# Optical measurement of virtual reality headset performance

Master's thesis, Friday 20th April, 2018

Author:

Sakari Kapanen

Supervisors:

Kimmo Jokinen (OptoFidelity)
Arttu Miettinen (JYFL)

UNIVERSITY OF JYVÄSKYLÄ
DEPARTMENT OF PHYSICS

# Abstract

Kapanen, Sakari
Optical measurement of virtual reality headset performance
Master's thesis
Department of Physics, University of Jyväskylä, 2018, 74 pages.

A measurement method and an embedded camera module were developed for measuring the performance of the tracking system of virtual reality headsets. The measurement setup is an end-to-end one where a robot moves the headset and the camera measures the changes of the display content. The objective is to optically measure the accuracy of the headset's tracking system by imaging the content on the device's display, synchronized to the display refresh. The thesis work focused on the development of the smart camera and the related computer vision algorithms.

In the measurement, a three-dimensional target object with detectable points is shown on display. By locating these points in the projected image, the pose (translation and rotation) of the virtual camera can be estimated. From this pose, one can deduce the head pose that is estimated by the headset's tracking system. By measuring the pose during different motion sequences performed by the robot, one can probe the tracking system for inaccuracies such as drifting and jitter.

A simple pose estimation method based on linear least squares fitting was developed for measuring small pose changes. The method was characterized by Monte Carlo simulation against a non-linear pose estimation method implemented in the OpenCV computer vision library, with noise added to the source data to represent real measurement conditions. It was found that a planar target object caused ambiguities between the pose variables. In the simulations, adding an off-plane point to the target made the problem better conditioned and less sensitive to noise. Error estimates were derived for the new method. In addition to simulations, measurements with the robot and the HTC Vive headset were also performed.

Keywords: virtual reality, computer vision, pose estimation, test automation

# Tiivistelmä

Kapanen, Sakari
Virtuaalitodellisuussilmikoiden suorituskyvyn optinen mittaaminen
Pro gradu -tutkielma
Fysiikan laitos, Jyväskylän yliopisto, 2018, 74 sivua

Virtuaalitodellisuussilmikoiden päänseurantajärjestelmän suorituskyvyn mittaamista varten kehitettiin mittausmenetelmä ja sulautettu kameramoduuli. Kyse on päästä päähän -mittauksesta (engl. end-to-end measurement), jossa robotti liikuttaa silmikkoa ja kamera mittaa sisällön muutoksia näytöllä. Tavoitteena on optisesti mitata päänseurantajärjestelmän tarkkuutta kuvaamalla näytön sisältöä näytön virkistykseen tahdistettuna. Työssä keskityttiin kameramoduulin ja siihen liittyvien konenäköalgoritmien kehittämiseen.

Mittauksessa näytöllä on kolmiulotteinen kohde, jossa on tunnistettavia pisteitä. Paikallistamalla nämä pisteet kuvassa voidaan määrittää grafiikkamoottorin virtuaalisen kameran asento (siirtymä ja kiertymä) ja saada siten selville päänseurantajärjestelmän arvioima asento. Mittaamalla tätä asentoa erilaisten robotilla toistettavien liikkeiden aikana saadaan tietoa seurannan epätarkkuuksista kuten ajautumisesta ja tärinästä.

Asennon arvioimiseksi kehitettiin lineaariseen pienimmän neliösumman sovitukseen perustuva menetelmä pienten asennon muutosten mittaamiseksi. Tätä menetelmää karakterisoitiin vertaamalla sitä OpenCV-konenäkökirjaston epälineaariseen vastaavaan menetelmään. Karakterisointi tehtiin Monte Carlo simulaatioilla, joissa simuloituun mittausdataan lisättiin kohinaa. Simulaatioissa huomattiin, että tasomaisella kohdekuviolla asennon eri komponentit sekoittuvat keskenään. Lisäämällä malliin tason ulkopuoleinen piste saatiin ongelmanasettelua parannettua ja kohinaherkkyyttä pienennettyä. Menetelmälle johdettiin myös virhe-estimaattori. Simulaatioiden lisäksi tehtiin mittauksia robotilla ja HTC Vive -silmikolla.

Avainsanat: virtuaalitodellisuus, konenäkö, asennon arviointi, testausautomaatio

# Preface

This thesis work was done for OptoFidelity, a global company focusing on test automation, robotics and machine vision. The work started as a part of a project where the aim was to develop a robotic system for measuring the tracking accuracy of virtual reality headsets.

In the project, there was a great deal of hardware and software engineering involved, ranging from robot mechanics design to microcontroller firmware development, and there were several employees in the project team. My work and the content on the thesis focused on the software engineering, including microcontroller firmware development and algorithm design. My colleagues also provided a wealth of ideas, support and great collaboration.

I would like to thank my supervisor Kimmo Jokinen, OptoFidelity's CTO, for providing this interesting subject that was a good match for my background and a great opportunity to learn. Special thanks go to Professor Steven M. LaValle for kindly reviewing the thesis and giving valuable ideas for improvement. Additionally, I would like to thank Arttu Miettinen, the other supervisor of this work, for giving essential advice on thesis writing and another perspective on the subject. Finally, I would like to express my gratitude for the unconditional support I have received from my family and friends during my studies.

Jyväskylä, Friday 20$^{\text{th}}$ April, 2018

Sakari Kapanen

# Contents

# 1 Introduction

Virtual reality (VR) has been a field of research for a long time, with one of the first room-scale simulations dating back in 1992 [1]. However, until very recently, the display and computing technology has not been advanced enough for manufacturing reasonably priced consumer grade virtual reality devices. In the last few years, the first head mounted displays (HMD), also called headsets, have began entering the consumer market. A VR headset is a device which is worn by the user and displays the rendered three-dimensional content as a stereoscopic image (one two-dimensional image for each eye) to create an illusion of a virtual world. In addition to the display and lenses, a headset also contains hardware and sensors which are used to track the user's head motions. This information is used to render to display content accordingly so that the user can explore the virtual world by actually moving around. Examples of VR headsets are HTC Vive [2] (shown in figure 1) and the Acer Windows Mixed Reality headset [3].

To create a convincing virtual reality experience, the digitally rendered and displayed content has to meet the capabilities of the human senses. Frame rate, visual density, field of view and latency (will be defined in section 1) are among the most important parameters in this sense. The manufacturers strive to improve these parameters in order to create more immersive VR experiences, i.e. to improve the feeling of presence in the virtual world. That is also where the current implementations are lacking, but research of new technologies is continuously being done to improve the situation. Especially display and motion tracking technologies are under rapid evolution.

To evaluate the new hardware and software technologies, the manufacturers and application developers must be able to objectively compare between hardware and software revisions. Repeatability and objective, numerical indicators of performance are particularly important for software developers to evaluate their changes in various algorithms and drivers involved in the system.

OptoFidelity is a company specializing in test automation solutions for smart devices and has also implemented a novel measurement method and device for


**Figure 1.** HTC Vive virtual reality headset. Image by Maurizio Pesce, licensed under CC BY 2.0 [4].

benchmarking VR head mounted displays [5]. At company level, this tester is first of its kind and although it already produces valuable information for customers, it has some limitations. Therefore, a new tester has been designed that is capable of more accurate measurements and reproduction of human motions in three translational and three rotational degrees of freedom.

This thesis work was done for OptoFidelity on the software and measurement design for the new VR tester. The work consisted of designing software for an in-house developed smart camera, designing measurement methods and algorithms and implementing measurement data flow, architecture and signal processing. The focus of the thesis is on measuring the performance of the head tracking system of VR headsets by means of computer vision and a smart camera synchronized to the display content refresh.

**Table 1.** Comparison of parameters of several head-mounted displays. Visual density is calculated using (1), assuming that the vendor-reported field of view is the vertical one. The display persistences have been measured with OptoFidelity Video Multimeter. The rest of the parameters are reported by the manufacturers.

| | HTC Vive [2] | PlayStation VR [6] | Acer AH101-D8EY [3] |
|---|---|---|---|
| Display technology | AMOLED | OLED | LCD |
| Resolution per eye | $1080 \times 1200$ | $960 \times 1080$ | $1440 \times 1440$ |
| Field of view (°) | 110 | 100 | 100 |
| Visual density (px/°) | 10.9 | 10.8 | 14.4 |
| Frame rate (Hz) | 90 | 120, 90 | 90 |
| Persistence (ms) | 1.9 | 2.5 | 2.6 |

# 2 Performance characteristics of virtual reality headsets

In this section, several important parameters of virtual reality devices are considered. Display parameters will be covered briefly, while the rest of the thesis focuses on the performance parameters of the tracking system. The goal is to identify potential test cases to be implemented on a VR performance testing platform.

## 2.1 Display characteristics

In modern VR headsets, the base display technologies are usually similar to what is found in mobile phones. Typically the panels are either (active matrix) organic light emitting diode ((AM)OLED) [2] or liquid crystal display (LCD) [3] type. However, to create a realistic virtual reality experience, the display needs to have some special properties which contribute to the immersivity in their own part. A comparison of several display parameters from a set of VR headsets is shown in table 1. The quantities will be defined later in this section.

Visual density is the number of pixels covering a certain unit of the visual angle of view. This affects how “pixelated” or “grainy” the image looks like. It is a useful quantity for comparing different devices since it accurately represents the perceived

image resolution. It can be calculated as

$$\rho = \frac{n}{\alpha}, \tag{1}$$

where $n$ is the horizontal resolution and $\alpha$ is the horizontal field of view (FOV). Vertical resolution and FOV could be used as well and would result in an essentially similar value for most displays. The units of $\rho$ are usually px/° (pixels per degree) or px/′ (pixels per arcminute).

There is a standardized, clinical measurement method for visual acuity. The test subject is placed at a fixed distance from a chart with so-called Landolt C-rings or similar patterns of different sizes. In the target patterns there is a small detail, e.g. a gap, that the subject has to distinguish. The smallest distinguished gap determines the visual acuity of the subject. The visual acuity is usually defined as

$$\text{visual acuity} = \frac{1}{\alpha}, \tag{2}$$

where $\alpha$ is the visual angle covered by the smallest distinguished detail, in arcminutes. [7]

Ideally, the visual density should match the capabilities of the typical human eye, specifically the visual acuity. Visual acuity of 1.0 is considered as "normal vision" [7]. Therefore, a person with normal vision can resolve angles of approximately one arcminute. To match that, the display in the headset would have to have a visual density of 60 px/°. As it can be seen from table 1, all the listed devices have a visual density below 20 px/° and thus there is room for improvement in future iterations. Increasing the visual density requires an increase of display resolution which places hard demands on the rendering hardware.

It is worth noting that the area of the sharpest vision on the retina, called fovea, is quite narrow and located in the centre of the field of view. Equipped with eye tracking technology, headsets could selectively render only a small part of the screen at high density, so that the fovea gets the highest visual density. This kind of technology is called foveated rendering and it may, in part, solve the problem of visual density in the future. [8]

The field of view of the headset also affects the immersivity of the virtual world. A single human eye has a FOV of roughly 160° horizontal × 175° vertical and the binocular field of view is 200° × 135°, respectively [9]. Comparing to the figures

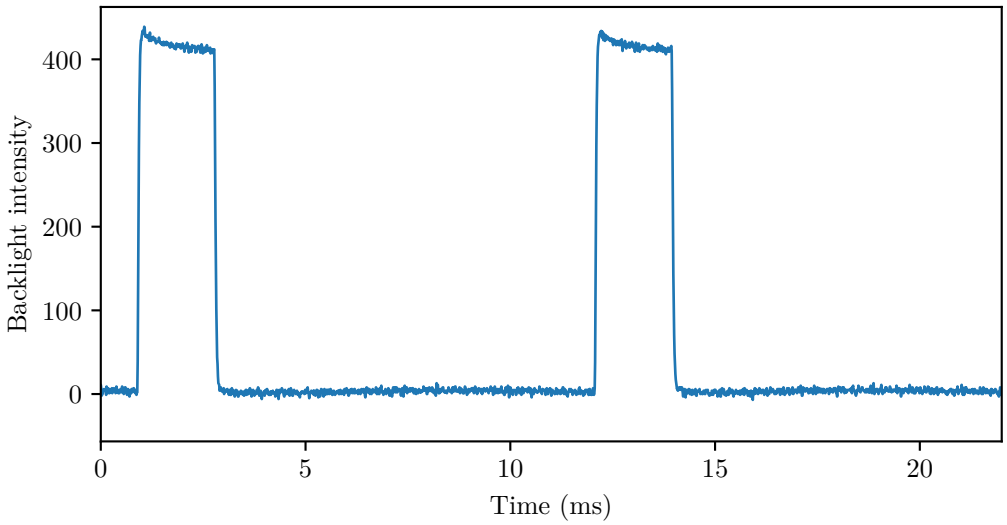



**Figure 2.** The backlight intensity (in arbitrary units) of HTC Vive as a function of time for two frame periods, measured by OptoFidelity Video Multimeter. The frame period is approximately 11 ms and the persistence (the time when the backlight is on) is approximately 2 ms.

in table 1, current HMD implementations are lacking in this respect, likely due to optical difficulties in achieving a distortion-free image at such a wide field of view.

Since liquid crystal displays have been adopted in wide usage, edge blur caused by moving targets on the display is a well-known effect. It is partially caused by slow response times of the display, partially by intensity averaging taking place in the eye itself [10]. While the former effect has been reduced in modern displays (especially OLED type), the latter cannot be avoided.

It has been noticed that driving the display in short impulses is effective in mitigating the intensity averaging effect [10]. This is why VR displays only display the content for a short sub-period of the whole frame time. This period is called the frame persistence and is typically a few milliseconds long. As an example, the backlight intensity of HTC Vive as a function of time is shown in figure 2. For each frame period of 11 ms, there is a short high pulse of 2 ms with sharp rising and falling edges. The length of this pulse is called the persistence.

It is worth noting that, for example, augmented reality devices like HoloLens [11] may employ more complex display technologies. However, the discussion of the basic parameters applies to them to a good extent.

## 2.2 Temporal characteristics of VR systems

While the display parameters play an important role in creating a good VR experience, there are also system level parameters that affect the experience. The headset has to track the user's motion and reflect it as changes on the content. The process of sampling the sensors and calculating the head orientation and three-dimensional position needs to be accurate and fast.

### 2.2.1 Head tracking technology

There are several different tracking technologies that are employed in modern VR headsets. Some virtual reality devices such as Android smartphones track the motion only in three rotational degrees of freedom (DOF), neglecting the user's translational motion (i.e. walking around) in the surrounding space. More advanced devices like HTC Vive have full rotational and translational tracking in six degrees of freedom.

Most HMDs use a sensor called an inertial measurement unit (IMU) as a part of the rotational tracking. Such a sensor is comprised of a three-axis accelerometer, gyroscope and sometimes a magnetometer. The readings from the sensors are used to estimate the rotation angles of the headset using a method called sensor fusion. Oculus Rift Development Kit, for example, utilizes a so-called complementary filter to combine acceleration and angular velocity measurements from the accelerometer and gyroscope to yield an estimate of the orientation. This method is discussed in detail in [12].

For positional tracking, there are two basic principles: inside-out and outside-in tracking. In an inside-out tracking system, there are photosensitive sensors or cameras on the headset itself. There may be some reference points or markers in the environment which are tracked by the sensors to estimate the pose of the headset. [13] HTC Vive lighthouses are an example of such markers. It is also possible to utilize advanced imaging technology and computer vision and use features of the environment itself as a reference, like done in the Acer Windows Mixer Reality headset. This results in a markerless tracking system, requiring no external components beside the headset itself.

In outside-in tracking systems this situation is reversed — the headset itself contains markers that are observed by sensors mounted in the environment. Oculus Rift, for example, utilizes an external camera that observes infrared LEDs that are

mounted on the headset [12].

Having to install external components in the environment can be seen as a burden to the user, tying the usage of the device to a certain space. Therefore, in the future, markerless inside-out tracking systems are likely to become increasingly common.

### 2.2.2 Head tracking performance

All the described tracking technologies have their own advantages and disadvantages. The readings from inertial measurement units can be noisy. Also, the positional tracking methods have some inaccuracies and can be sensitive to disturbances. For example, if an object partially or completely blocks the line of sight between HTC Vive and the lighthouses, the tracking can be lost. The same applies also for outside-in setups if the camera fails to see the headset. On the other hand, markerless inside-out tracking systems might have difficulties due to varying surface properties, for example.

The sensor fusion algorithm also has an important role in the tracking accuracy. It is largely responsible for rejecting temporary disturbances and reacting quickly to changes. In addition to applying advanced statistical algorithms such as Kalman filters, some compromises have to be made in order to achieve a balance between quick response and noiseless tracking.

Tracking inaccuracies manifest themselves in several different ways visible to the user. Drifting, stationary jitter and the gimbal lock are some of the common inaccuracy modes. There are even more artifacts related to the quality of tracking and rendering, especially in the context of augmented reality and hologram rendering, as suggested by Microsoft [14].

Drifting is an artifact where the content on screen slowly drifts to some direction as a consequence of inaccuracies building up in the tracking algorithm. One possible cause of drifting is inaccuracy of numerical integration in the sensor fusion — for example, the angular velocities measured by the gyroscope have to be integrated to get the rotation angles. This results in numerical errors due to finite sample rate. Usually this is mitigated by using an absolute reference, e.g. gravity vector obtained from the accelerometer. However it may be difficult to get a reliable absolute reference for all six degrees of freedom, and thus drifting might be observed.

