedron. The prototype models the front cover of the HMD as tracked object, and a picture of the model can be found in figure 4.3. The coordinates of the LEDs within this structure can be found in figure 4.4. This prototype has been designed with the help of Dr. Michael Stengel.

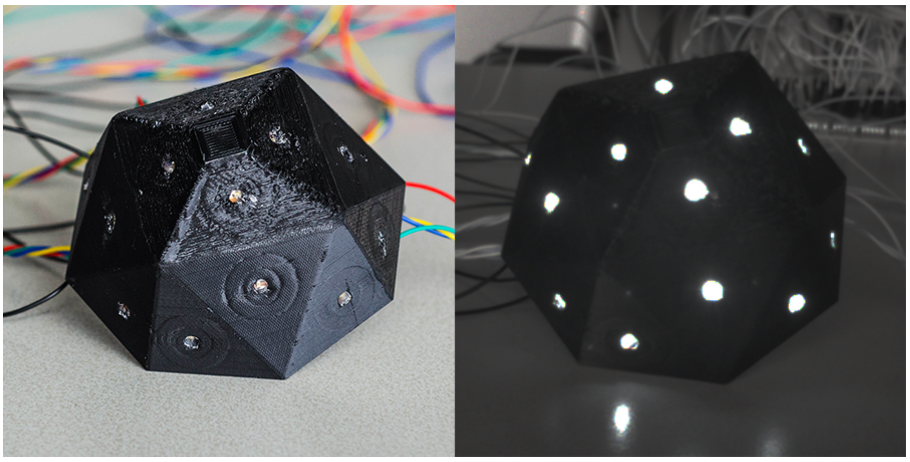
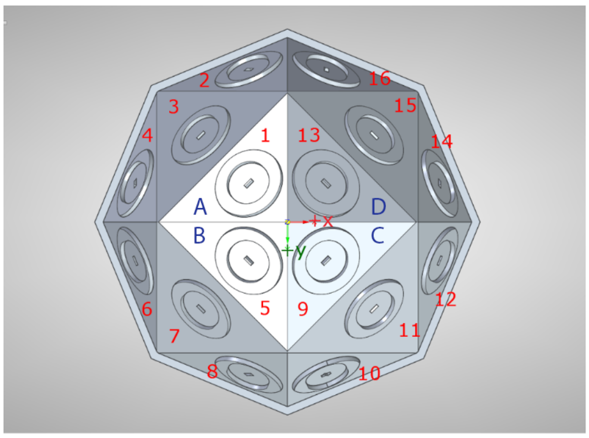


Figure 4.3: Picture of the prototype captured with an IR filter (left) and with an IR pass filter (right)

In this design the aim was to create a structure in which at any azimuthal angle φ between -90° and 90°, at least 4 LEDs on the structure are visible. Moreover, when these LEDs are visible, the distance between adjacent LEDs should be kept as large as possible, so that they can still be distinguished from each other



```
1  = [-a, -a,  b]   a = 8.37
2  = [-a, -b,  a]   b = 32.48
3  = [-c, -d,  d]   c = 18.49
4  = [-b, -a,  a]   d = 18.55
5  = [-a,  a,  b]
6  = [-b,  a,  a]
7  = [-c,  d,  d]
8  = [-a,  b,  a]
9  = [a,   a,  b]
10 = [a,   b,  a]
11 = [c,   d,  d]
12 = [b,   a,  a]
13 = [a,  -a,  b]
14 = [b,  -a,  a]
15 = [c,  -d,  d]
16 = [a,   b,  a]
```

Figure 4.4: The labels of the LEDs and their positions within the structure in mm

Since the system needs to drive 16 LEDs, the amount of output pins of the micro-controller and their current carrying capacity will become a limitation. To alleviate this limitation, an LED control circuit was designed based on the concept laid out in [21]. A schematic for the modified circuit can be found in figure 4.5.

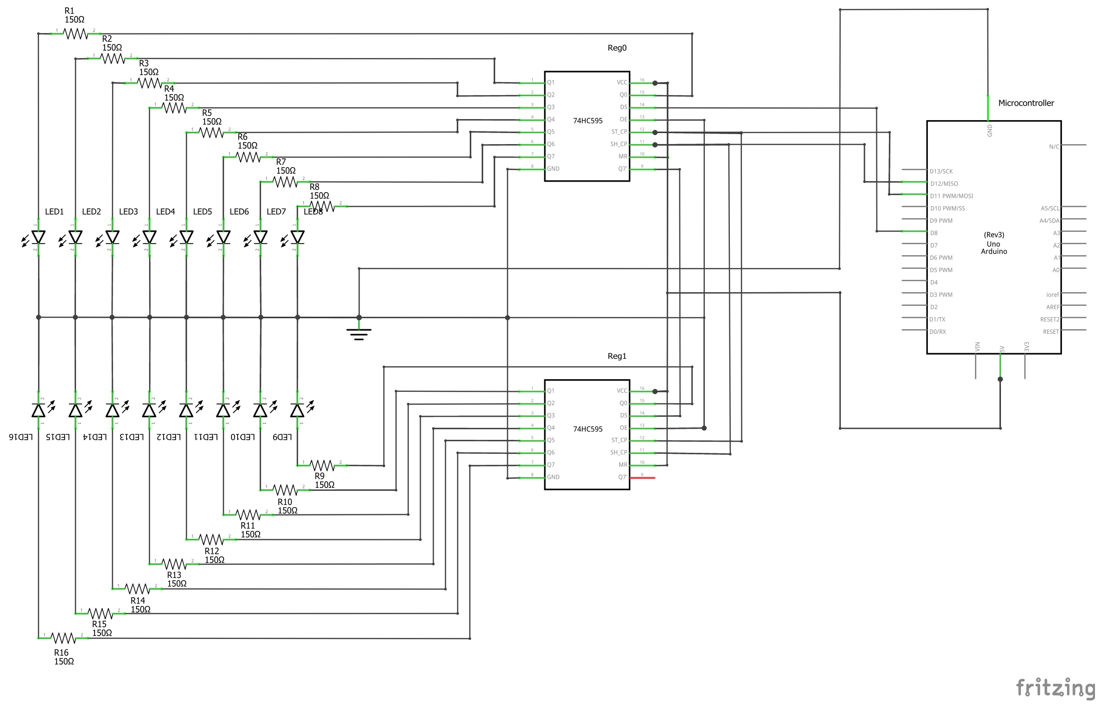


Figure 4.5: Schematic of the circuit that drives the LEDs

The output pins of the Arduino are extended by using a couple of daisy chained Nexperia 74HC595N shift registers [22]. These shift registers read in a serial stream of data synchronous to a supplied clock signal, and store up to 8 bits of data. The data within the shift register can then be latched into a storage register by sending a strobe signal to the IC. The data within the storage register can, on its turn, be sent in parallel on the output pins by providing a high signal on the output enable pin. The shift registers also have a serial output, which in the case that more than 8 bits are sent before latching the data into the storage register, supplies the other shift register down the chain with the overflow of data. By sending two bytes in series to the shift registers, 16 LEDs can now be controlled by using only 3 output pins, +5V and GND on the micro-controller.

An important reason for using the 74HC595N, is the fact that this IC can source a relatively high amount of current on its parallel output pins, since these are rated at currents up to 35mA. Most data communications oriented shift registers are not suitable for this, as the LEDs that are used require approximately 15-20mA to function optimally. Cost (€0,60 per piece) and availability were also decisive factors.

Since LEDs are non-linear devices that vary widely in intrinsic resistance, it's important to connect them in series with a current limiting resistor. In a parallel circuit, the LED with the least amount of intrinsic resistance could otherwise draw current that is outside of its designed specifications and draw most of the current, while the other LEDs would hardly light up. Since resistors have a much lower variance in resistance, and have a higher impedance than the LEDs, the current division will be equalized. The current flowing through every single LED is also restricted at around 22mA by using 150Ω resistors.

Calibration

Aside from driving the LEDs at the right power, the Arduino is also tasked with all communications to the tracking system as to blink the LEDs at predefined patterns, so that we can uniquely label them. This is a crucial constraint in order to guarantee proper functioning of the pose reconstruction. In this current implementation the Arduino constantly listens on a specified serial port until it receives a command. If it receives a command, it will turn on one of the subgroups of LEDs in a predefined pattern. If for example, the command 'C' is received, all the LEDs will first turn off and then LED subgroup C will turn on, starting with LED 9 and ending with LED 12. A calibration command (A,B,C,D) is always followed by either another calibration command, or the command to turn all the LEDs on ('1'). The LEDs will stay on as long as the array is successfully tracked. When tracking is lost, i.e. less than 4 LEDs are found and therefore no unambiguous pose can be reconstructed, the calibration routine will be called again initiating a series of patterns.

In order to accommodate tracking LEDs that come into LoS, we use the current calculated pose to predict the positions of the LEDs from the array onto the captured image. If there is an artefact close to a predicted LED and the angle of the vector this predicted LED is facing, with respect to the camera, is smaller than the threshold, it will match the artefact to the LED. It will also use this method to un-match LEDs of which the angle has become too large, since it is hard to accurately track artefacts that are at an almost 90 degree angle with the camera.

With this type of calibration, after an initial pose has been found, all visible LEDs can be marked and this will in turn increase the accuracy of the pose reconstruction. Furthermore, some obstruction between the camera and the HMD is possible, as long as more than 4 LEDs are still visible.

4.2.3 Positional Tracking Algorithm

Reconstructing the pose is a crucial part of the system. If the pose cannot accurately be reconstructed, the VR experience will be greatly impacted. In order to reconstruct a pose, a minimum of 4 LEDs need to be found and the reconstruction becomes more stable with more LEDs. In this section, we assume that at least 4 LEDs have been found by the camera calibration process as discussed in section 4.2.2. The pose reconstruction is based on a tutorial in OpenCV [23].