Tracking noise is seen as jitter and shaking of the displayed content when the headset stays stationary. Thus, this phenomenon is called stationary jitter. Inade-

quate filtering or poor signal-to-noise ratio can result in stationary jitter. Jumpiness is also a related type of inaccuracy, but there the artifacts are less frequent and potentially larger in their magnitude. Jumping of the pose could be caused by e.g. the tracking system failing to resolve between two ambiguous poses.

Gimbal lock is a fault in the rotation arithmetics, where the 3D orientation representation loses one degree of freedom when facing in a certain direction. This particularly happens when Euler angles are used as the primary representation of orientation. While this is not strictly an inaccuracy of tracking, it may be caused by a poor implementation of sensor fusion or some aspect of the 3D graphics engine.

### 2.2.3 Motion-to-photon latency and prediction

Another important contributor to the VR experience is the motion-to-photon latency. It is defined as the latency between user's motion and the respective update of the content on the screen. This is a particularly important parameter because too high a latency may cause motion sickness. According to Oculus, an acceptable value is 20 milliseconds or lower [15].

There are multiple factors in the hardware and software which contribute to this latency. The raw sensor readings from the tracking system have to be processed first. The result is passed to the software that renders the content. When the rendering is done, the content is transferred to the display, which has its own pixel switching latency. A simplified motion-to-photon pipeline is illustrated in figure 3.

Summing all the transfer and processing latencies in each part of the chain, one gets the end-to-end motion-to-photon latency, which is an actual measurable quantity and is probably close to what is perceived by the user. It is important not only for device manufacturers, but also VR software developers, to keep this latency under control. Therefore, motion-to-photon latency seems to be one of the most interesting parameters.

Most VR systems try to compensate for the motion-to-photon latency by employing a technology called motion prediction. It works by extrapolating the head motion based on observed angular velocities and other signals. The prediction usually succeeds in reducing the latency when the motion is predictable, with low acceleration. However, for unpredictable, high acceleration motions, the prediction fails and results in high latencies again. For sudden changes of direction, the prediction could even overshoot. Some measurements of the motion-to-photon latency in unpredictable

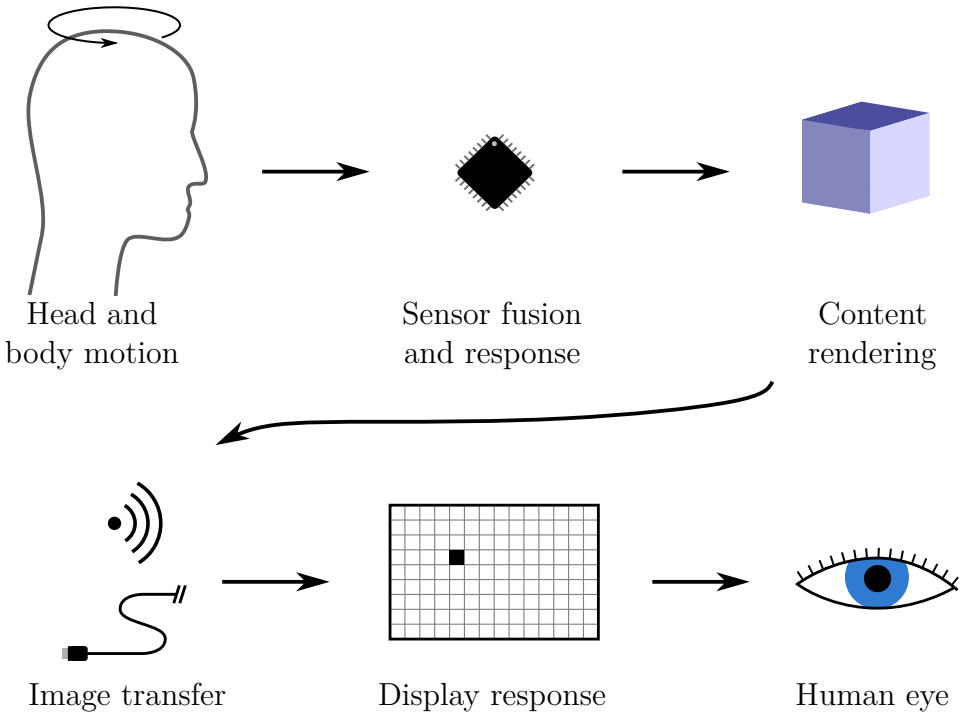



**Figure 3.** A block diagram of the motion-to-photon pipeline, drawn after the textual description in [15].

movements are presented in [16], where increased latencies and overshooting are seen to occur on multiple devices in unpredictable movements.

### 2.2.4 Perception of temporal quality

Due to differences between human individuals, the experience of presence and immersion in virtual reality is subjective. While the motion-to-photon pipeline presented in figure 3 is correct in principle, it must be noted that the human brain and senses are also important parts of the total pipeline.

The sickness caused by a poor virtual reality implementation has been a popular subject of research lately. The sickness is usually attributed to a phenomenon similar to motion sickness, caused by e.g. a high motion-to-photon latency and the resulting conflict between the signals given by the visual and the vestibular systems [17]. The vestibular system essentially serves as a sense of balance and measures linear and angular acceleration. If the user wearing a VR headset makes a rapid movement, the feedback is given quickly via the vestibular system. However, the visual feedback arrives late due to a nonzero motion-to-photon latency.

There is also a perceptual mechanism that causes improvement of perceived quality with decreasing persistence. Namely, the vestibulo-ocular reflex acts as a built-in image stabilization mechanism in the human brain: when a person rotates their head, the eyes move in the opposite direction to maintain the target of the gaze. The increase of motion blur and smearing with display persistence has been partially attributed to this reflex [18].

Another interesting phenomenon is the perception of the actual motion-to-photon latency, particularly when low persistence displays are considered. The question is what happens during the blank period when the display is switched off after showing the frame. An interesting question is if the brain can fill in the blank periods, "interpolating" between successive frames and effectively reducing the perceived motion-to-photon latency. The idea of such interpolation associated to low-persistence displays has been suggested by e.g. Paul Bakaus [19].

There could be multiple effects participating in this hypothetical interpolation. Firstly, even if the display persistence is very low, after passing a certain frequency threshold the video starts to seem continuous. This indicates that there is some kind of temporal averaging taking place in the visual system. Secondly, the role of the vestibular system is interesting. It is known that part of the motion perception comes from the vestibular system, giving rise to the aforementioned motion sickness. In principle, the vestibular system could also compensate for the latency during the display off periods by keeping up the perception of motion even when the visual content is not updated. This is a hypothesis that could be a subject of further research.

To more deeply understand the visual processing and possible interpolation between frames, one has to consider the sensory processing performed by the brain. That relates to the fields of neuroscience and brain research. At the time of writing the author is not aware of research dedicated to this specific subject matter. However, some interesting and potentially related research has been conducted. Rufin VanRullen [20] and his team, for example, have been researching periodic operations in the brain and the discreteness of sensory sampling. These phenomena are likely related to the perception of virtual reality content and the results of such research are worth investigating.

## 2.3 Existing measurement instruments

It is easy to understand that the experience of immersivity and quality of a VR implementation are highly subjective. However, subjective experiences are hard to compare. Therefore, objective numerical measurements would be valuable information in many occasions — for example, those developing software or hardware for HMDs would be able to evaluate and compare the performance between different revisions. The same applies for VR content developers, as well.

Due to this demand, several testing solutions have already emerged. Several testers and their design principles are discussed, including OptoFidelity's solution and several others.

### 2.3.1 OptoFidelity VR Multimeter

In 2016, OptoFidelity launched their first VR testing solution, OptoFidelity VR Multimeter [5]. As its name suggests, it is designed to measure multiple parameters from the device under testing, including frame rate, persistence and motion-to-photon latency.

The device uses OptoFidelity Video Multimeter [21] at its core. Video Multimeter is an embedded device for measurement of video playback performance and is described in detail in [22]. As a standalone device, it is already capable of measuring most of the directly video-related parameters from HMDs, such as frame rate and persistence.

The in-house developed Video Multimeter firmware has been since expanded to cover more VR measurements, most importantly motion-to-photon latency when the headset is rotated in one degree of freedom. The overall setup is shown in figure 4. In the VR Multimeter test hardware there is a rotary stage on which the HMD is mounted. An instrumented target pattern is shown on the HMD while it is rotated. There is a camera capturing the content on the HMD and a shaft encoder giving an absolute reference of the rotational position. A custom target pattern is shown on the HMD and its optical flow is tracked with the camera during rotation. The encoder and optical flow curves are fitted by finding a temporal shift that minimizes the difference of the curve shapes.


**Figure 4.** OptoFidelity VR Multimeter setup, including the Video Multimeter, rotary head stage and controller PC. © OptoFidelity, 2017.

### 2.3.2 Other testing solutions

Virtual reality is still quite young as a field of industry, and although research has been going on for several tens of years, it has not focused on performance measurement until very recently. Hence, there are not that many ready-made solutions for e.g. latency measurement in the market. Most of the solutions tend to only measure a part of the motion-to-photon pipeline, thus not measuring the full end-to-end latency. End-to-end measurement of the latency depends on the ability to compare the actual physical motion of the headset to the display content updates. Thus the motion has to be observed by external, physical sensors and the content updates must be captured by an optical sensor. There are a few solutions, however, which implement an end-to-end measurement.

Zhao and Allison [23] describe a physical method of measuring the motion-to-photon latency. It is conceptually very similar to the optical flow based method employed in OptoFidelity VR Multimeter. It uses instrumented content where the display brightness varies with the tilt angle and a photodiode for measuring the brightness. The HMD is put in sinusoidal motion using a pendulum, which has a potentiometer on the joint for giving an absolute reference of the rotation angle. The signals from the potentiometer and photodiode are measured, and the phase shift between these is obtained using the Fourier transform.

Raaen et al. [24] also proposed a simple method for measuring the motion-to-photon latency. It essentially measures the latency between a small change of the HMD orientation and a corresponding change in the VR content. The method is focused on the small, jerky motions and can give good one-shot estimates of latency, but it is not suitable for measuring the latency as a function of time.

Oculus has developed their own latency tester [25], targeted at content developers using the Oculus Rift Development Kit. As such, it is not as universal as the other solutions and serves a niche. Also, it seems it only actually measures the latency from the graphics engine to the display (application-to-photon latency). It does not contain a motion sensor so it is not able to provide actual motion-to-photon latency readings. In a similar spirit, Basemark ships a solution called VRTrek, which also measures application-to-photon latency but is not specific to Oculus Rift and measures the latency for both eyes [26].

## 2.4 Designing a more advanced testing solution

Many measurable parameters of VR HMDs have been discussed in this chapter. It has become clear that while tracking technology is advancing in a good pace, there are still challenges to overcome. However, as of writing the thesis, there are no commercial solutions available for benchmarking temporal parameters such as drifting or stationary jitter. Moreover, none of the listed solutions cover benchmarking in full six degrees of freedom. There clearly is demand for such testing capabilities.

OptoFidelity's current VR Multimeter is a capable tester for the parameters it has been designed to test, most importantly motion-to-photon latency in one degree of freedom. However, there are several shortcomings in the design that prevent it from being directly expanded for more complex measurements. Specifically, the image sensor used there, ADNS3080, sets some limitations. Firstly, it is not possible to precisely trigger the exposure of a frame. Secondly, the frame size is limited to $30 \times 30$ pixels and the maximum image transfer speed 2 MHz is also low. Therefore the sensor is not particularly well suited for demanding computer vision tasks or synchronizing to low persistence screens. Finally, the product is marked as obsolete by the manufacturer.

Therefore, OptoFidelity set out to design a completely new tester. The goal is to establish a flexible software and hardware architecture with distributed processing, which can be easily adapted for a multitude of use cases. The mechanics of the tester are designed such that the robot is able to replicate human-like motions in six degrees of freedom. A novel optical measurement instrument with a camera sensor synced to display refresh has also been designed, sharing a common code base with the Video Multimeter. The focus of this thesis work is in the design of the camera module, temporal measurement methods and algorithms and the software architecture.

# 3 Tester hardware and architecture

While the 1 DOF tester has served its purpose well, it has turned out that especially with the advent of augmented reality devices, customers have widely varying measurement needs. For example, there might be a need to perform motions in full room scale, lengths of the axes extending to several metres. In another case there might be a particularly challenging machine vision task requiring special camera arrangements.

However, common to all cases is the desire to repeat human-like, even abrupt, motions in six degrees of freedom with high accuracy. That aspect has been taken in account in the design of the robot mechanics and control.

These requirements were used as guiding principles in the design of the 6 DOF system. The thesis work did not focus on the robot mechanics and control but they are briefly covered here.

## 3.1 Robot stages and mechanics

The requirement of repeating human-like motion sequences define and restrict the mechanical design of the robotics, particularly the choice of kinematics. A very common kinematic in industrial robotics is a robot arm with several rotational stages. The main benefit of such a kinematic is smaller size (and therefore potentially lower price) compared to a corresponding Cartesian robot, i.e. a robot where the motion of the furthermost point (end effector) takes place along the Cartesian $x$, $y$ and $z$ axes.

However, thinking of the application, human-like spatial motion is easily represented in three Cartesian coordinates. It might be necessary to quickly and smoothly interpolate between two arbitrary poses running at relatively fast speeds. This is where the shortcomings of robotic arms are seen — there are some poses of the robot where one can not move smoothly to an arbitrary direction. These points are called singularities and are also seen in the mathematical description of the kinematics.

A Cartesian robot, on the other hand, does not have singularities. The kinematic calculation and trajectory planning is simple compared to a robotic arm. Also, one

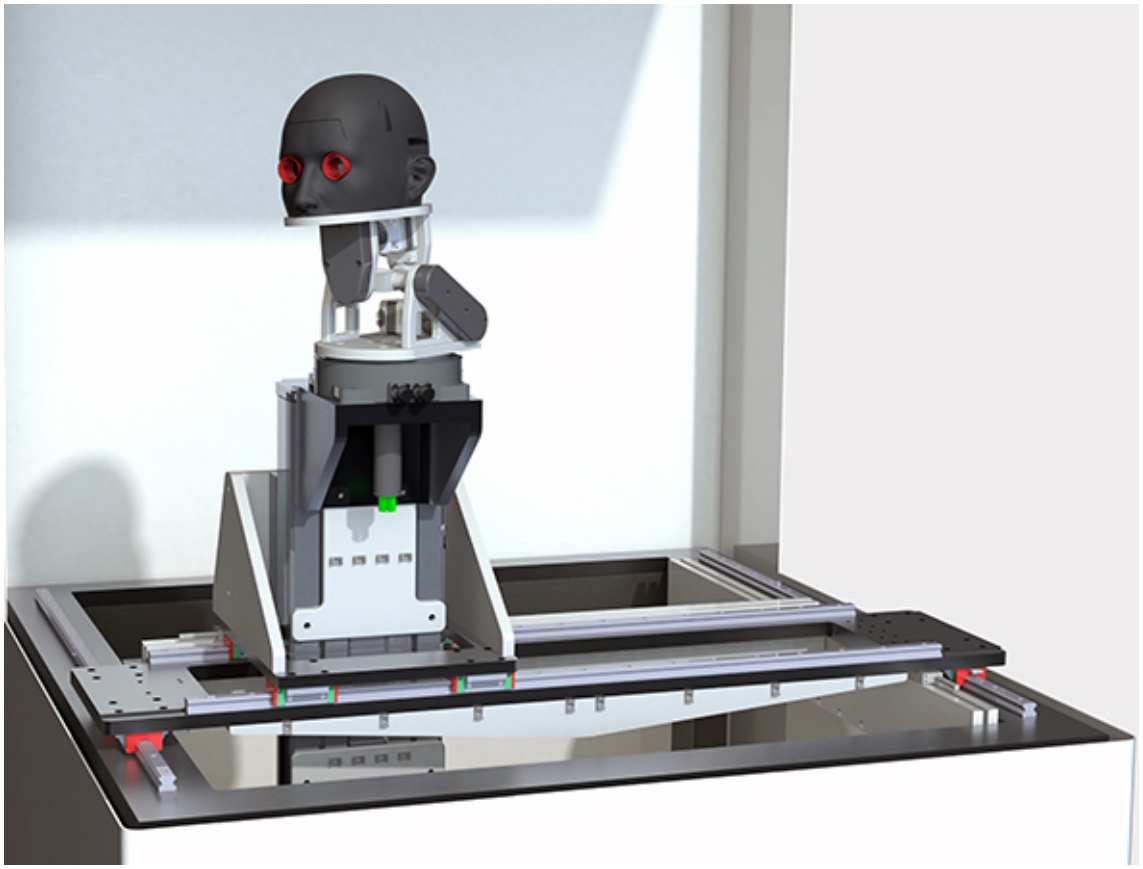


**Figure 5.** A three-dimensional rendering of the 6 DOF robot mechanics. © OptoFidelity, 2017.

can quite likely achieve accurate movements at greater speeds than with a robot arm. The Cartesian kinematic makes it simple to scale the robot to greater dimensions. This is why a Cartesian base was chosen for the reference design.

The three rotational degrees of freedom are implemented as a gimbal mounted in the endpoint of the Cartesian kinematic. The extents of its motion are chosen such that it can turn the head like a human would do. The construction consisting of the Cartesian base and a gimbal makes it also easy to get motion feedback from encoders on all the six axes. The whole assembly is shown in figure 5.