In OpenCV there are several methods for solving a perspective-n-point (PnP) camera pose estimation as discussed in section 3.2. To determine which method was the best for optimizing the accuracy of our algorithm we tested multiple PnP solvers against each other with as performance criterion the re-projection error. The test involved a stationary test object with the camera rotating around it, capturing the image at different resolutions and frame rates. The results can be seen in figures 4.8, 4.7 & 4.6. True/False is concerning the use of extrinsic guess, where the previous pose will be assumed to be correct and taken into consideration while estimating a new pose if set to true.

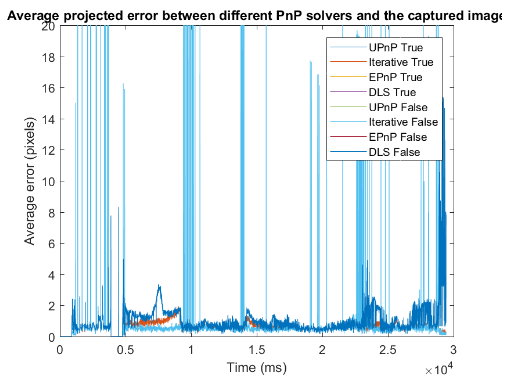


Figure 4.6: Different solvers tested on re-projection error - 320x240 @190 fps

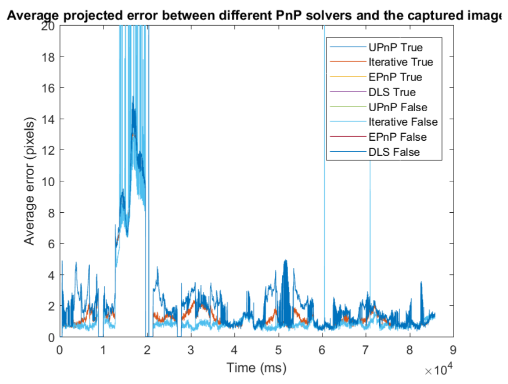


Figure 4.7: Different solvers tested on re-projection error - 640x480 @60 fps

A couple of factors are worth mentioning here. The vast majority of the stack performed exactly the same and are therefore not visible in the plot. This is a bug in the library, as the function *cv::solvePnP()*

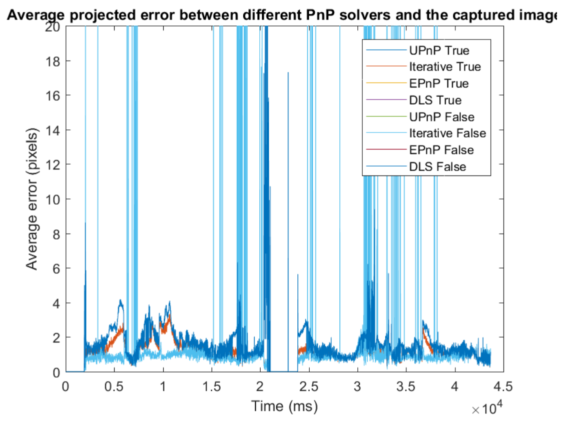


Figure 4.8: Different solvers tested on re-projection error - 640x480 (upscaled) @190 fps

automatically defaults to EPnP when DLS or UPnP are requested [24]. Extrinsic guesses also seem to be disabled for these methods. Then there remain 3 different solvers that do show a difference in accuracy.

The first is EPnP with extrinsic guess turned off. This one seems to perform the worst across the board, but it should be said that this implementation is also a lot less computationally expensive.

Next there are the 2 variants of the Iterative methods, which are as discussed in section 3.2, implemented based on the iterative Levenberg-Marquardt algorithm, with and without extrinsic guess. If it is turned off the algorithm seems to perform best of all implementations. There are however instances where the algorithm can not find an accurate pose, in which case the re-projection error ramps up. If extrinsic guess is on, the accuracy drops overall, but spikes are prevented. Prolonged use of extrinsic guess could decrease accuracy over time, if there is an error in the pose it will be exaggerated by the use of extrinsic guess.

Since the spikes for Iterative false are only for one frame at a time, we have decided to rely on the extrinsic guess off by default and only switch to extrinsic guess on if the re-projection error jumps above a certain threshold (5 pixels in our case). If the extrinsic guess has a large error as well, EPnP is used. When all are above the threshold, the last correct pose is used instead.

The found pose is then put into a Kalman filter, which is implemented as in the tutorial [23], which uses an implementation of the OpenCV function *cv::KalmanFilter* based on a linear Kalman filter for position and orientation tracking [25]. This is done in order to smooth out transitions between different poses. The effect this has on the VR experience has not been tested yet. It is possible that it introduces to much latency when moving quickly.

Chapter 5

Validation

In this chapter the performance of the system will be discussed based on the results from the conducted tests. The design will also be evaluated based on whether it meets the specified requirements from chapter 2. First we will discuss whether the calibration procedure for our camera was capable of effectively suppressing the distortion. Finally, we will evaluate the optical positional tracking system based on how robust our tracking algorithm is and how accurately we can estimate the head pose.