Despite the advantages gained with the choice of the Cartesian kinematic, no motion controller or motor driver can achieve perfect trajectories. There will always be deviations from the modeled trajectory due to the physical limitations of the robot. However, for the purposes of measurements and data analysis, it is not necessary to achieve arbitrary motion accuracies. Rather, only a good enough baseline motion accuracy is required to ensure the repeatability of motion sequences and to meet

the customer's requirements for the measurement. For measurements it is most important to have accurate, real time information about the end point location. This information is readily provided by encoders which are built in the system for closed loop motor control on every axis.

## 3.2 Data acquisition and processing architecture

The 1 DOF OptoFidelity VR Multimeter setup is architecturally simple. At the core of the system there is the Video Multimeter, which takes care of sampling the RGB sensor for synchronization, triggering image readout and acquiring data from the image sensor and shaft encoder which share the same Serial Peripheral Interface (SPI) bus.

In the 6 DOF system, the overall complexity is greater. There are six encoders in total in the system, and even more could be added in the future. The computer vision tasks such as drifting and stationary jitter measurement will also require greater resolution than what is used in the 1 DOF system. Therefore, demand for both transfer and processing bandwidth increases. Also, the physical distances between encoders, camera and the data acquisition hardware become greater, in the range of several metres. This increases the risk of interference in the data lines, especially in the environment where the robot stage motors emit electromagnetic interference.

### 3.2.1 Communication buses

The data flow has to be carefully designed to minimize interference. While in short-range connections one can usually use single wires carrying logic level voltage signals (with a common ground reference), longer wires are sensitive to interference and also suffer from voltage drop along the wire. Therefore, long distance data transfer is usually done on a differential, twisted pair of wires where there is a closed circuit between the two endpoints and the signal level is determined from the current instead of voltage. Common examples of differential data transfer buses are RS-485 (driven by a Universal Asynchronous Receiver Transmitter (UART) controller), Controller Area Network (CAN) and Universal Serial Bus (USB) 1.1.

The encoders signals are also transferred in differential mode. In a typical quadrature encoder, there are two differential signal lines emitting pulses when moving the encoder axis. Thus, transferring the pulses from the encoders should

not pose a problem in terms of interference. These signals can be wired to a pulse counter located close to the host processor.

Before the choice of a particular transfer bus can be made, the architecture and topology of the data flow have to be considered. In the case of the 1 DOF measurement, there is a clear master-slave architecture where the Video Multimeter is the master and the image sensor and encoder counter are slave devices. This arises naturally because the Video Multimeter communicates with both devices via a shared SPI bus, which inherently has a master-slave architecture.

The original intent for the 6 DOF system was to also use the Video Multimeter as the master processor. The evaluation of communication buses was thus largely based on the capabilities of the microcontroller used in the Video Multimeter, STM32F407 [27]. It was decided that the most demanding, real time processing of raw image and encoder data would be offloaded to the slave devices, therefore implementing a revised version of the master-slave architecture of the 1 DOF measurement. This relaxes the processing power and transfer speed requirements set on the Video Multimeter — when the raw data is processed locally on the slave, one can transfer processed metadata on the system bus, consuming less bandwidth.

UART and CAN are widely adopted protocols for serial communication between computers and microcontrollers. Both of them define a packet format, but unlike UART, CAN also specifies the properties of the physical layer, i.e. the electrical properties of the connection between devices. For UART, there are a variety of physical layer specifications, RS-485 being a differential one designed for long distance communication. The CAN implementation on STM32F407 is capable of transferring 1 Mb/s and the UART implementation 10.5 Mb/s.

USB 1.1 is a commonplace protocol especially in the personal computer and consumer electronics markets. It implements a serial data bus using a differential pair for half-duplex data transfer. The protocol makes a clear distinction between host and slave devices, the PC usually taking the role of the host. Slave devices only write to the bus upon request by master. Many microcontrollers implement both USB host and slave interfaces in their hardware, so implementing a USB-based sensor network should be feasible. The advantages over UART and CAN include higher transfer rate (12 Mb/s), clear architecture and easy connectivity with the help of USB hubs. Greater bandwidth brings some room for future modifications, possibly adding more devices and data on the bus.

On the other hand, a disadvantage of USB is the increased complexity — the architecture of the USB standard itself has a hierarchy consisting of entities like interfaces and endpoints on the device side. There are many different standardised communication protocols that can be implemented over USB, such as the Human Interface Device specification for mouses and keyboards, standards for audio and video sources, and several network emulations. It is desirable not to implement one's own protocol but rather choose an existing, widely supported one. This way, writing a host side low level driver can be avoided.

After a careful comparison between the various options for the communication bus, the decision was made to use USB 1.1 and a personal computer as the master. While the intent was to use the Video Multimeter as the master device, it was recognized that the 6 DOF measurements might be too complex to implement on such a small device, partially because of the small display. Also, in the 1 DOF system a PC is often used as the actual master already, sending remote control commands to the Video Multimeter. A PC environment gives more flexibility in terms of software development, allowing e.g. the usage of OpenCV and other helpful but heavyweight libraries. The 12 Mb/s transfer speed of USB 1.1 provides some headroom for future additions. The choice of USB 1.1 was also motivated by its widespread presence in embedded processors from small microcontrollers to tablet computer processors — for example, a tablet could be used as the measurement master.

For the communication protocol, the Microsoft RNDIS specification was implemented. It is a standard that implements Ethernet emulation over USB, thus making a connected device addressable with a unique IP address. The communication between the master and sensors happens via HTTP requests which may contain data in the common JSON format, for example. This choice was made for convenience — HTTP requests and the JSON format are readily supported by the standard libraries of many programming languages. Thus the protocol does not impose strong restrictions on the implementation of the master PC software.

Another common communication standard, Ethernet, was not included in the initial comparison of communication buses. It was ruled out in the beginning because of the intention to use the Video Multimeter as the master, which does not have an Ethernet port. STM32F407 itself supports Ethernet by providing an interface for connecting an external Ethernet physical layer (PHY). Since Ethernet emulation over USB was chosen as the communication method, the software uses the actual

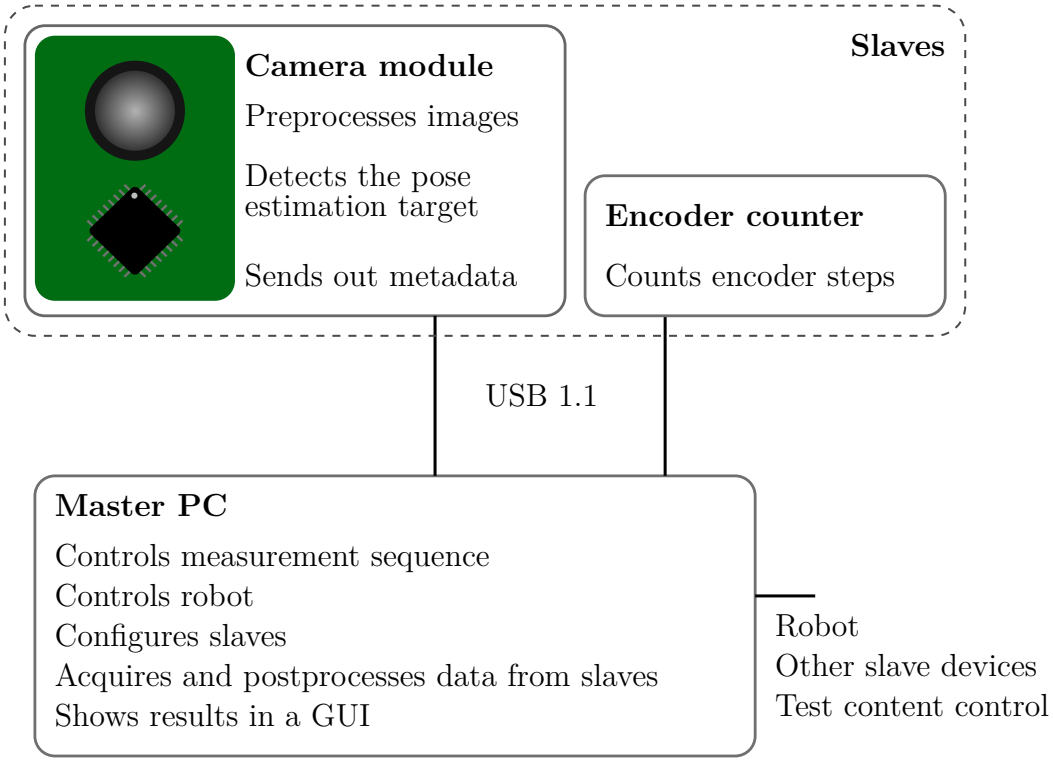



**Figure 6.** An illustration of the measurement architecture of the 6 DOF tester.

networking components of the underlying operating system. Therefore, very few software modifications would be required if one would desire to switch to Ethernet as the actual communication standard later on.

### 3.2.2 Measurement control and synchronization

In the master-slave topology, the master takes the responsibility of controlling the flow of the measurement. This involves following a prescribed measurement sequence consisting of commanding the robot, sensors and the instrumented content displayed on the device under testing. The master also takes care of acquiring and post-processing the measurement data from slave devices. The architecture is illustrated in figure 6.

During the measurement sequence, the master must also ensure that the actions performed by the different slave devices happen synchronously. The slaves must report their own state and give an indication to the master when a certain action has been completed.

To perform accurate temporal measurements, it is essential to ensure that the

slave device clocks are synchronized to a common time base so that the timestamps of samples from different devices can be compared. The slaves have their own onboard crystal oscillators providing the base clock frequency for the microcontroller. However, due to variations in temperature and other physical conditions, clock drift is inevitable. Therefore the master must regularly synchronize the slave clocks to its own clock. The robot motor controller itself does not have to share the time base with the master — the actual motion data is acquired from the encoder counter.

The USB specification already contains a synchronization mechanism called start of frame packets. These are packets that are broadcast by the master to all devices at 1 ms intervals and have a frame number as the payload. This provides the slave with sufficient information to synchronize its clock to the master.

# 4 Design of the smart camera

Due to the special nature of virtual reality displays described in section 2.1, particularly the low persistence, it is not trivial to capture images of the display content, even given a high speed camera. Because the persistence times are usually in the range of a few milliseconds, simple free running video capture is not sufficient.

The requirement of synchronizing the image capture with display frames gives rise to the idea of a programmable smart camera module with a microcontroller controlling the image capture. Given proper sensors, one can detect the rising edges of the display backlight and trigger the image capture precisely synchronously with them. In OptoFidelity Video Multimeter, there is a backlight detection method based on a tristimulus light sensor and a realtime edge detection algorithm that are used for video playback measurements. The same technology can also be utilized in the synchronization of image capture, the on-board microcontroller providing trigger pulses to the camera sensor from its general purpose input/output (GPIO) pins.

Additional requirements on the camera are imposed by the frame rate of the device under testing. As the technology evolves, the display frame rates tend to increase, common values currently being 90 Hz and 120 Hz. To capture images at the full frame rate, image integration and readout both have to happen in sufficiently short time. Thus there is a trade-off between image resolution and frame rate, limited by the transfer rate of the camera bus. It is important to choose a camera sensor and a processor that are capable of transferring image data fast enough.

The transfer rate of an image sensor is usually expressed as megapixels per second (Mpx/s). As an example, if the transfer rate of a sensor is 27 Mpx/s and one desires to capture 120 images per second, the maximum number of pixels per frame is

$$N = \frac{27\,\mathrm{Mpx/s}}{120\,\mathrm{s^{-1}}} = 225\,\mathrm{kpx}.$$

This would equate roughly to a $470 \times 470$ pixel square image.

**Table 2.** Several commercial smart camera modules along with their sensor and processor.

| Board | Sensor | Processor / core |
|---|---|---|
| PX4FLOW [28] | OnSemi MT9V034 | STM32F407VG<br>ARM Cortex M4 MCU |
| JeVois [29] | OmniVision OV9653 [30] | Allwinner A33<br>ARM Cortex A7 SoC |
| CMUcam5 [31] | OmniVision OV9715 | NXP LPC4330<br>ARM Cortex M4/M0 MCU |
| OpenMV Cam M7 [32] | OmniVision OV7725 | STM32F765VI<br>ARM Cortex M4 MCU |

## 4.1 A review of camera modules

The idea of a smart camera module is not new — there already is a good offering of single printed circuit board (PCB) devices with an image sensor and a processor in the market. Hence, before starting implementation of a new camera module, a review of existing solutions was done.

A few existing, popular camera modules are listed in table 2. They all share a small size and low power consumption. Also, three out of four are using a camera sensor from the same manufacturer, OmniVision. All the listed modules look promising and suitable for implementing the previously described imaging and measurement. However, there are some clear differences between the modules which are discussed further.

PX4FLOW, CMUcam5 and OpenMV Cam M7 are using a microcontroller for image acquisition and processing. These devices are low-powered and run at a CPU clock frequency of 100 MHz to 200 MHz. The JeVois smart camera, on the other hand, differs from the others by its use of the more powerful Allwinner A33 system-on-chip processor. It runs Linux by default, whereas the other modules have a custom firmware. While this might sound favourable, the hardware is also more complex to operate, especially if one desires to write a custom firmware. Also, a simpler device might be preferable for implementing a special purpose instrument such as the planned smart camera. For prototyping and experimentation with various computer vision algorithms, the JeVois smart camera could be ideal, supporting OpenCV and an extensive set of ready-to-use computer vision code.

On the other hand, PX4FLOW and OpenMV Cam M7 use STmicroelectronics'

ARM microcontrollers, one of which is also used in OptoFidelity Video Multimeter. The choice of an already known platform would thus be beneficial considering reuse of code and know-how. There is a wealth of community support for these devices, including ports of many real-time operating systems such as NuttX [33] (the one used in Video Multimeter). PX4FLOW and Video Multimeter even have the same processor model, so they could run the same firmware, which gives the benefits of a known and tested code base. Otherwise OpenMV and CMUcam5 look more compelling, both breaking out a good set of GPIO pins that one could use to attach external sensors.

At the time of doing the comparison, there wasn't significant difference in the pricing and availability of these four modules. PX4FLOW was picked as the development platform due to the ability to run the Video Multimeter firmware on it and share the code base.

## 4.2 A review of image sensors

PX4FLOW was used for "bootstrapping" the project, serving as an initial development platform for the camera firmware. It was recognized that it would not be the final solution — retrofitting the tristimulus sensor requires manual work and the resulting assembly is not as clean as a custom PCB. Therefore, a review of image sensors was also done in order to choose a sensor for use in a custom camera design based on STM32F407 or a similar microcontroller.

There is a good selection of small, machine vision oriented CMOS image sensors available, some of the major manufacturers being OmniVision and ON Semiconductor, for example. The prices of discrete sensors are in the range a few tens of euros even in low quantities, making designing one's own camera module a feasible option.

A common problem with commercial sensors is the lack of proper, open documentation. Manufacturers usually make a general specification sheet publicly available but more specific documentation of the functionality and control registers is treated as confidential. This would mostly pose a problem if one was developing an open source driver for the camera, thus disclosing the details via the source code. However, that is not the case in this work.

An important detail regarding camera sensors are the configuration and data transfer buses of the sensor. The host processor has to be able to communicate via these buses and additionally use direct memory access (DMA) on the image

transfer bus, freeing the CPU from the expensive task of transferring the images. The Inter-Integrated Circuit (I2C) bus is a common configuration interface in the camera modules, providing access to the sensor's registers. I2C is readily supported in hardware by most microcontrollers, including STM32F407.

Several buses exist for image transfer, including a simple parallel interface and several kinds of serial interfaces. In the parallel interface, there is a dedicated pin for each bit of a pixel, i.e. in 10-bit sensors there are 10 data pins. Additionally, there is a pixel clock signal emitted by the sensor, indicating the availability of a new pixel by a rising or falling edge. Often, horizontal and vertical synchronization signals are provided, indicating the start and end of each data row and frame, respectively. STM32F407 has a digital camera interface (DCMI) which is designed for receiving data from such parallel interfaces. Direct memory access is available for the interface, making it feasible to actually transfer the images to the processor's memory.

The serial interfaces, on the other hand, are usually more complex, having a custom packet format for transferring the data. The electrical and physical properties of the transfer bus are specified by a physical layer standard, two common standards being low voltage differential signaling (LVDS) and the MiPi camera interface [34]. Both of these are based on transferring the data on a differential pair of wires, reducing interference and making for high transfer rates. However, support for these is not readily available in the STM32F407 hardware.

Resolution and frame rate are also parameters that have to be considered. However, as demonstrated earlier, the actual limits of these parameters in most cases are set by the speed of the image transfer bus. STM32F407 sets a hard limit to this, the DCMI hardware having a maximum transfer rate of 54 Mpx/s [35]. It was also noticed that the intended host processor STM32F407 has only a limited amount of memory available, making it difficult to store large images. Therefore it does not make sense to aim for a very high resolution.

In this application the camera is used for imaging electrical displays which usually are globally refreshed, therefore making a global shutter unnecessary. However, such a camera could be useful for other machine vision applications where the lack of global shutter would cause undesired artifacts like demonstrated in figure 7.

Table 3 lists several global shutter, small resolution image sensors that were considered for the application, along with their most important parameters.

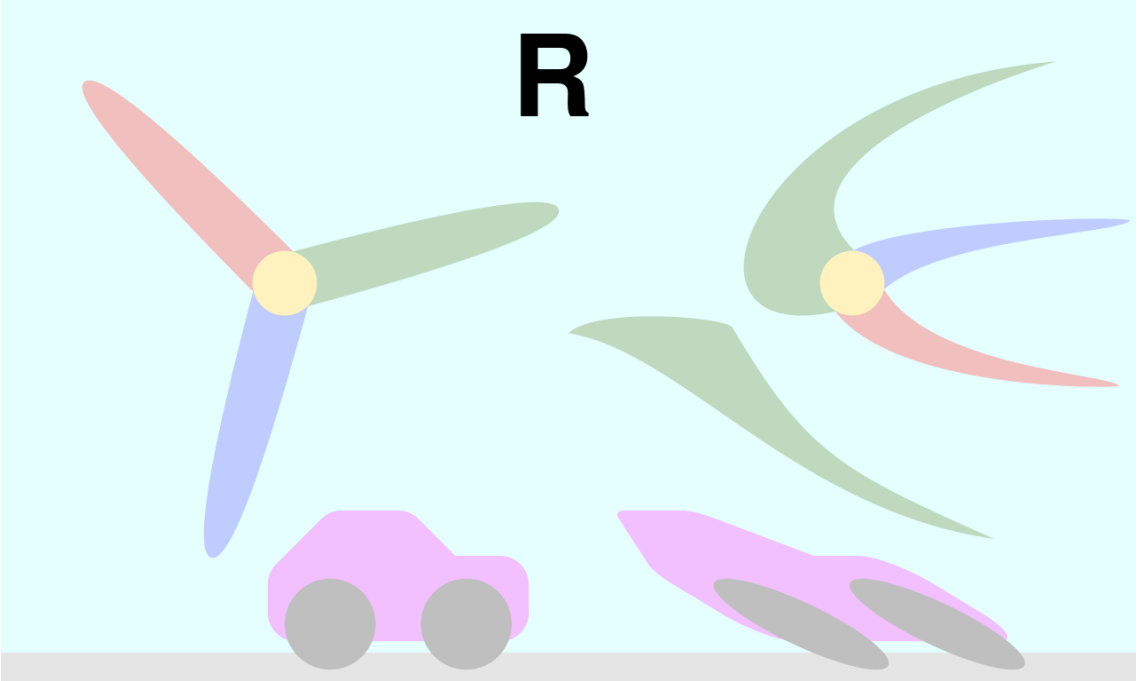



**Figure 7.** A simulated example of an artifact caused by a rolling shutter. In the simulation, the car is driving to the right and the propeller is rotating. The left image has been captured with global shutter, right with rolling shutter. Image courtesy of CMG Lee, licensed under CC BY-SA 3.0 [36].

**Table 3.** A set of image sensors considered for the smart camera application, along with their important specifications. All sensors have global shutter.

| Sensor | Resolution | Config. bus | Data bus | Data rate |
|---|---|---|---|---|
| OnSemi MT9V034 [37] | 752 × 480 | I2C | parallel, LVDS | 27 Mpx/s |
| OnSemi LUPA1300-2 [38] | 1280 × 1024 | SPI | LVDS | 630 Mpx/s[1] |
| OmniVision OV9282 [39] | 1280 × 800 | I2C | parallel, MiPi | 123 Mpx/s[2] |

Of the listed sensors, LUPA1300-2 offers the highest transfer rate and resolution. However, it also has a high pin count of 168 and requires a 315 MHz system clock [38], making it more demanding to use and operate compared to the other sensors which have much lower pin count and a system clock frequency of only 27 MHz. While LUPA1300-2 is a powerful device, its capabilities would be partially left unused due to the host processor not having hardware capabilities to communicate via the fast LVDS bus.

[1]The LVDS has 12 data lanes, each clocked at 630 Mb/s [38].

[2]Calculated according to specified transfer rate of 120 frames per second at 1280 × 800 pixel resolution [39].

OmniVision OV9282 and ON Semiconductor MT9V034 look similar to each other in terms of basic parameters. However, little information about the former is available in the publicly released product brief [39] while the public MT9V034 datasheet [37] packs a good amount of technical information on registers and configuration. Due to little information being available, it is even hard to tell if OV9282 even has an external trigger input (although the FSIN pin might designate that).

After the evaluation, MT9V034 seemed to be the best alternative due to a couple of factors. Firstly, it does not have an unnecessarily large resolution, making the pixels physically larger and more sensitive. Secondly, it has a global shutter. Moreover, its communication interfaces are simple (I2C and parallel interface) and it comes in a package that it easy to solder even by hand. Additionally, it has been already tried and tested in combination with STM32F407 in the PX4FLOW camera module, performing optical flow tracking. Based on this, the combination of resolution, transfer rate and processing power was considered sufficient for the computer vision tasks in the VR measurement system. In the final camera design, STM32F407 was replaced by a compatible component, STM32F427, which has fixed some of the silicon errata of STM32F407 among other things.

## 4.3 Liquid lens and autofocus

One of the issues of the camera system in the 1 DOF tester is the difficulty of focusing the lens. There is no autofocus system, so the operator has to manually focus the lens by turning it in the lens mount. In practice this is difficult because physically accessing the lens is inconvenient when the tester is assembled. For the new camera, automatic focusing capability was taken as a design goal.

The chosen camera sensor MT9V034 has a diagonal of 1/3 inch [37]. Practically, this also leads to the use of a small lens. A common option for sensors of this size is the S-mount, sometimes called M12 mount after its standard metric thread size. Compatible lenses are common and easy to acquire in many different focal lengths and coatings.

In conventional, larger cameras, focus control is usually realized by a motorized mechanism actuating a lens or a group of lenses in the optics. However, in the size of M12 lenses, such a mechanism is impractical to implement. Therefore a technology called “liquid lens” has been invented. In a liquid lens, there is a cylindrical container with two immiscible liquids and their interface forming a refractive element. One of

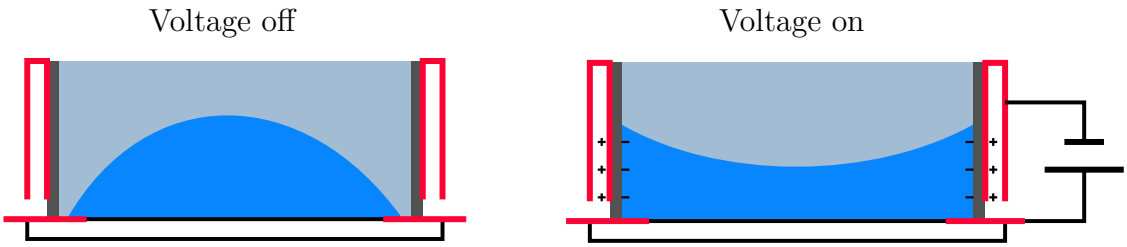



**Figure 8.** Cross section of a liquid lens element with and without voltage applied, illustrated after [40]. When voltage is not applied, the conductive liquid (blue) forms a drop on the bottom of the element, with the insulating liquid (light grey) on top. When voltage is applied to the electrodes (red), charges are accumulated on both sides of the wall insulator (dark grey) and the conductive liquid is attracted toward the walls, changing the shape of the interface between the liquids, thus changing the optical power of the lens.

the liquids is conductive, the other an insulator. When a voltage is applied between the conductive liquid and the cylinder wall, the liquid is drawn towards the walls and the shape of the refractive interface is altered, thus changing the focal point of the lens. This process is called electrowetting and is illustrated in figure 8. By adjusting the voltage, one can change the focus distance, practically making the construction a voltage adjustable lens. [40]

There are commercially made S-mount liquid lenses available, such as the C-S series of lenses by Corning Varioptic [41]. These lenses feature a connector for directly supplying the control voltage to the focus element. To cover the physically achievable focus range of the lens, a maximum voltage of about 100 V is typical [40]. On the smart camera, this is realized by a boost converter that is controlled by the microcontroller. This way the focus point can be controlled by the camera firmware for autofocusing purposes.

### 4.3.1 Designing an autofocus algorithm

The design of an autofocus algorithm is strongly dependent on the application. For example, if one has to lock focus on a moving subject, particularly sophisticated methods are required for selecting the region of interest and estimating the direction of motion. In some scenarios there might also be several subjects at different distances from the camera, resulting in multiple possible points of focus. It might be desired that the autofocus algorithm could resolve between these sharpness peaks and lock to one of them, presenting another algorithmic challenge.

In the case of the smart camera, the target is the VR display, which makes the autofocus implementation relatively simple. There is only one planar subject, the display, which remains at a fixed distance from the camera throughout the measurement. It is sufficient to perform a one-shot autofocus in the beginning of the measurement sequence to maximize the sharpness of the image.

The simplest way to perform one-shot autofocusing is to linearly sweep over the full focus range and take images at fixed intervals of the focus control voltage. Using these images, one can estimate their sharpness and choose the focus point where the greatest sharpness is achieved. This way, the true global maximum of sharpness is found.

It is desirable to evaluate the image sharpness using a region of interest smaller than the entire image. A subset of the pixels at the center of the image is a natural choice — for machine vision algorithms, it is usually desirable to achieve maximal sharpness at the center of the image. This choice of a region also eliminates edge effects (vignetting, for example) that might be caused by the optics or the sensor. In the following discussion of sharpness metrics, therefore, the term *image* designates the preselected, smaller region of interest of the source image.

There are many ways to estimate image sharpness, ranging from typical gradient-based approaches to methods relying on the image histogram or the statistical properties of the image. A comprehensive, comparative study of sharpness metrics has been conducted in [42]. Although the study focuses on digital photography applications, the results are interesting also from the point of view of this application. In the study, the gradient or derivative based methods generally achieved the best accuracy, led by a method originally proposed by Brenner et al. [43] that is defined by the equation:

$$S = \sum_{x=0}^{M-1} \sum_{y=0}^{N-3} (f(x, y+2) - f(x, y))^2, \tag{3}$$

where $S$ is sharpness, $f$ is the image, $x$ and $y$ are the horizontal and vertical image coordinates and $M$ and $N$ are the width and height of the image, respectively. The equation essentially presents the sum of squared image derivatives in the vertical direction over the image. [42] The step of 2 pixels (instead of 1) in the difference calculation is used to reject pixel level noise, which might result in unrealistically high sharpness numbers. From the original equation (3), one can also separate the

local sharpness estimator:

$$S_{\text{local}}(x, y) = (f(x, y + 2) - f(x, y))^2. \tag{4}$$

For this application, a different sharpness estimator based on the horizontal and vertical Sobel operators was devised. While the estimator defined by (3) yields an estimate of the overall sharpness, for this algorithm the goal was to make it sensitive to small sharp details, focusing on the one with the highest sharpness.

The horizontal and vertical Sobel operators use convolution kernels which, when convolved with an image, approximate horizontal and vertical image derivatives. They also incorporate an averaging effect, giving the approximation some resistance to noise. The Sobel operators are defined as:

$$\mathbf{G}_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * \mathbf{A}, \mathbf{G}_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * \mathbf{A}, \tag{5}$$

where $\mathbf{G}_x$ is the horizontal and $\mathbf{G}_y$ the vertical Sobel operator and $\mathbf{A}$ is the image. Applying the both operators on the image, one gets a vector field

$$G(x, y) = (G_x(x, y), G_y(x, y)),$$

corresponding to the local gradient of the image. The domain of the operation is restricted such that a one pixel wide strip is left off the domain on each image edge. This works around the fact that one cannot compute the convolution at these points based on existing image data without making assumptions of the image data outside the boundaries (e.g. assuming values to be zero). This choice does not affect the sharpness estimation results in any significant way but avoids artifacts that could be caused by extending the image at the boundaries.

Further, one may calculate the squared local gradient magnitude based on the Sobel operators as

$$S_{\text{local}}(x, y) = \|G(x, y)\|^2 = G_x(x, y)^2 + G_y(x, y)^2. \tag{6}$$

The squared magnitude is used to save the computation cost of the square root function — as an increasing function, it would not have an effect on the location

of the extrema. This equation serves as a local sharpness estimator, with its value corresponding to the edge steepness at the particular point $(x, y)$. It should be noted that in contrast to (3) which performs the differentiation only in one direction, this estimator accounts for edges in any direction.

Then, some method is needed to convert the scalar field $S_{\text{local}}$ to a single scalar number that describes the image sharpness. One method, as used in (3), is to sum over all the local values. However, as the intent was to make the estimator sensitive to small, sharp details, taking the maximum of the local sharpnesses would be a more natural choice. Simply finding the maximum, however, is too sensitive to noise as a single, spurious large value of $S_{\text{local}}$ would override all the other values.

A simple way to obtain a more robust maximum is to use some high ($> 95$ %) percentile instead. For example, a 99.5th percentile of the distribution of $S_{\text{local}}$ values would return a value $S_{\text{global}}$ such that 99.5 percent of the $S_{\text{local}}$ values are less than $S_{\text{global}}$. This is easily realized by calculating the cumulative histogram of $S_{\text{local}}$ values and then looking up the percentile.

### 4.3.2 Characterizing the autofocus algorithm

In order to verify the sharpness estimation method and find out if the global sharpness estimation function works as desired, several tests were conducted. Most importantly, the noise sensitivity of the method was inspected as it affects the accuracy and repeatability of the autofocus.

A good method for testing of algorithm properties is to generate test data in a controlled manner by simulation. This way one knows the expected outcome of the algorithm in various scenarios and may compare it to the actual results. Therefore, in the first tests, a test photograph (figure 9) was blurred with various blur strengths, and its sharpness at each point was evaluated using different methods.

The simulations were done using Python and the NumPy and SciPy libraries for numerical processing. The 16-bit linear grayscale source image was first loaded, and a sequence of test images was generated from it. All the image processing and sharpness estimation were done in the linear space where the grayscale image was converted to a 64-bit floating point representation and scaled to the full scale range $[0, 1[$.

For each test image, the source image was first blurred by convolving it with the

**Figure 9.** The test image that was used in the sharpness estimation simulations. The representation in this document is in 8-bit sRGB space for display while the real source data is represented in 16-bit linear space.

two-dimensional Gaussian kernel:

$$G(x,y) = \frac{1}{2\pi\sigma_{\text{blur}}^2} \exp\left(-\frac{x^2+y^2}{2\sigma_{\text{blur}}^2}\right), \tag{7}$$

where $x$ and $y$ are the image coordinates in pixels and $\sigma_{\text{blur}}$ the standard deviation of the distribution, used for controlling the blur strength. A series of images was generated with linearly spaced values of $\sigma_{\text{blur}}$ between 0 and 3.5. This step was done to generate a set of data where the sharpness of each image relative to the others is known a priori.

The local sharpness of these images was estimated using the squared gradient magnitude and the Sobel operators as described by equations (5) and (6). From there, three different methods of obtaining a global sharpness estimate were used. The methods were simple maximum of local values, the 99.5th percentile and the mean of local values. To be able to compare the values between methods which might result in differently scaled values, the sharpness relative to the case where $\sigma_{\text{blur}} = 0$ was calculated by equation

$$S_{\text{relative}}(\sigma_{\text{blur}}) = \frac{S_{\text{global}}(\sigma_{\text{blur}})}{S_{\text{global}}(\sigma_{\text{blur}} = 0)}$$

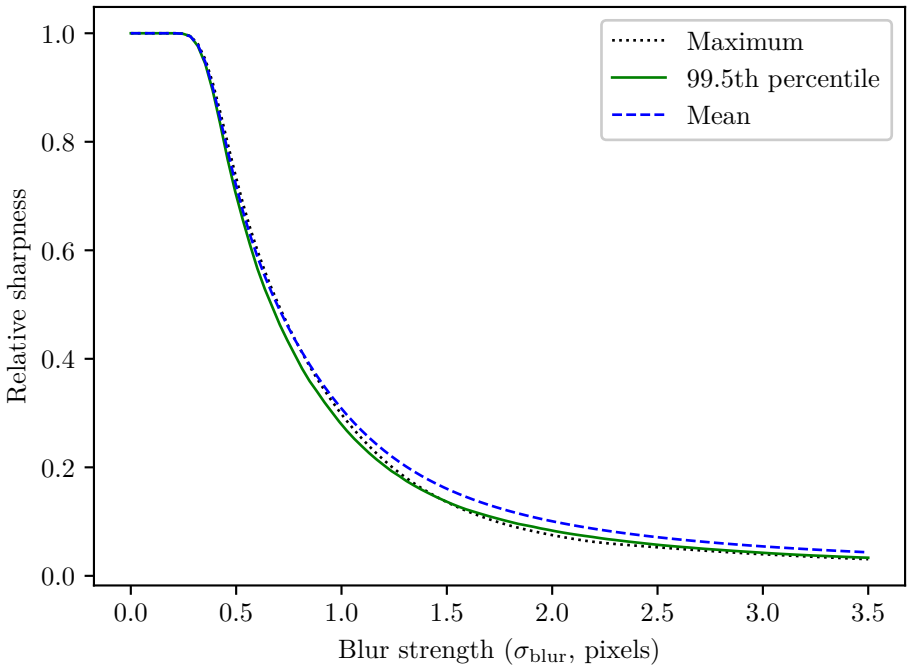



**Figure 10.** Estimated relative sharpnesses of the test image (figure 9) with different strengths of blur. No noise was added to the test images in this sequence.

for each global estimation method separately. The results are presented in figure 10. As it can be seen from the figure, in this simulated case the global estimation methods are mostly in agreement, with only slight differences between the sharpness estimates. The behaviour is as expected, $S_{\text{global}}$ being a decreasing function of $\sigma_{\text{blur}}$ in all cases. There is a small interval between approximately 0 and 0.3 where $S_{\text{global}}$ retains its maximum value. This is due to the small standard deviation of the Gaussian distribution and the neighbouring pixels getting practically zero weight in the discrete convolution. After that interval, however, all the functions rapidly decay toward zero, which is as expected.