5.1 Distortion Suppression

After calibration with the OpenCV function *cv::calibrateCamera()*, the following results were obtained for the intrinsic parameters PS3 Eye camera when capturing at a resolution of 320x240:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 17.59 & 0 & 148.3 \\ 0 & 17.60 & 117.4 \\ 0 & 0 & 1 \end{bmatrix}$$

And for 640x480:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 34.32 & 0 & 307.3 \\ 0 & 34.17 & 251.4 \\ 0 & 0 & 1 \end{bmatrix}$$

The OpenCV calibration function stores the focal lengths in millimetres. What can be seen from the results, is that the horizontal dimension, f_x, is slightly different than the vertical dimension f_y. Moreover, the principal point of the camera (c_x, c_y) does not align with the center of the sensor (160, 120) and (320, 240) resp.

The distortion parameters returned by the calibration function for 320x240 were:

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -6.36 \cdot 10^{-2} \\ -9.49 \cdot 10^{-1} \\ 6.20 \cdot 10^{-4} \\ -5.70 \cdot 10^{-4} \\ 5.72 \end{bmatrix}$$

And for 640x480:

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -1.41 \cdot 10^{-1} \\ -2.21 \cdot 10^{-1} \\ -1.62 \cdot 10^{-3} \\ 2.51 \cdot 10^{-3} \\ 2.63 \cdot 10^{-1} \end{bmatrix}$$

Distortion suppression was tested by comparing the tracking accuracy by assuming a perfect camera model (inaccurate mathematical model), to the tracking accuracy obtained by providing the calibrated camera model to the PnP solvers. Applying these distortion parameters to our pose reconstruction algorithm significantly improved the tracking accuracy of our system. The results for the re-projection errors caused by distortions can be found in figures 5.1, 5.2 & 5.3.

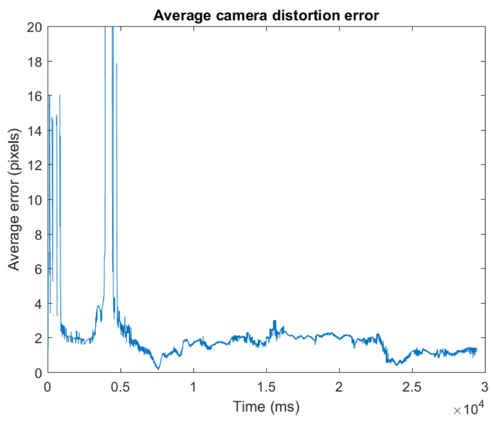


Figure 5.1: Average re-projection error due to distortions - 320x240 @190 fps

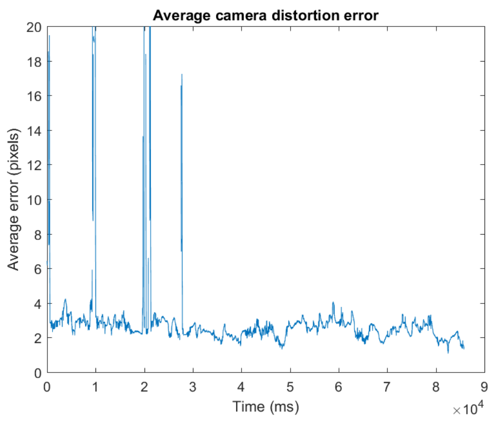


Figure 5.2: Average re-projection error due to distortions - 640x480 @60 fps

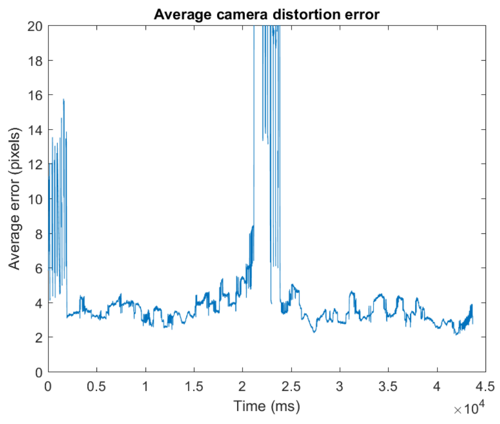


Figure 5.3: Average re-projection error due to distortions - 640x480 (upscaled) @190 fps

Note that while, the error in figure 5.1 seems lower, the resolution is also 4 times as low, which means that the error also scales proportionally. As can be seen in section 5.2.2 later on, pixel deviations ranging from 2 to 4 pixels are more than 2 to 4 times the average re-projection error of the entire tracking system. We can conclude that the distortion suppression is essential in our design and that our requirements in this aspect have been met.

5.2 Optical Tracking

In this section we will discuss the results of our tests regarding our positional tracking system. In these tests we moved the tracking object in front of the camera in different random poses and tested the system based on its capabilities to maintain tracking, finish calibration quickly and accurately reconstruct a pose. After each test, some thresholds (for example maximum angle, exposure and gain) were slightly changed to improve the algorithm, as such the third test is the best test. Note that these thresholds need to be redetermined for the final product and it is therefore not useful to mark the exact values of these thresholds. All tests were done while capturing at a resolution of 320x240 at around 180-190 fps and upscaling this image to 640x480 before processing it. The test object was only moved at a relatively low speed of about 1-2 m/s as can be seen in figures 5.7, 5.11 and 5.15. In all tests, x is the horizontal movement parallel to the camera lens, y is the vertical movement and z is the horizontal movement perpendicular to the camera lens. All tests captured 5000 iterations of the algorithm (which is independent from the framerate). The specifications for the laptop on which this test is performed can be found here [26].

5.2.1 Tracking Robustness

During the first two tests, tracking of the test object was never lost, which can be seen in figures 5.5 and 5.9. Note that in the third test, figure 5.13 20-25 seconds, tracking was lost on 3 occasions. For the first and third time it took a bit more than a second to track the test object again, which is higher than you should expect, however our current test object has some poses in which the calibration procedure can never succeed. This is due to this specific faceted design of the object, since there is only 1 LED on each facet and the angle between each facet is 45 degrees. The final HMD design will not have this problem and calibration can thus be expected to be faster.

In figures 5.5 and 5.9, you can clearly see that the initial calibration was quick. When the test started (at t=0), the program calibrates and finds the HMD almost instantaneous.

5.2.2 Tracking Accuracy

The accuracy of the algorithm can be determined by looking at the graphs of the pose re-projection error. When looking at figures 5.4, 5.8 and 5.12, you can see that the calculated pose is in general only off by about 1-4 pixels with only a handful of outliers.

5.2.3 Tracking Smoothness

The smoothness of tracking can be determined by looking at the graphs of the position and rotation. When looking at the first 2 tests, figures 5.5 5.6 5.9 and 5.10, the tracking is in general smooth, but there are a few outliers in the position and/or rotation. In test 1, there are jumps at 6 and 20 seconds. In test 2 there are jumps at 13, 23, 25, 37, 38 and 39 seconds. When looking at the third test however, figures 5.13 and 5.14 the only outliers during tracking occur during the calibration phases, at 1, 21 and 24 seconds. One other observation that is noteworthy, is that while the positional tracking is almost completely smooth (fluctuating with an amplitude of 1-2 mm, depending on the pose), the rotational tracking fluctuates at some points with an amplitude of 0.07 radians. However, at different poses (when other/more LEDs are within line of sight) the amplitude is only about 0.01-0.02 radians.

From the results for the robustness, accuracy and smoothness, we can conclude that the algorithm can accurately track the position of a single HMD as long as it remains within line of sight of the camera and there is no large obstruction between the HMD and the camera. The rotational tracking is not always accurate, depending on the pose.

5.2.4 Tracking Speed

Since all tests captured 5000 iterations of the algorithm and the fps was on average 185, the average computation time can be calculated by dividing the total time of the test by 5000. Test 1 took 40.67 s to complete, test 2 took 42.88 s to complete and test 3 took 35.18 s to complete. This results in an average computation time of 8.13 ms, 8.58 ms and 7.04 ms respectively. To compare this, the average time between each frame is also needed. When using the average fps of 185, this is 5.41 ms.

From the result of the tracking speed, we can conclude that the algorithm is slower than the required 5 ms to compute the pose on every frame on the laptop on which this test is performed. It should be noted that the laptop on which the tests are done, does not have the specifications required to accommodate a HMD.