This simulation, however, doesn't particularly well represent real images taken by a camera, since there is practically no noise in the test images. While the source image might contain some noise, it is mostly removed by applying the Gaussian blur. Therefore, to more realistically represent the situation with real images, random noise should be added to the test images after performing the convolution with the Gaussian kernel. This kind of noise could potentially affect the sharpness estimation results, especially if there are high variations of grayscale values introduced between adjacent pixels.

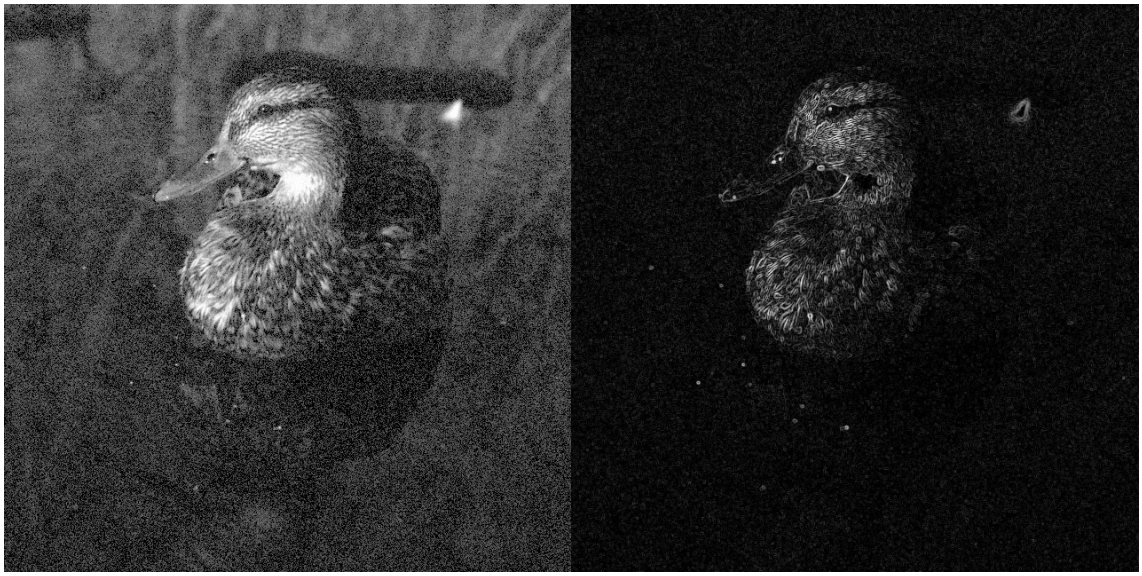


**Figure 11.** An example of a blurred image ($\sigma_{\text{blur}} = 0.5$) with added noise ($\sigma_{\text{noise}} = 0.05$) (left) and its local squared gradient magnitude values (right). The right picture has been converted from the linear space to sRGB for display.

A subsequent simulation was performed where Gaussian, pixel-level noise was added to the blurred images used in the first simulation. The Gaussian (normal) noise is described by the distribution function

$$f(t) = \frac{1}{\sigma_{\text{noise}}\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma_{\text{noise}}^2}\right), \tag{8}$$

where $t$ is the value of the noise field and $\sigma_{\text{noise}}$ is the standard deviation of the noise. For each pixel of each blurred image, a number $t$ was drawn from this distribution and added to the pixel value. The result was clipped to the full scale range $[0, 1[$. An example of an image with added noise and its squared gradient magnitude is shown in figure 11.

The results with $\sigma_{\text{noise}} = 0.2$ are presented in figure 12, accompanied with the 99.5th percentile curve from the noise free case (figure 10) as a baseline. One should note that the noise standard deviation of 0.2 is quite extreme, corresponding to 20 percent of the full scale range. A high value is intentionally used to make the differences between the methods clearly visible.

As expected, the simple maximum calculation suffers most from the noise, resulting in unusably inaccurate sharpness estimates. It is not surprising since any

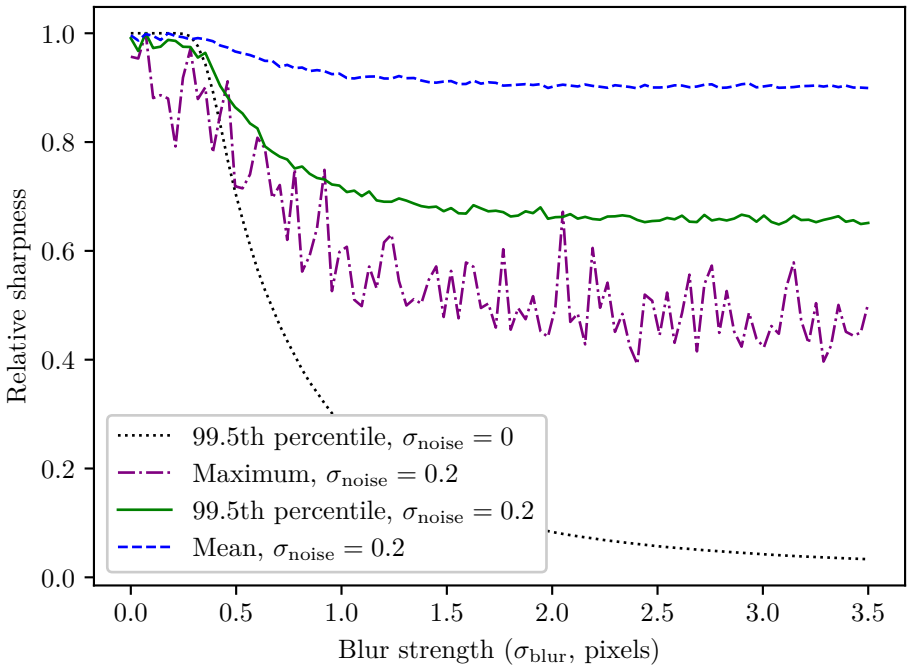



**Figure 12.** Estimated relative sharpnesses of the test image (figure 9) with different strengths of blur. Pixel-level normal noise with standard deviation $\sigma_{\mathrm{noise}} = 0.2$ was added to the images.

spuriously high local gradient value resulting from the noise can affect the sharpness estimate. The mean, on the other hand, doesn't seem to have as much noise due to averaging, but the relative sharpness decays to quite high a value, leaving the smallest ratio between the "sharp" and "unsharp" values.

The 99.5th percentile case is particularly interesting. Visually inspecting the curve, it settles between the mean and maximum curves while having significantly less noise than the maximum curve. The source image being $640 \times 640$ pixels, roughly 2048 pixels (0.5 percent) have a value above the 99.5th percentile. This seems like a good compromise for this source image, and the percentile method can be seen as a good choice, judging by the simulations.

An advantage of the percentile method is that it is tuneable. Lower percentiles result in less sensitivity to noise but also smaller ratio between sharp and unsharp values, higher percentiles result in behaviour close to the maximum curve and higher sensitivity to noise. One can select a suitable percentile depending on the source image size and amplitude of image noise.

For comparison, a similar test was conducted using the Brenner local sharpness

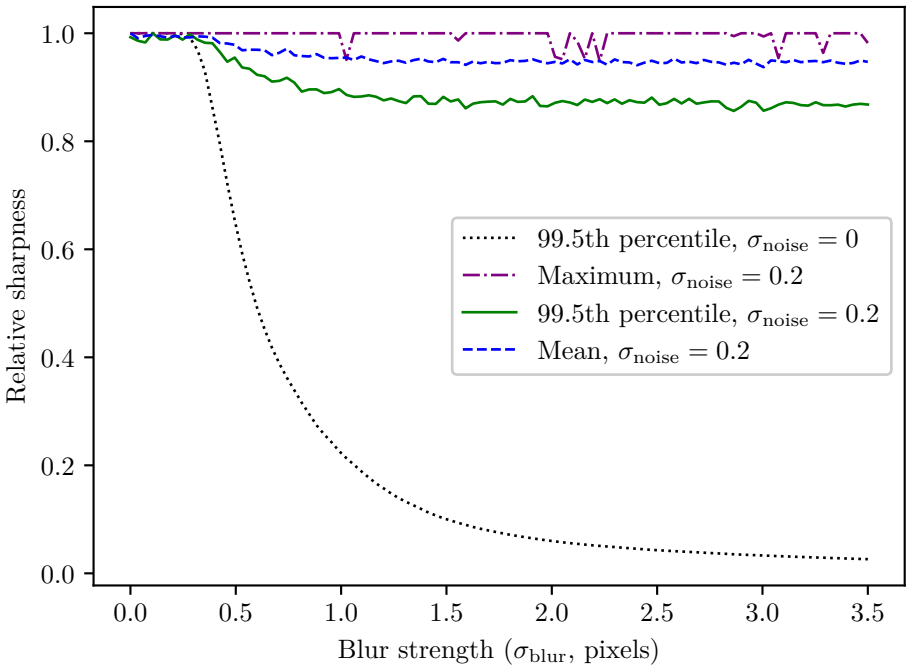



**Figure 13.** Comparison of global sharpness estimators similar to figure 12 with Brenner local sharpness estimator (4) instead of the Sobel magnitude (6).

estimator (4). The results are plotted in 13.

It is seen that compared to figure 12, the noise floor for all the global sharpness estimation methods is higher. Therefore the Brenner local estimator seems to be less tolerant to noise than the squared Sobel magnitude. This is not surprising since the Sobel kernels perform averaging to obtain the local sharpness values. There is also more information used for the local values in the Sobel magnitude method because both horizontal and vertical gradients are taken in account.

Finally, a test was done using a real-world target and the actual smart camera module. The camera was placed inside an artificial head which was facing a QR code printed on paper. The measurement setup is shown in figure 14. A QR code was chosen as the target because it would be also later used in actual measurements where the camera has to be focused to a display. The liquid lens control voltage was linearly swept through the full range from 24 to 70 volts, divided in 100 equally sized steps. At each step, an image of the QR code was taken and downloaded to the controlling PC.

For the captured images, sharpness estimates were obtained using the Sobel squared magnitude and 99.5th percentile method described above. The relative

**Figure 14.** Measurement setup used for testing the sharpness estimation in practice. The smart camera module is mounted inside the artificial head.

sharpness as a function of liquid lens control voltage is plotted in figure 15. The plot generally shows that the sharpness is well-behaved with only one maximum and smooth, quick decay to almost zero after the maximum. However, in the lower end of the control voltage values, there is slower, less well-defined rise and then an abrupt jump to a higher value. The exact reason for this behaviour is unknown but it is assumed to be due to the liquid lens not properly functioning at such low voltages.

Unfortunately, a ground truth measurement could not be performed since the relationship between the lens control voltage and focus distance was not known. Therefore, the correctness of the maximum was visually inspected, comparing the perceived sharpness to a few adjacent focus points. The maximum was found at 32.28 V. Images for 30.44 V, 32.28 V and 34.12 V are shown in figure 16. Judging from the images roughly two volts apart from the maximum, the one at the maximum seems the sharpest — thus, it can be concluded that the sharpness estimator works as designed.

The linear sweep autofocus algorithm has also proven to work well in practice. The STM32F427 processor calculates the Sobel operators and the percentile quickly enough, run time being mostly limited by liquid lens control voltage settling time at each step. The focusing has been found accurate and suitable for the one-shot focus use case it was designed for.

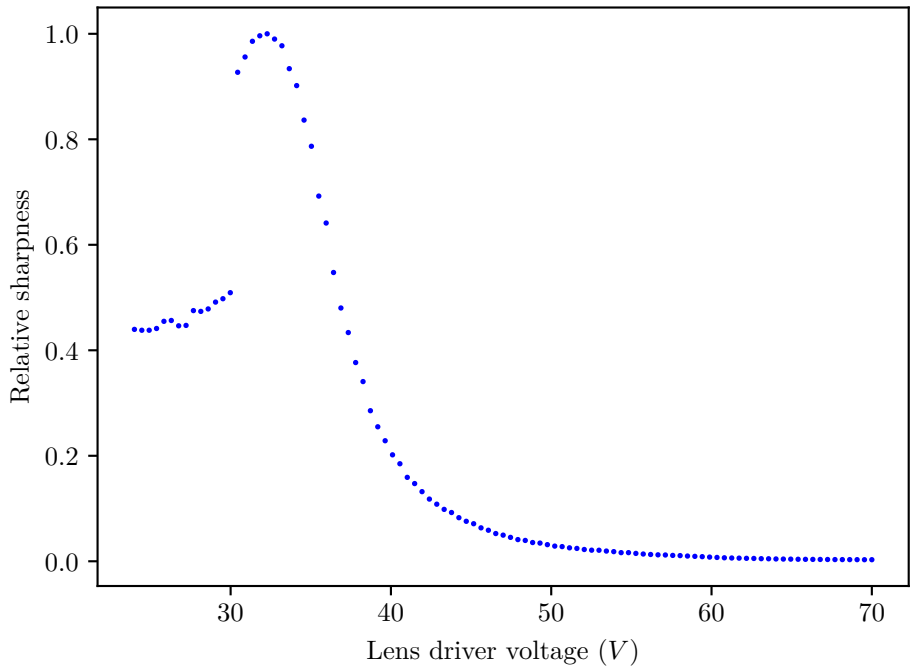



**Figure 15.** Relative sharpness as a function of the liquid lens control voltage, with a printed QR code as the target. Global sharpness was estimated using the 99.5th percentile of local squared gradient magnitudes calculated by (6).

**(a)** 30.44 V **(b)** 32.28 V **(c)** 34.12 V

**Figure 16.** Images of the QR code at voltages below, at and above the voltage at the sharpness maximum (32.28 V). Images have been scaled up from the original 120 × 120 pixel size for easier inspection.

# 5 Camera pose estimation from display

For the 6 DOF tester, there are several test cases planned that measure the tracking accuracy from various aspects. In some measurements, the intent is to measure the drifting and jitter of the tracking as a function of time, stimulated by sudden robot movements.

For a true end-to-end measurement, information about the status of the head tracking has to be acquired from the display. Specifically, measurements of the full pose is desired, based on the 3D content that is shown on the HMD. The pose has six degrees of freedom and is represented by the pose vector:

$$\mathbf{p} = (t_x, t_y, t_z, r_x, r_y, r_z), \tag{9}$$

where the vector $\mathbf{t} = (t_x, t_y, t_z)$ represents the spatial location of the camera, corresponding to the user moving three-dimensionally in the space and the rotation vector $\mathbf{r} = (r_x, r_y, r_z)$ describes the orientation of the camera coordinate system with respect to world coordinates. A more accurate definition of the pose will follow.

The head pose measured by the tracking system is used by the graphics engine that renders the content displayed on the HMD. The engine derives two different camera poses from the estimated head pose, corresponding to the left and right eyes. These virtual cameras are used for rendering a stereo view of the original content. The stereo view is finally displayed on the HMD, one half of the view for each eye.

It is possible to use one of the halves to estimate the pose of the corresponding virtual camera and recover the pose that is estimated by the tracking system. Each frame of the display content is captured by the smart camera. There is a target pattern displayed in the 3D content, where there are feature points which can be detected from the two-dimensional projected image. Given these points, one can recover the pose of the camera that was used to render the projected image.

More generally, the task of finding the camera pose given a projection of a fixed target is called the Perspective-n-Point (PnP) problem. It is well known in the machine vision and robotics industry, and there are ready-made solutions such as

the `solvePnP` function in the OpenCV computer vision library [44] that implements a variety of methods for solving the problem. Most likely similar methods are also employed in the head tracking systems of VR HMDs.

In this section, a method is derived for estimating pose in the case of small displacements. OpenCV's iterative PnP solver is also examined as an alternative. The effects of the choice of target pattern are studied and characterization of the algorithms' stability is carried out.

In the first actual pose estimation measurements, a QR code was selected as the target pattern. QR codes have an easily recognizable shape, and their orientation is well defined via the asymmetry of the pattern. There are four recognizable corner points which should suffice for solving for the camera pose. Additionally, it is easy to embed information in the QR code if desired. For example, if there were multiple targets, each of these could have a unique identifier.

An algorithm for detecting QR codes was implemented on the smart camera, which can detect the QR code corner points with subpixel accuracy, which is essential for accurate measurements. The algorithm also corrects barrel distortion if any is present due to the optics. The implementation was not part of the thesis work and thus is not further discussed.

## 5.1 The pinhole camera model

Many of the existing pose estimation methods, including the iterative solver in OpenCV, take the pinhole camera model as a starting point. This camera model is essentially a perspective projection, describing how a point in three-dimensional world coordinates is transformed to two-dimensional image coordinates by the camera.

The camera defines its own coordinate frame, where the camera's optical axis is conventionally taken as the $z$ axis. The $x$ and $y$ axes are defined by the image's horizontal and vertical axes. In this coordinate system, the camera is fixed at the origin and the image plane, orthogonal to the $z$ axis, is situated at $z = f$ where $f$ is the focal length of the camera. The $z$ axis passes through the image center point $(c_x, c_y)$. This setting is illustrated in figure 17.