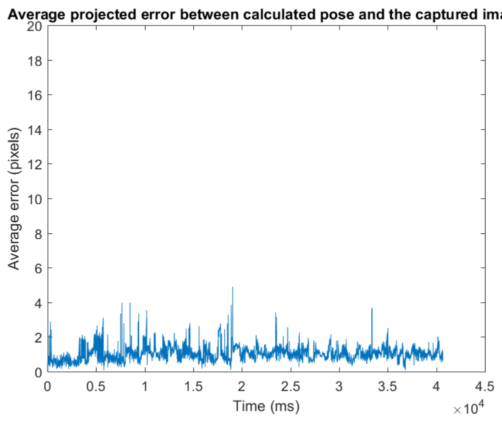


Figure 5.4: Average re-projection error of the pose reconstruction, test 1

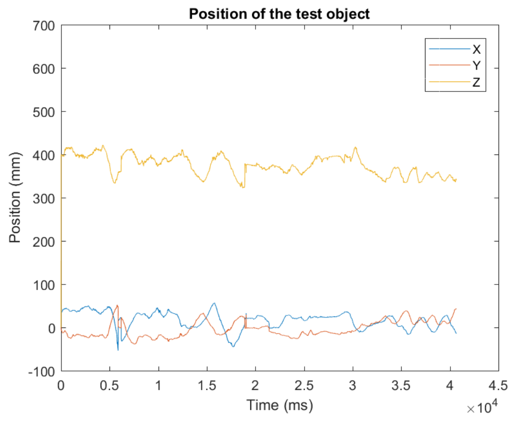


Figure 5.5: Tracked position of the test object, test 1

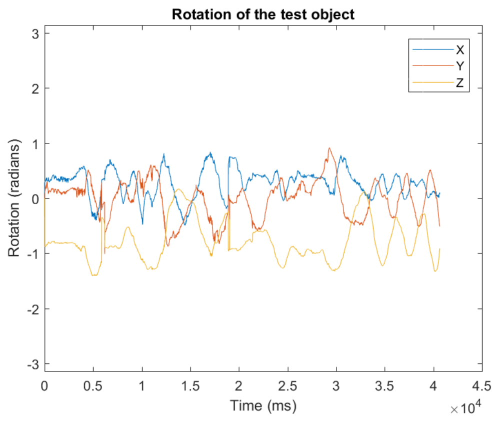


Figure 5.6: Tracked rotation of the test object, test 1

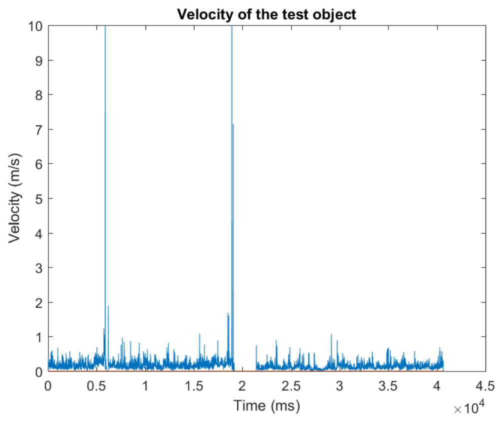


Figure 5.7: Tracked velocity of the test object, test 1

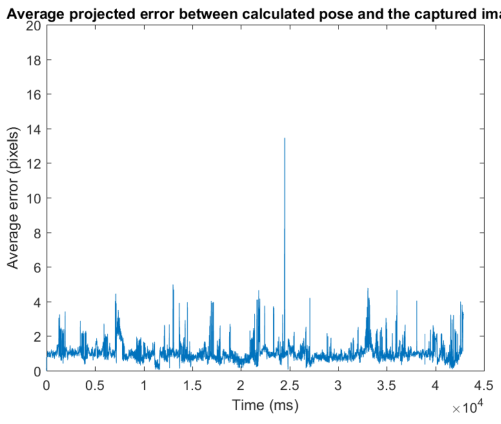


Figure 5.8: Average re-projection error of the pose reconstruction, test 2

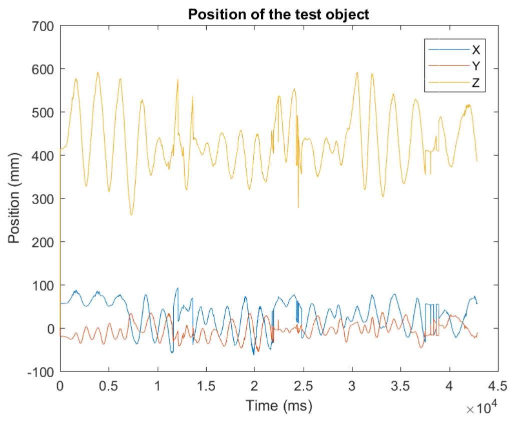


Figure 5.9: Tracked position of the test object, test 2

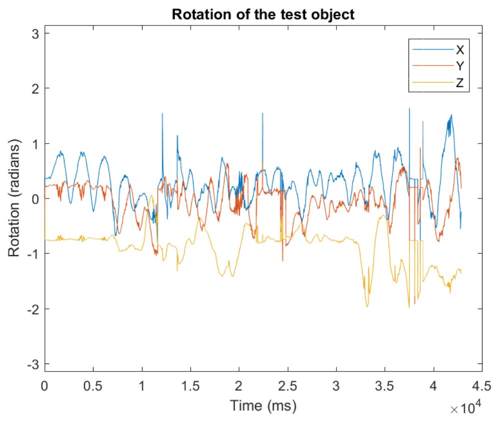


Figure 5.10: Tracked rotation of the test object, test 2

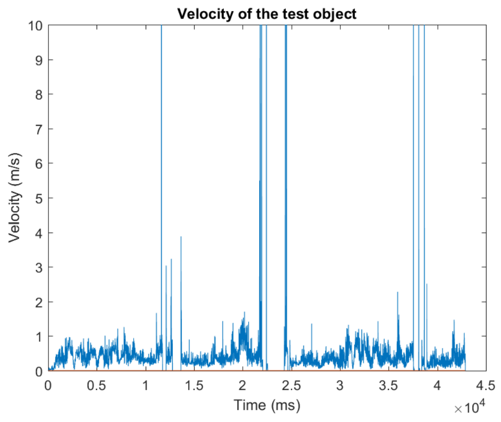


Figure 5.11: Tracked velocity of the test object, test 2

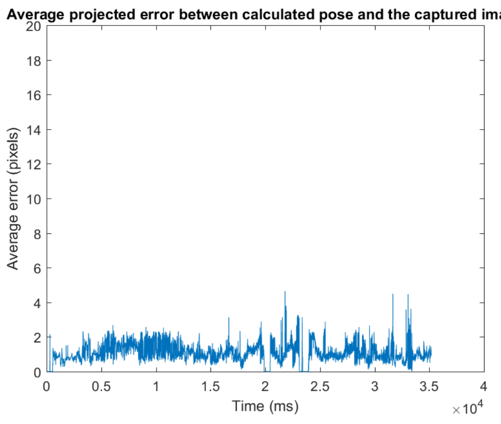


Figure 5.12: Average re-projection error of the pose reconstruction, test 3

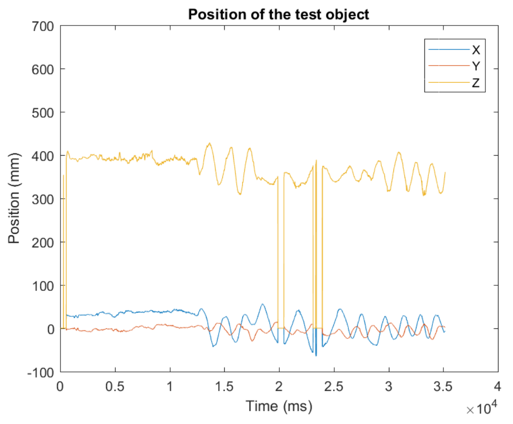


Figure 5.13: Tracked position of the test object, test 3

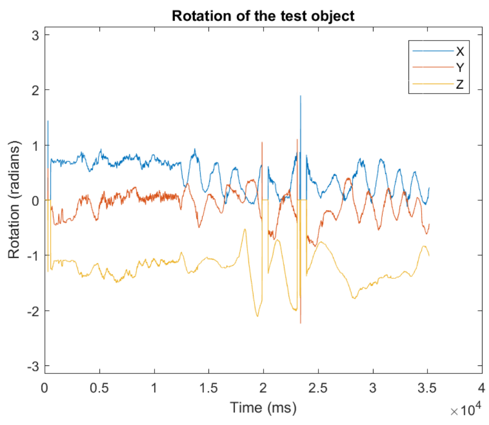


Figure 5.14: Tracked rotation of the test object, test 3

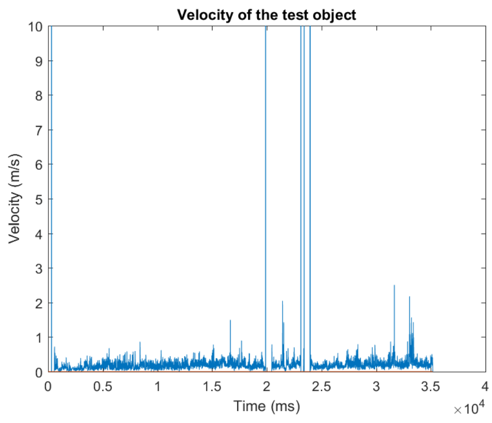


Figure 5.15: Tracked velocity of the test object, test 3

Chapter 6

Conclusions & Recommendations

First, let's start with a summary of all the requirements from section 2.

Requirement	**Satisfied?**	**Additional remarks**
The system must be capable of tracking a single HMD	✓	
The system must be capable of tracking user movement in 6 Degrees of Freedom	no	Pose estimation is unstable at certain angles, see section 5.2.1
The LED array must not transmit light within the visible spectrum	✓	The use of 940nm IR LEDs assures that this requirement is met, see section 4.2.2
The system must indicate when tracking is lost, so that the user can be disconnected from the virtual world gracefully	✓	The specific application of this is up to the developer
The system must be capable of correcting for distortions caused by the camera lens	✓	See section 5.1
The system must be capable of distinguishing the LED array from ambient light and internal lens reflections	✓	The calibration routine makes it virtually impossible to incorrectly label image artefacts, see section 4.2.2
The tracking algorithm must not exceed 5ms of runtime per frame on PCs that meet the recommended specifications in table 2.1	no conclusion	The system has not been tested yet on a qualifying PC, see section 5.2.4
The optical tracking will be implemented in C++ with the open-source OpenCV 3.2 library	✓	
The LED array will be controlled by a Teensy micro-controller programmed in C	partly	For the prototype an Arduino Uno is used, but this code is portable to a Teensy, see section 4.2.2
The total sum of the costs of the components used in the tracking solution should not exceed €50,-	✓	The total bill of materials amounts €44,90, see section 4

Table 6.1: Design tested against the Program of Requirements

From our tests it is clear that the design is capable of tracking an HMD, however due to time constraints it has not been tested if the VR experience will be pleasant. Our design is able to accurately track the position of our test object as long as it remains within line of sight (without any obstruction) and it is moving at a relatively low speed of about 2-3 m/s.

From our tests, it is not entirely clear if our algorithm can accurately determine the rotation of a HMD. We strongly believe the fluctuation in the rotation is due to not having enough LEDs within line of sight and because the positions of the LEDs are not completely aligned with the positions in the design of the test object, due to the difficulties during the assembly of the LEDs inside the test object. This believe is supported by the test results, which show that the tracking is greatly influenced by the pose of the test object. Our recommendation is to design a HMD, where it is easy to assemble the LEDs in a fixed location. Another solution is to determine the locations of the LEDs post-assembly using multiple capture points.