In figure 17, there is also an object that has coordinates $(x^c, y^c, z^c)$ in the camera frame (indicated by the superscript $c$). There is also a distinct world coordinate frame, where the object has coordinates $(x, y, z)$. These are transformed to the

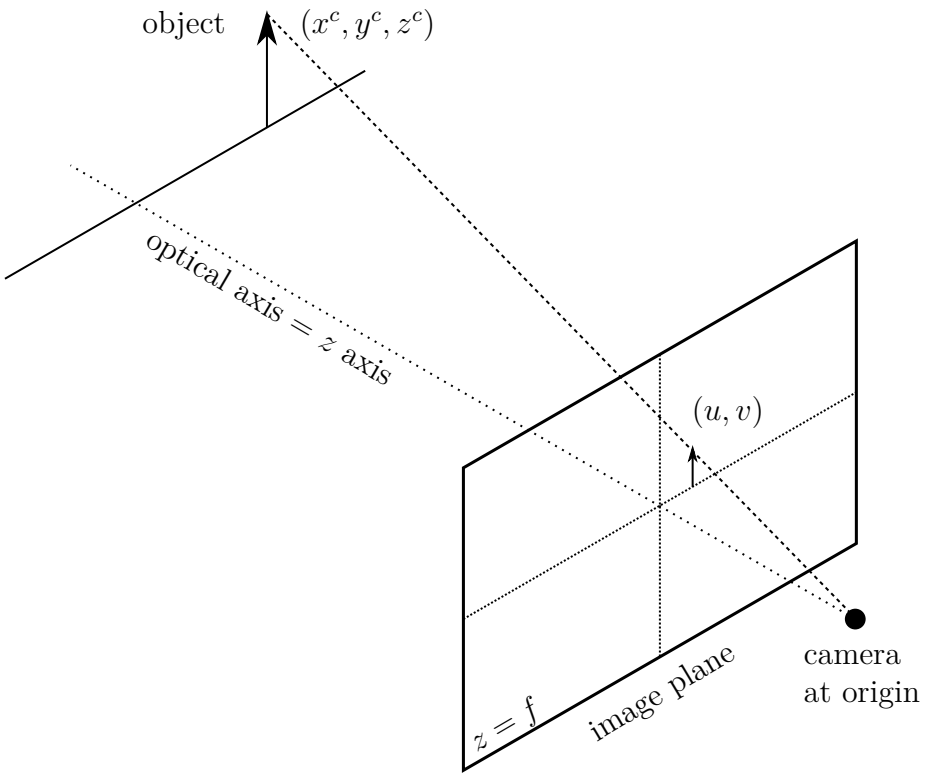



**Figure 17.** An illustration of the perspective projection that is described by the pinhole camera model. Object coordinates $(x^c, y^c, z^c)$ are in the camera coordinate system. $(u, v)$ are the horizontal and vertical image coordinates, respectively.

camera frame by

$$\begin{pmatrix} x^c \\ y^c \\ z^c \end{pmatrix} = \mathbf{R} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \mathbf{t}, \tag{10}$$

where superscript $c$ indicates the camera reference frame, $\mathbf{R}$ is an arbitrary three-dimensional rotation matrix and $\mathbf{t}$ is a translation vector. The inverse transformation

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \mathbf{R}^{-1} \left[ \begin{pmatrix} x^c \\ y^c \\ z^c \end{pmatrix} - \mathbf{t} \right] = \mathbf{R}^{-1} \begin{pmatrix} x^c \\ y^c \\ z^c \end{pmatrix} - \mathbf{R}^{-1}\mathbf{t} \tag{11}$$

takes points from camera coordinates to world coordinates. From here, one can also read off the camera pose in world coordinates: the camera origin is at the world

coordinate point $-\mathbf{R}^1\mathbf{t}$ and its axes are obtained by rotating the world coordinate axes by $\mathbf{R}^{-1}$.

The intrinsic parameters of the camera define a perspective projection which brings the object from camera coordinates $(x^c, y^c, z^c)$ to image coordinates $(u, v)$ (figure 17). Given the transformation (10), one may transform objects from world coordinates to camera coordinates by

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{=\mathbf{K}} \left[ \mathbf{R}\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \mathbf{t} \right], \tag{12}$$

where $s$ is a scale factor, $f$ is the camera focal length in pixels and $(c_x, c_y)$ the image center point in pixels. The matrix $\mathbf{K}$ is the matrix of intrinsic camera parameters and is acquired by calibrating the camera. [44] One can also equivalently express the equation with a $3 \times 4$ rotation and translation matrix and the model point in homogeneous coordinates $\mathbf{x}^h = \begin{pmatrix} x & y & z & 1 \end{pmatrix}^{\mathrm{T}}$, resulting in a more compact form:

$$\mathbf{q}^h = \mathbf{K}\begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix}\mathbf{x}^h, \tag{13}$$

where $\mathbf{q}^h = s\begin{pmatrix} u & v & 1 \end{pmatrix}^{\mathrm{T}}$ is the image point in homogeneous coordinates and $\begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix}$ is the combined rotation and translation matrix. The homogeneous image coordinates $\mathbf{q}^h$ must be transformed to get the final two-dimensional image coordinates. Since

$$\mathbf{q}^h = s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} su \\ sv \\ s \end{pmatrix} \equiv \begin{pmatrix} x^h \\ y^h \\ z^h \end{pmatrix},$$

one can calculate $u$ and $v$ as

$$\mathbf{q} = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{z^h}\begin{pmatrix} x^h \\ y^h \end{pmatrix}. \tag{14}$$

Despite being called the pinhole camera model, this model describes more conventional cameras such as machine vision cameras reasonably well. One must note, though, that this model does not account for optical distortions such as barrel distortion. However, the QR code detection algorithm that is used in the measurements

estimates the amount of barrel distortion, and thus the distortion is already taken in account.

## 5.2 OpenCV's iterative PnP solver

The OpenCV machine vision library has a versatile suite of algorithms for camera calibration, distortion correction and pose estimation [44]. One of the functions contained in the module is `solvePnP` which performs pose estimation and has a selection of algorithms for solving the PnP problem, including algebraic and iterative methods.

In the programming interface, the iterative method is chosen as the default. It starts with a coarse estimate of the pose, either obtained internally by the function or given by the user. It also takes the camera calibration matrix, the model points and the corresponding image points as arguments. From there, it tries to iteratively refine the initial pose estimate. [44]

The refinement is based on iteratively minimizing a cost function (also called the error norm). An intuitive choice of a cost function, the reprojection error, is used in the iterative solver [44]. The idea is to use the estimated camera pose to project the model points to image points using the camera model. The error function is then calculated as the sum of squared distances between the calculated and measured image points. This cost function seems intuitive since it also corresponds to the perceived distance between the projected and measured points and hence could potentially yield correct results.

Formally, the cost function is represented as the sum of squared Euclidean distances between the projected points and the actual image points:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} \left\| \mathbf{q}_i - P(\mathbf{R}, \mathbf{t}; \mathbf{x}_i) \right\|^2, \tag{15}$$

where $i$ runs over the model coordinates $\mathbf{x}_i$, $\mathbf{q}_i$ are the corresponding measured image points $(u_i, v_i)$ and $P$ is the projective mapping that transforms the model points from world coordinates to two-dimensional image coordinates. In OpenCV, $P$ is an extended version of the camera model (13) which also takes lens distortion in account [44].

With the cost function defined, the pose estimation problem is stated as a

nonlinear minimization problem:

$$\arg\min_{\mathbf{R},\mathbf{t}} E(\mathbf{R},\mathbf{t}), \tag{16}$$

i.e. find such $\mathbf{R}$ and $\mathbf{t}$ that minimize $E(\mathbf{R},\mathbf{t})$. OpenCV performs the minimization using the Levenberg-Marquardt algorithm which is a standard, iterative method for solving non-linear least squares problems [44]. It is particularly convenient because the cost function (15) is already in a sum-of-squares form and thus a least squares method is directly applicable.

## 5.3 Pose estimation by linear least squares

OpenCV's pose estimation algorithms are designed to operate in generic cases where the displacements might have large values. The measurement at hand in this research is quite specialized — the aim is to accurately detect small displacements with respect to an initial pose, which likely are in the regime where small angle approximations are justified. Therefore, the question arises whether it would be sufficient to approximate the camera model to first order with respect to the spatial and angular displacements and use this linearization to minimize the cost function (15). A linear, non-iterative method would at least have the benefits of lower computational complexity and potentially stabler operation. An obvious disadvantage is error due to nonlinearity when the approximation is no longer valid.

Let us take the camera model (13) as a starting point, not taking optical distortions in account. To estimate small displacements with respect to an initial pose, it is convenient to define the world coordinate frame such that it coincides with the initial camera coordinate frame. That is, the camera rests at the origin of the world coordinate frame and its axes coincide with the world coordinate axes. This choice simplifies the mathematics, since in the initial pose $\mathbf{R} = \mathbf{I}$ and $\mathbf{t} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}}$.

Equation (13) already exposes the three spatial degrees of freedom in the translation vector $\mathbf{t}$. The rotation matrix $\mathbf{R}$, however, has been so far left undefined other than as an arbitrary three-dimensional rotation matrix. A convenient way to express

an arbitrary three-dimensional rotation is the rotation vector:

$$\mathbf{r} \equiv \theta \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \theta \hat{\mathbf{r}}, \tag{17}$$

where $\theta = \|\mathbf{r}\|$ defines the rotation angle and the unit vector $\hat{\mathbf{r}} = \mathbf{r}/\|\mathbf{r}\|$ defines the rotation axis. Any rotation in $\mathbb{R}^3$ can be defined as a rotation about a single axis, and this leads to the axis-angle representation $(\theta, \hat{\mathbf{r}})$. Exploiting the fact that $\hat{\mathbf{r}}$ is a unit vector, it is often multiplied by $\theta$, encoding all the information about the rotation into a single vector of $\mathbb{R}^3$ as presented in (17).

The rotation vector may be translated into a rotation matrix using the matrix exponential arising from the theory of the rotation group SO(3). The rotation matrix is calculated as

$$\mathbf{R}(\theta, \hat{\mathbf{r}}) = e^{\theta \mathbf{K}} \equiv \sum_{k=0}^{\infty} \frac{\theta^k}{k!} \mathbf{K}^k = \mathbf{I} + \theta \mathbf{K} + \frac{\theta^2}{2} \mathbf{K}^2 + \ldots, \tag{18}$$

where $\mathbf{K}$ is the cross product matrix of $\hat{\mathbf{r}}$:

$$\mathbf{K} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix},$$

i.e. a matrix such that for any $\mathbf{v} \in \mathbb{R}^3$, $\hat{\mathbf{r}} \times \mathbf{v} = \mathbf{K}\mathbf{v}$. A detailed discussion of the matrix exponential representation can be found in [45]. Equation (18) also has a closed form, representing an arbitrary rotation [45]:

$$\mathbf{R}(\theta, \hat{\mathbf{r}}) = \mathbf{I} + \mathbf{K} \sin\theta + \mathbf{K}^2 (1 - \cos\theta). \tag{19}$$

Assuming that the angle $\theta$ is small, one can arrive at a linear approximation by truncating the power series (18) at the first order term:

$$\mathbf{R}(\theta, \hat{\mathbf{r}}) \approx \mathbf{I} + \theta \mathbf{K}, \tag{20}$$

which is linear with respect to the rotation vector components, in contrast to (19) which is nonlinear due to the trigonometric functions. Defining the rotation vector

components as $\mathbf{r} = \begin{pmatrix} r_x & r_y & r_z \end{pmatrix}^{\mathrm{T}}$, (20) can be written as

$$\mathbf{R}(\mathbf{r}) \approx \begin{pmatrix} 1 & -r_z & r_y \\ r_z & 1 & -r_x \\ -r_y & r_x & 1 \end{pmatrix}. \tag{21}$$

The pose is thus now parametrized by the rotation vector $\mathbf{r} = \begin{pmatrix} r_x & r_y & r_z \end{pmatrix}^{\mathrm{T}}$ and the translation vector $\mathbf{t} = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^{\mathrm{T}}$, totaling six degrees of freedom.

Substituting (21) into (13) yields a linear group of equations:

$$\begin{aligned} \mathbf{q}^h &= \begin{pmatrix} x^h \\ y^h \\ z^h \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -r_z & r_y & t_x \\ r_z & 1 & -r_x & t_y \\ -r_y & r_x & 1 & t_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} x(f - c_x r_y) + y(c_x r_x - f r_z) + z(f r_y + c_x) + c_x t_z + f t_x \\ x(f r_z - c_y r_y) + y(c_y r_x + f) + z(c_y - f r_x) + c_y t_z + f t_y \\ -x r_y + y r_x + z + t_z \end{pmatrix}. \end{aligned} \tag{22}$$

In pose estimation, the model point $\begin{pmatrix} x & y & z \end{pmatrix}^{\mathrm{T}}$ remains fixed and the pose parameters are in the role of unknowns that we want to estimate about $\mathbf{t} = \mathbf{r} = 0$. The equation is easily reparametrized as

$$\mathbf{q}^h = \mathbf{q}_0^h + \underbrace{\begin{pmatrix} f & 0 & c_x & c_x y & f z - c_x x & -f y \\ 0 & f & c_y & c_y y - f z & -c_y x & f x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix}}_{\equiv \mathbf{J_x}} \begin{pmatrix} \mathbf{t} \\ \mathbf{r} \end{pmatrix}, \tag{23}$$

where $\mathbf{J_x}$ is the Jacobian matrix of $\mathbf{q}^h$ with respect to the pose parameters and

$$\mathbf{q}_0^h = \begin{pmatrix} f x + c_x z \\ f y + c_y z \\ z \end{pmatrix}. \tag{24}$$

Despite linearizing the rotation matrix, there is still nonlinearity in the camera model due to the conversion from homogeneous coordinates to image coordinates

in equation (14). This may, however, also be approximated to first order in the neighbourhood of the point $\mathbf{q}_0^h$:

$$\mathbf{q}(\mathbf{q}^h) = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{z^h} \begin{pmatrix} x^h \\ y^h \end{pmatrix} \approx \mathbf{q}(\mathbf{q}_0^h) + \mathbf{J}_\mathbf{q}(\mathbf{q}_0^h)(\mathbf{q}^h - \mathbf{q}_0^h), \tag{25}$$

where $\mathbf{J}_\mathbf{q}(\mathbf{q}_0^h)$ is the Jacobian matrix of $\mathbf{q}(\mathbf{q}^h)$:

$$\begin{aligned} \mathbf{J}_\mathbf{q}(\mathbf{q}_0^h) &= \begin{pmatrix} \frac{\partial u}{\partial x^h} & \frac{\partial u}{\partial y^h} & \frac{\partial u}{\partial z^h} \\ \frac{\partial v}{\partial x^h} & \frac{\partial v}{\partial y^h} & \frac{\partial v}{\partial z^h} \end{pmatrix}\Bigg|_{\mathbf{q}^h=\mathbf{q}_0^h} = \begin{pmatrix} \frac{1}{z^h} & 0 & -\frac{x^h}{(z^h)^2} \\ 0 & \frac{1}{z^h} & -\frac{y^h}{(z^h)^2} \end{pmatrix}\Bigg|_{\mathbf{q}^h=\mathbf{q}_0^h} \\ &= \begin{pmatrix} \frac{1}{z} & 0 & -\frac{fx+c_x z}{z^2} \\ 0 & \frac{1}{z} & -\frac{fy+c_y z}{z^2} \end{pmatrix}, \end{aligned} \tag{26}$$

where $\mathbf{q}_0^h$ was substituted from (24).

Now, to estimate the change in pose, one can calculate the difference between the observed image points $\mathbf{q}_i = \mathbf{q}(\mathbf{q}_i^h)$ and the corresponding observed image points $\mathbf{q}_{i,0} = \mathbf{q}(\mathbf{q}_{i,0}^h)$ in the initial pose where $i$ is the index of the model point. Substituting $\mathbf{q}_i^h - \mathbf{q}_{i,0}^h$ from (23), one obtains

$$\mathbf{q}_i - \mathbf{q}_{i,0} = \mathbf{J}_\mathbf{q}(\mathbf{q}_{i,0}^h)(\mathbf{q}_i^h - \mathbf{q}_{i,0}^h) \overset{(23)}{=} \mathbf{J}_\mathbf{q}(\mathbf{q}_{i,0}^h)\mathbf{J}_\mathbf{x}(\mathbf{x}_i) \begin{pmatrix} \mathbf{t} \\ \mathbf{r} \end{pmatrix}.$$

Carrying out the multiplication of the Jacobian matrices, the final form reads

$$\Delta\mathbf{q}_i = \begin{pmatrix} \Delta u_i \\ \Delta v_i \end{pmatrix} = \underbrace{\frac{f}{z_i^2} \begin{pmatrix} z_i & 0 & -x_i & -x_i y_i & z_i^2 + x_i^2 & -y_i z_i \\ 0 & z_i & -y_i & -(z_i^2 + y_i^2) & x_i y_i & x_i z_i \end{pmatrix}}_{\equiv \mathbf{J}_i(\mathbf{x}_i)} \underbrace{\begin{pmatrix} \mathbf{t} \\ \mathbf{r} \end{pmatrix}}_{\equiv \mathbf{p}} \equiv \mathbf{J}_i(\mathbf{x}_i)\mathbf{p}, \tag{27}$$

where the pose vector $\mathbf{p}$ has been defined as

$$\mathbf{p} = \begin{pmatrix} \mathbf{t} \\ \mathbf{r} \end{pmatrix} = \begin{pmatrix} t_x & t_y & t_z & r_x & r_y & r_z \end{pmatrix}^\mathrm{T}. \tag{28}$$

Given $N$ model points $\mathbf{x}_i$, one can write equation (27) for each of them, resulting

in system group of $N \times 2$ equations and six unknowns:

$$\begin{pmatrix} \Delta u_1 \\ \Delta v_1 \\ \vdots \\ \Delta u_N \\ \Delta v_N \end{pmatrix} = f \underbrace{\begin{pmatrix} \frac{1}{z_1} & 0 & -\frac{x_1}{z_1^2} & -\frac{x_1 y_1}{z_1^2} & 1+\frac{x_1^2}{z_1^2} & -\frac{y_1}{z_1} \\ 0 & \frac{1}{z_1} & -\frac{y_1}{z_1^2} & -\left(1+\frac{y_1^2}{z_1^2}\right) & \frac{x_1 y_1}{z_1^2} & \frac{x_1}{z_1} \\ & & & \vdots & & \\ \frac{1}{z_N} & 0 & -\frac{x_N}{z_N^2} & -\frac{x_N y_N}{z_N^2} & 1+\frac{x_N^2}{z_N^2} & -\frac{y_N}{z_N} \\ 0 & \frac{1}{z_N} & -\frac{y_N}{z_N^2} & -\left(1+\frac{y_N^2}{z_N^2}\right) & \frac{x_N y_N}{z_N^2} & \frac{x_N}{z_N} \end{pmatrix}}_{\equiv \mathbf{J}} \begin{pmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{pmatrix}. \tag{29}$$

It is immediately noted that at least three points are required to render the system well-determined. In the case of the QR code target and four detectable corner points there are eight equations, resulting in overdetermination of (29). An overdetermined system does not necessarily have an exact solution — rather, this kind of a system is usually solved in a least squares fashion. That is, a numerical method is used to obtain a solution (pose) that minimizes the sum of squared norms of the of the $\Delta \mathbf{q}_i$ vectors. Standard methods such as the **QR** decomposition or the Moore-Penrose pseudoinverse may be used for acquiring the solution.

It is immediately noticed that the error norm corresponds to the reprojection error cost function (15) that is also used by OpenCV's solver. Therefore, the newly developed method is based on foundations similar to the OpenCV solver, with the difference that the new methods approximates the camera model to first order and does not iteratively improve the solution.

## 5.4 Characterizing the pose estimation methods

Because the pose estimation method is to be used as a quantitative measurement method, it is important to be aware of its limitations, error margins and other issues potentially affecting the measurement results. Ideally, an estimate of the results' uncertainties should be produced.

There are several ways to analyze this kind of an algorithm. First, the linear least squares method is mathematically quite simple and therefore some of its properties are easily explored. Second, both methods can be characterized by generating simulated measurement data and adding measurement noise, similarly to what was done with the sharpness estimation algorithms. Finally, ground truth measurements could be performed by assembling a static target, e.g. a QR code printed on paper,

and moving the camera in front of it using a robot that has position feedback.

The linear least squares algorithm was characterized using Monte Carlo simulations, with OpenCV's non-linear iterative method as a baseline. Measurements were also made using an application that runs on PC and displays a QR code with a given pose on HTC Vive.

### 5.4.1 Simulating measurement data

Using the camera model (13), points of the target model can be mapped to image coordinates with different poses. Gaussian noise is added to the $(u, v)$ coordinates to perturb the algorithm and get an idea of its behaviour when noise is present in real measurement data. This way the estimated pose can be compared to the original pose used to generate the measurement data.

The error between the estimated and measured pose has to be calculated using some metric. It is not trivial since the pose is comprised of six components where the first three ones correspond to translation and the last three ones to rotation. Therefore, simply calculating the Euclidean distance between the two pose vectors does not yield meaningful results. The error has to be calculated separately for the translational and rotational parts.

For the translational part, the Euclidean distance is well defined and can be used to calculate the distance between the estimated and actual camera locations:

$$\Delta t = \left\| \mathbf{t}_{\text{estimated}} - \mathbf{t}_{\text{actual}} \right\|. \tag{30}$$

For the rotational part, one may note that the estimated rotation can be written as the composition of the actual rotation and another rotation. In terms of rotation matrices,

$$\mathbf{R}_{\text{estimated}} = \mathbf{R}_{\Delta}\mathbf{R}_{\text{actual}}.$$

Multiplying on the right by the inverse of $\mathbf{R}_{\text{actual}}$, $\mathbf{R}_{\Delta}$ is written as

$$\mathbf{R}_{\Delta} = \mathbf{R}_{\text{estimated}}\mathbf{R}_{\text{actual}}^{-1}.$$

The rotations matrices can be readily computed from the rotation part of the pose vector using equation (19). $\mathbf{R}_{\Delta}$ represents the rotation between the estimated and

actual coordinate system. The angle of this rotation would be a descriptive error metric. This angle can be recovered from $\mathbf{R}_\Delta$ using the equation:

$$\Delta\theta = \cos^{-1}\left(\frac{\mathrm{tr}(\mathbf{R}_\Delta) - 1}{2}\right), \tag{31}$$

where $\mathrm{tr}(\mathbf{R}_\Delta)$ is the matrix trace, i.e. the sum of the diagonal elements [45]. The pair $(\Delta t, \Delta\theta)$ thus serves as an error estimator and preserves the ability to distinguish errors in the rotational and translational parts.

To study the effect of a variable such as noise magnitude or the measurement geometry, a Monte Carlo method was utilized. A large number of random poses were generated and the model projected to image coordinates for each pose using the camera model. The pose was estimated from the image coordinates, and the error between the original and the estimated pose was calculated. With a sufficient number of pose samples, a reasonably good coverage of the pose space was achieved. In practice, for each different measurement configuration, 50000 pose samples were drawn. The standard deviations of the translational and rotational parts were calculated from the Monte Carlo samples as

$$\sigma_t = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta t)_i^2}, \quad \sigma_r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta r)_i^2}, \tag{32}$$

where $N$ is the number of Monte Carlo samples, $\sigma_t$ is the standard deviation of the translational part and $\sigma_r$ the standard deviation of the rotational part.

In each simulation, a fixed magnitude of displacement from the origin pose was specified for the translational and rotational parts. The displacement was interpreted to correspond to the radius of a spherical shell in the corresponding three-dimensional (translation or rotation) space. Then samples were uniformly drawn from these spherical shells, generating pose samples which all deviate from the origin by the same amount but in a different direction. Due to the axis-angle rotation representation, this also leads to a fixed rotation angle with respect to the origin pose, with a random rotation axis. Sphere point picking is realized by drawing two uniform random numbers $\theta \in [0, 2\pi[$ and $u \in [-1, 1]$ and calculating the point

coordinates according to

$$x = \sqrt{1-u^2}\cos\theta, \tag{33}$$

$$y = \sqrt{1-u^2}\sin\theta, \tag{34}$$

$$z = u, \tag{35}$$

as described in [46].

For example, with the translational displacement set at 5 mm and rotational at 2°, two sphere points would be picked — the first from a spherical shell with a radius of 5, the second from one with a radius of 2. The first sphere point corresponds to the pose components $t_x, t_y$ and $t_z$ and the second sphere point corresponds to the $r_x, r_y$ and $r_z$ components.

In these simulations, it was assumed for simplicity that neither the camera calibration $\mathbf{K}$ nor the origin points $\mathbf{q}_{0,i}$ have any measurement uncertainty. While this is not the case in real-life measurements, it is reasonable to assume that if calibration is done with care, the uncertainty of calibration or the origin image points is negligible compared to the uncertainty of individual samples of the image points.

A fixed camera calibration matrix was used in the simulations. The calibration parameters $f = 450\,\text{px}$, $c_x = 94\,\text{px}$ and $c_y = 60\,\text{px}$ were chosen in correspondence to a real life measurement scenario where the image resolution is $188 \times 120$ pixels.

### 5.4.2 Planar target

In the first simulations, a planar, square target pattern with four corners was used to produce a scenario that is principally equivalent to the QR code target described before. The set of model points is defined as

$$\{\mathbf{p}_i\} = \{(-l/2, -l/2, d), (l/2, -l/2, d), (-l/2, l/2, d), (l/2, l/2, d)\}, \tag{36}$$

where $l$ is the side length of the square and $d$ is the $z$ position of the target, i.e. its distance to the camera in the initial pose. In these simulations, these geometry parameters were set to $l = 50\,\text{mm}$ and $d = 300\,\text{mm}$.

First, the sensitivity to noise was characterized by performing simulations with different amounts of noise added to the projected image points. The pose was estimated from the image points using both the newly derived linear least squares

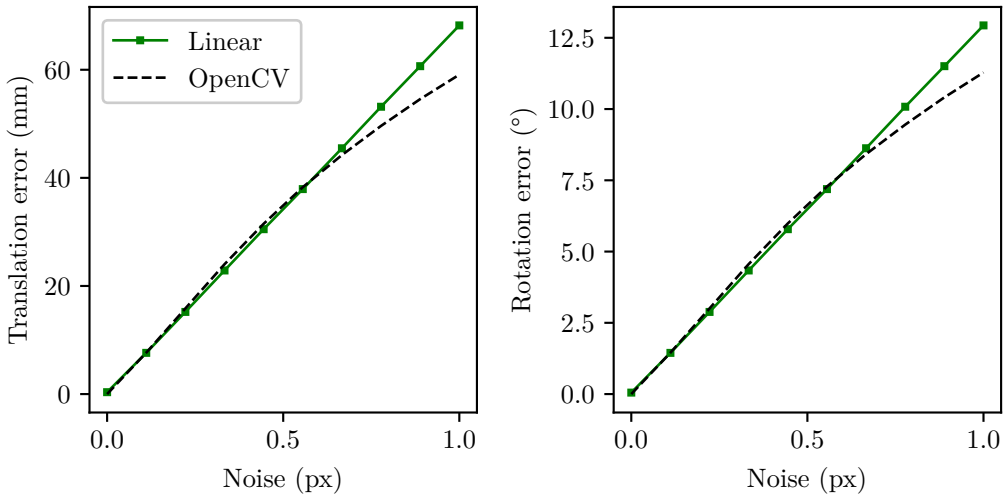



**Figure 18.** Rotation and translation errors of the linear least squares and OpenCV iterative methods as a function of the amount of coordinate noise. Planar target was used. The test poses had both translational and rotational displacements.

method and the OpenCV iterative PnP solver. The translational and rotational displacements of the samples from the origin were set to small values of 5 mm and 2°. Values calculated from equation (32) for rotational and translational parts with different noise standard deviations are plotted in figure 18.

From the figure, it is immediately seen that both translational and rotational error increase rapidly with the noise standard deviation, quickly reaching unacceptable values of several tens of millimeters and over ten degrees. As another property, both error measures are linear with respect to coordinate noise for the linear least squares method, which is an expected results. On the other hand, OpenCV's solver exhibits non-linear response to noise, probably due to the non-linear operation of the algorithm.

By inspecting individual samples, it was noticed that the main source of pose estimation error was mixing of rotations into translations and vice versa — in the image coordinate data, a pose with rotation about the $x$ axis would seem considerably similar to a pose with $y$ translation. A similar ambiguity is seen between $y$ translation and $x$ rotation. The more noise was added to the coordinates, the harder the ambiguity was to resolve for both pose estimation algorithms.

To confirm the finding by simulations, two additional experiments were made.

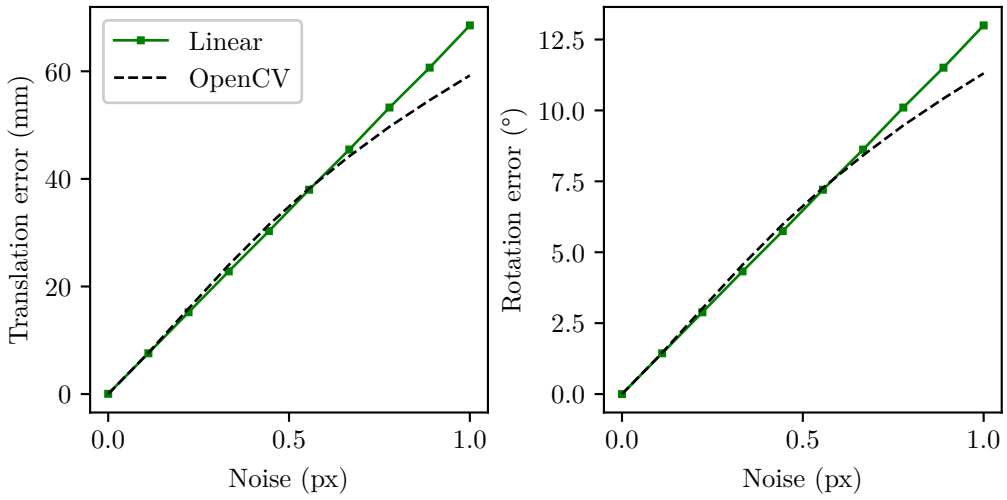



**Figure 19.** Rotation and translation errors of the linear least squares and OpenCV iterative methods as a function of the amount of coordinate noise. Planar target was used. The test poses had only translational displacements.

In both experiments, the sampled poses had only translational displacements of 5 mm but no rotational displacement. The first experiment was conducted with both algorithms unmodified. The result, presented in figure 19, was very similar to the earlier result in figure 18, having high error values.

In the second experiment, the linear least squares method was modified such that the rotational $x$ and $y$ degrees of freedom were entirely removed from the equation group, in order to remove the ambiguity. The OpenCV algorithm was unmodified. The results is shown in figure 20.

The removal of the ambiguity resulted in a dramatic decrease of error values and increased noise resistance. It must be noted that even though the error values seem low in this test, the modified algorithm is not immediately useful: if the rotational displacement was nonzero, the rotation would then be interpreted as translation, rendering the results erroneous again.

In further investigation of the problem, it became clear that similar ambiguities with planar targets have been found in earlier research of pose estimation. Particularly, Schweighofer and Pinz present a robust pose estimation method that overcomes a specific ambiguity with planar targets by algebraically finding two local minima of the error function and choosing the one that produces the lowest error value [47]. The benefits of using this algorithm for tracking error estimation are not obvious,

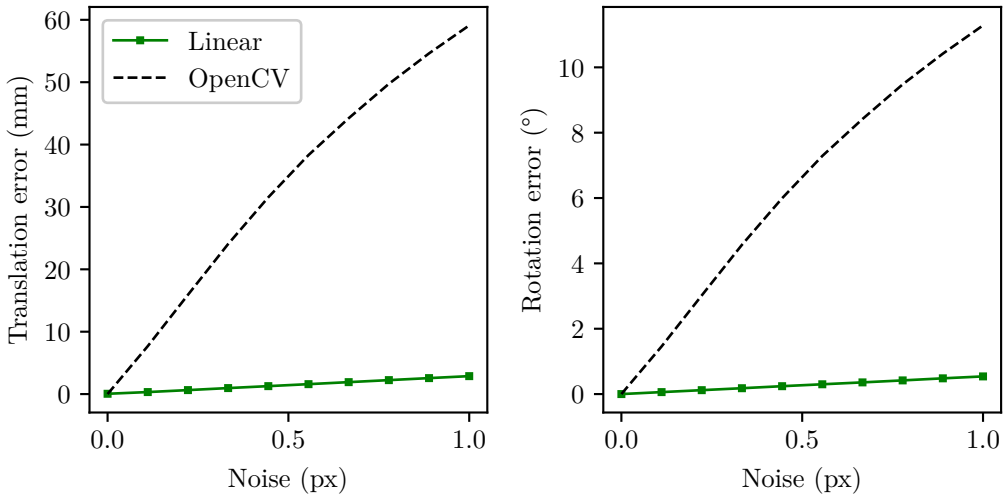



**Figure 20.** Rotation and translation errors of the modified linear least squares and OpenCV iterative methods as a function of the amount of coordinate noise. Planar target was used. The test poses had only translational displacements.

however, due to the inherently unstable geometric configuration and requirement to get accurate angle estimates despite high levels of noise. Even after resolving between the two ambiguous minima, the algorithm still showed several degrees of standard deviation of the angle [47].

### 5.4.3 Planar target with additional off-plane point

Since the rotation-translation ambiguity was attributed to the planar target, the next logical step was to simulate a non-planar target. As a minimal modification to the original target pattern, one off-planar point was added to the model. The points of the new model are then

$$\{\mathbf{p}_i\} = \{(-l/2, -l/2, d), (l/2, -l/2, d), (-l/2, l/2, d), \tag{37}$$

$$(l/2, l/2, d), (0, 0, d')\}, \tag{38}$$

where the additional point was added on the optical axis at $z = d'$. In these simulations, the parameters were set to the values $l = 50\,\text{mm}$, $d = 300\,\text{mm}$ and $d' = 200\,\text{mm}$.