We recommend to implement the use of an IMU sensor [27] in the design for two reasons. First, it gives additional data to track the rotation of the HMD, which will improve the VR experience. Secondly, our design cannot track an HMD when the program is in the calibration phase. For small timeframes, preferably less than a second, the IMU sensor can be used to track the position of the HMD. For larger timeframes the positional error grows quickly due to integration of the velocity and acceleration vector to obtain the position and is therefore not suitable to track the position of the HMD on its own for a prolonged period.

The program at this point lacks the ability to gracefully close the view inside the HMD. We recommend to implement an algorithm that, when tracking is lost, extrapolates the position, using the velocity and acceleration computed from the last couple of frames where the HMD is tracked, and then slowly fades the screen to black. Furthermore, when tracking is acquired, the screen should slowly fade in from black to the desired VR world.

The algorithm at this point does not meet the 5 ms computation time requirement as stated in the program of requirements. However, the laptop on which the tests were done is not representative for the computers on which the program will run. Despite this it is not far off and it can compute the pose at over 140 fps. It remains to be seen if the program is fast enough to provide a smooth VR experience on computers with better specifications. We recommend to optimize the code further to reduce the time required to process a frame and to test this on a computer better capable of accommodating a HMD.

Finally, the program has been fully implemented using open-source libraries, and using components that fit well within the envisioned budget. Some recommendations to improve the tracking solution while still keeping the cost of the entire system in check, would be to use a higher resolution camera and a cheaper micro-controller. This would improve the robustness of the tracking solution, while having a negligible impact on the the latency of the system.

Bibliography

[1] S. Kovach. (2017) What happened to virtual reality? [Online]. Available: http://www.businessinsider.com/what-happened-to-virtual-reality-2017-1

[2] M. Stengel. (2015) Gaze-Tracking Head Mounted Display. [Online]. Available: https://inmagicwetrust.wordpress.com/2015/11/09/gaze-tracking-head-mounted-display/

[3] (2017) Oculus Rift + Touch - Recommended PC Specifications. [Online]. Available: https://www.oculus.com/rift/

[4] J. Durbin. (2016) Oculus Sensors Can Now Be Purchased Individually Online For $79. [Online]. Available: https://uploadvr.com/oculus-sensors-can-now-purchased-individually-online-79/

[5] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics Principles and Practices (Third Edition)*. Pearson Education, 2014, ch. 11, p. 265.

[6] A. Kaehler and G. Bradski, *Learning OpenCV 3*. O'Reilly Media, 2017, ch. 18, pp. 787–823.

[7] Z. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE*, 2000.

[8] (2016) OpenCV modules. OpenCV. OpenCV documentation. [Online]. Available: http://docs.opencv.org/3.2.0/

[9] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, "Iterative Pose Estimation Using Coplanar Feature Points," *Elsevier*, 1996.

[10] J. Ventura, C. Arth, C. Reitmayr, and D. Schmalstieg, "A Minimal Solution to the Generalized Pose-and-Scale Problem," *IEEE*, 2014.

[11] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography," 1981.

[12] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," 2008.

[13] L. Kneip, H. Li, and Y. Seo, "UPnP: An Optimal O(n) Solution to the Absolute Pose Problem with Universal Applicability," *Computer Vision - ECCV 2014*, 2014.

[14] D. Kosmopoulos, "Robust Jacobian matrix estimation for image-based visual servoing," *Elsevier*, 2011.

[15] S. R. Buss, "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods," 2009.

[16] H. P. Gavin, "The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems," 2017.

[17] (2017) Virtual Reality Devices. [Online]. Available: https://xinreality.com/wiki/Virtual_Reality_Deviceshttps://xinreality.com/wiki/Virtual_Reality_Devices

[18] M. Stengel. (2017) Mono/Stereo Camera Calibration Using OpenCV. [Online]. Available: https://inmagicwetrust.wordpress.com/2017/05/09/camera-calibration-using-opencv-3/

[19] Teensy USB Development Board. PJRC. [Online]. Available: https://www.pjrc.com/teensy/index.html

[20] (2009) HT-191IRAJ LED datasheet. [Online]. Available: http://downloads.cdn.re-in.de/175000-199999/181698-da-01-en-SMD_IR_LED_805_HT_170IRPJ.pdf

[21] C. Maw and T. Igoe. (2006) Serial to Parallel Shifting-Out with a 74HC595. [Online]. Available: https://www.arduino.cc/en/Tutorial/ShiftOut

[22] (2016) 74HC595N Shift Register Datasheet. [Online]. Available: https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf

[23] Real Time pose estimation of a textured object. [Online]. Available: http://docs.opencv.org/trunk/dc/d2c/tutorial_real_time_pose.html

[24] P. Hasper. (2015) Ignoring the flags of solvePnP and silently executing EPNP. [Online]. Available: http://code.opencv.org/issues/4472

[25] M. Bauer. Sensor Fusion using the Kalman Filter. [Online]. Available: http://campar.in.tum.de/Chair/KalmanFilter

[26] Dell XPS 13 9360-80VVD - Specificaties. [Online]. Available: https://tweakers.net/pricewatch/610035/dell-xps-13-9360-80vvd/specificaties/

[27] Starlino. A Guide To Using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. Starlino. [Online]. Available: http://www.starlino.com/imu_guide.html