In the first simulation, randomly generated poses were used and the amount of noise added to the image points was varied. Similar to the corresponding simulation

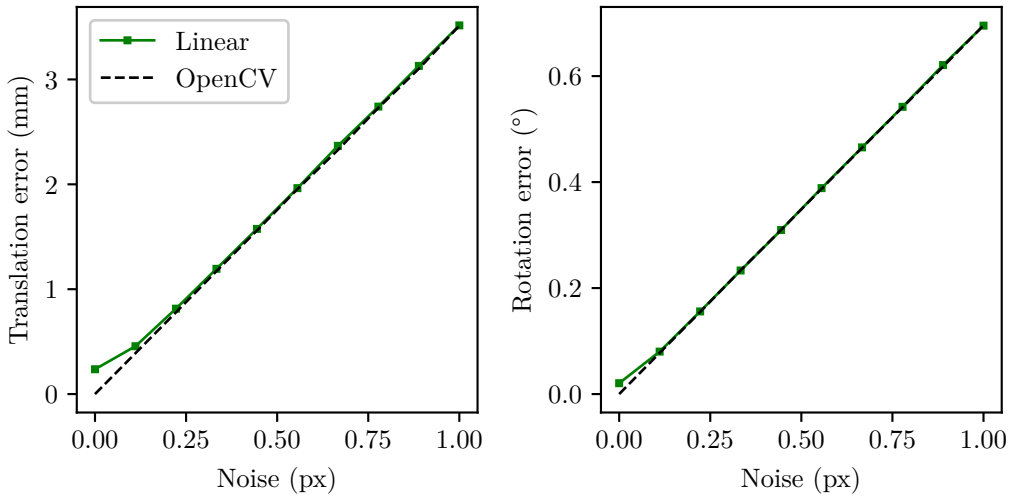



**Figure 21.** Mean error of the poses estimated by the linear least squares method and OpenCV's iterative solver as a function of coordinate noise. Planar target with an additional off-plane point was used. The test poses had both translational and rotational displacements.

with the planar target, all the generated poses had a deviations of 5 mm and 2° from the origin pose. The results are shown in figure 21.

Compared to the corresponding simulation with the planar target (figure 18), here the sensitivity to noise is considerably lower. The behaviour of the error is practically linear with respect to the noise amount at least at the small displacements that were used here. It appears that the additional point has a stabilizing effect on the equation group.

While the behaviour of the linear method at large displacements could not be practically studied with the planar target due to instabilities, the new target gives an opportunity to do so. Monte Carlo simulations were performed again, plotting pose error as a function of rotation angle, ranging from 0 to 15 degrees. Coordinate noise was kept at the constant value of 0.2. The results are shown in figure 22.

Both rotation and translation errors show a rapid rise after about five degrees when the linear method is used. This is where the small angle approximation done in (20) begins to cause large errors especially in the translational part. OpenCV, on the other hand, shows small constant error for both parts. As a non-linear method it obviously has better behaviour here because it is able to take larger displacements in account when minimizing the error function.

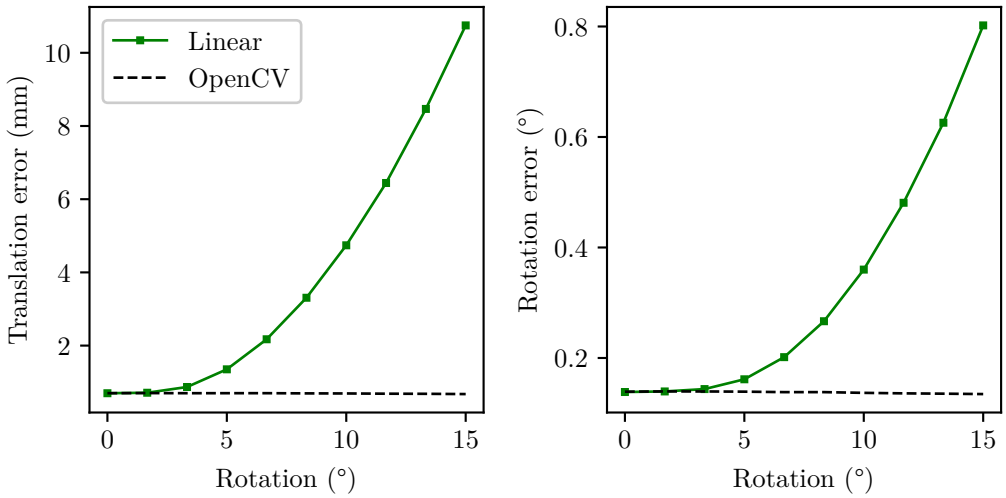



**Figure 22.** Mean error of the poses estimated by the linear least squares method and OpenCV's iterative solver as a function of rotation angle. Planar target with an additional off-plane point was used.

Despite its behaviour at large rotational displacements, the linear method can serve certain purposes well, for example detecting small drifting of the head tracking algorithm. Larger, temporary rotations due to jitter, for example, are also shown in the estimated pose although their values are not accurate. Thus the linear method can still serve as estimator where objective estimates of tracking quality are desired. On the other hand, it is relatively easy to extend the linear method to a non-linear one by approximating the rotation matrix $\mathbf{R}$ to a higher order by using the matrix exponential (18) and using a non-linear least squares approximation method to acquire the solution. The change would bring the algorithm closer to the one in OpenCV. This development direction is left for future research.

As another experiment, the effect of the $z$ coordinate $d'$ of the off-planar point on the pose estimation error was studied by simulation. Errors are shown in figure 23. Test poses were chosen such that their displacements were 5 mm and 2°.

With the plane placed at $z = 300\,\text{mm}$, the errors increase with $d'$. This is to be expected since the closer $d'$ is to the plane $z = 300\,\text{mm}$, the more planar the target is. At $d' = 200\,\text{mm}$, further decreasing $d'$ does not show much benefit. Both linear and OpenCV pose estimation methods show similar characteristics in this simulation.

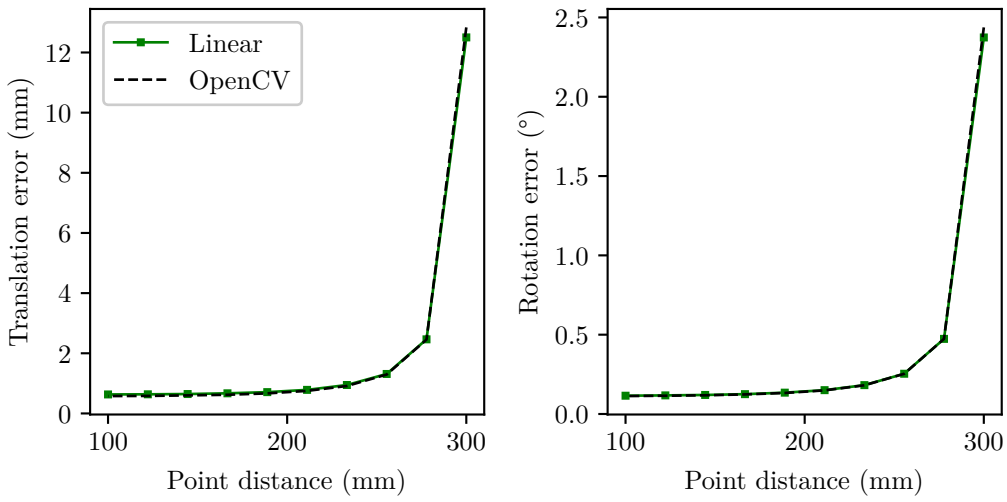



**Figure 23.** Pose estimation errors as a function of the $z$ coordinate $d'$ of the off-planar model point.

## 5.5 Error estimation and statistical analysis

As a final confirmation and characterization of the ambiguity problem, statistical analysis of the linear least square pose estimator was carried out. The pose estimate is calculated by finding the least squares solution of equation (29). The measurement uncertainties of the detected feature points propagate to the pose through the solving process. The uncertainty of the pose estimate can be deduced using the laws of uncertainty propagation.

Equation (29) has the form

$$\Delta\mathbf{q} = \mathbf{J}\mathbf{p}.$$

Finding the least squares solution to this linear system is equivalent to multiplying $\mathbf{q}$ by the Moore-Penrose pseudoinverse of $\mathbf{J}$:

$$\underbrace{\mathbf{J}^{+}\mathbf{J}}_{=\mathbf{I}}\,\mathbf{p} = \mathbf{p} = \mathbf{J}^{+}\Delta\mathbf{q}, \tag{39}$$

where $\mathbf{J}^{+}$ is the pseudoinverse of $\mathbf{J}$. Thus, if the pseudoinverse can be computed, the solution is linear with respect to the input vector.

Assuming that the variances and covariances of the input variables $\Delta\mathbf{q}$ are known,

the (co)variances of the output variables $\mathbf{p}$ can be determined. For a matrix equation of the form $\mathbf{y} = \mathbf{A}\mathbf{x}$, the error propagation law reads [48]:

$$\mathbf{\Sigma_y} = \mathbf{A}\mathbf{\Sigma_x}\mathbf{A}^T, \tag{40}$$

where $\mathbf{\Sigma_x}$ and $\mathbf{\Sigma_y}$ are the covariance matrices of the input and output variables, respectively. The covariance matrix is defined as

$$\mathbf{\Sigma_x} = \begin{pmatrix} \mathrm{var}(x_1) & \mathrm{cov}(x_1, x_2) & \cdots & \mathrm{cov}(x_1, x_n) \\ \mathrm{cov}(x_2, x_1) & \mathrm{var}(x_2) & \cdots & \mathrm{cov}(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(x_n, x_1) & \mathrm{cov}(x_n, x_2) & \cdots & \mathrm{var}(x_N) \end{pmatrix}, \tag{41}$$

where $\mathrm{var}(x_i) = E\left[(x_i - \overline{x}_i)^2\right]$ is the variance of the variable $x_i$, $\mathrm{cov}(x_i, x_j) = E\left[(x_i - \overline{x}_i)(x_j - \overline{x}_j)\right]$ is the covariance of the variables $x_i$ and $x_j$, $E[\dots]$ denotes the expectation value and $\overline{x}_i$ the mean of the observed values of $x_i$. The matrix is symmetric and can be calculated from measurement data.

Equation (39) has a form such that equation (40) can be applied to it. If we also assume no uncertainty in the camera calibration and the origin image points, the covariances of the output variables can be calculated as

$$\mathbf{\Sigma_p} = \mathbf{J}^+\mathbf{\Sigma_q}(\mathbf{J}^+)^T. \tag{42}$$

Thus, $\mathbf{\Sigma_p}$ is dependent on the covariances of the input variables and the Jacobian pseudoinverse which encodes the geometrical configuration of the measurement setup.

Let us assume further for simplicity that all measured coordinates (input variables) have a constant variance of $\sigma_q$ and their covariances are zero. The covariance matrix for $\Delta\mathbf{q}$ then becomes

$$\Sigma_\mathbf{q} = \mathrm{diag}(\sigma^2_{u_1}, \sigma^2_{v_1}, \dots \sigma^2_{u_n}, \sigma^2_{v_n}) = \sigma_q^2\mathbf{I}.$$

With this simplification, equation (42) reduces to

$$\mathbf{\Sigma_p} = \sigma_q^2\mathbf{J}^+(\mathbf{J}^+)^T, \tag{43}$$

where $\sigma_q$ only has the role of scaling. Therefore, the product $\mathbf{J}^+(\mathbf{J}^+)^T$ itself describes

the statistical behaviour of the least squares solution well enough.

For the planar model (36) with $l = 50\,\text{mm}$ and $d = 300\,\text{mm}$, the output covariance matrix evaluates to

$$\boldsymbol{\Sigma}_\mathbf{p} = \begin{pmatrix} \operatorname{var}(t_x) & \operatorname{cov}(t_x,t_y) & \operatorname{cov}(t_x,t_z) & \operatorname{cov}(t_x,r_x) & \operatorname{cov}(t_x,r_y) & \operatorname{cov}(t_x,r_z) \\ \operatorname{cov}(t_y,t_x) & \operatorname{var}(t_y) & \operatorname{cov}(t_y,t_z) & \operatorname{cov}(t_y,r_x) & \operatorname{cov}(t_y,r_y) & \operatorname{cov}(t_y,r_z) \\ \operatorname{cov}(t_z,t_x) & \operatorname{cov}(t_z,t_y) & \operatorname{var}(t_z) & \operatorname{cov}(t_z,r_x) & \operatorname{cov}(t_z,r_y) & \operatorname{cov}(t_z,r_z) \\ \operatorname{cov}(r_x,t_x) & \operatorname{cov}(r_x,t_y) & \operatorname{cov}(r_x,t_z) & \operatorname{var}(r_x) & \operatorname{cov}(r_x,r_y) & \operatorname{cov}(r_x,r_z) \\ \operatorname{cov}(r_y,t_x) & \operatorname{cov}(r_y,t_y) & \operatorname{cov}(r_y,t_z) & \operatorname{cov}(r_y,r_x) & \operatorname{var}(r_y) & \operatorname{cov}(r_y,r_z) \\ \operatorname{cov}(r_z,t_x) & \operatorname{cov}(r_z,t_y) & \operatorname{cov}(r_z,t_z) & \operatorname{cov}(r_z,r_x) & \operatorname{cov}(r_z,r_y) & \operatorname{var}(r_z) \end{pmatrix}$$

$$= \sigma_q^2 \mathbf{J}^+ (\mathbf{J}^+)^T = \sigma_q^2 \begin{pmatrix} 2336.22 & 0 & 0 & 0 & -443.09 & 0 \\ 0 & 2336.22 & 0 & 443.09 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 443.09 & 0 & 84.04 & 0 & 0 \\ -443.09 & 0 & 0 & 0 & 84.04 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.29 \end{pmatrix},$$

where the results have been truncated to two decimal places and the units of length and angle are millimeters and degrees, respectively.

The large values on the diagonal are immediately noticed, leading to large uncertainties of the output variables. These uncertainties can be calculated as

$$\sigma_{p_i} = \sigma_q \sqrt{(\boldsymbol{\Sigma}_\mathbf{p})_{ii}}, \tag{44}$$

i.e. the square root of the corresponding diagonal element of $\boldsymbol{\Sigma}_\mathbf{p}$. For the planar target, assuming $\sigma_q = 0.2\,\text{px}$ (a plausible value based on tests of the QR code locating algorithm), the uncertainties are

$$\begin{aligned} \sigma_\mathbf{p} &= \sigma_q \begin{pmatrix} 48.3 & 48.3 & 2.8 & 9.2 & 9.2 & 0.5 \end{pmatrix}^\mathrm{T} \\ &\overset{\sigma_q = 0.2}{\approx} \begin{pmatrix} 9.7 & 9.7 & 0.6 & 1.8 & 1.8 & 0.1 \end{pmatrix}^\mathrm{T}. \end{aligned}$$

The tracking errors, that are to be observed, are potentially in the sub-millimeter and sub-degree range. To capture such small errors, the uncertainty of the tester's pose estimation algorithm has to be smaller. In this context, the above uncertainties are too large.

There are also large off-diagonal values, particularly the terms $\mathrm{cov}(t_x, r_y)$ and $\mathrm{cov}(t_y, r_x)$. This is in agreement with the simulation observations: these pairs of translation and rotation have high covariances, i.e. they simultaneously have large variations from their expectation values. These covariances are a clear indication of the ambiguity between translations and rotations and seem to arise inherently from the statistical behaviour of the equation group. One may conclude that as is, the problem is ill-posed and has to be posed in a different way to yield useful results.

As it was noticed in the Monte Carlo simulations, entirely removing the $x$ and $y$ rotational degrees of freedom helped reduce the noise sensitivity of the system, at the obvious cost of losing two degrees of freedom. The effect can be also observed from the corresponding covariance matrix:

$$\mathbf{\Sigma_p} = \sigma_q^2 \begin{pmatrix} 0.11 & 0 & 0 & 0 \\ 0 & 0.11 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0.29 \end{pmatrix}, \tag{45}$$

where the order of variables is $t_x$, $t_y$, $t_z$, $r_z$. A particularly interesting observation is the decrease of the variances of $t_x$ and $t_y$ by several orders of magnitude, below the variance of $t_z$. There are also no non-zero off-diagonal elements, confirming that there are no ambiguities between the pose variables in this system.

A better posed problem, according to simulations, was formed by adding the off-planar point to the model. Therefore, error analysis should also show lower variances and covariances of the pose parameters. The pose covariance matrix for the improved model (37) with $l = 50\,\mathrm{mm}$, $d = 300\,\mathrm{mm}$ and $d' = 200\,\mathrm{mm}$ is

$$\mathbf{\Sigma_p} = \sigma_q^2 \begin{pmatrix} 2.15 & 0 & 0 & 0 & -0.45 & 0 \\ 0 & 2.15 & 0 & 0.45 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0.45 & 0 & 0.10 & 0 & 0 \\ -0.45 & 0 & 0 & 0 & 0.10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.29 \end{pmatrix},$$

where the values are orders of magnitude smaller than in the earlier one. Similarly,

the uncertainties of the output variables are smaller:

$$\begin{aligned}\sigma_{\mathbf{p}} &= \sigma_q \begin{pmatrix} 1.47 & 1.47 & 2.8 & 0.31 & 0.31 & 0.54 \end{pmatrix}^{\mathrm{T}} \\ &\overset{\sigma_q=0.2}{\approx} \begin{pmatrix} 0.29 & 0.29 & 0.57 & 0.06 & 0.06 & 0.11 \end{pmatrix}^{\mathrm{T}}.\end{aligned}$$

The $z$ rotation and translation values are unaffected, although they did not have as severe problems to begin with.

The error analysis confirms the findings from the simulations and thus gives encouragement to improve the actual measurement setup by adding a detectable off-plane point. The linear pose estimation method proved to be advantageous here: thanks to its relatively simple formulation, it is easy to analyze. It is noted, though, that the error analysis does not account for camera calibration errors or other error sources alongside coordinate errors.

To an extent, the presented analysis method is useful for analyzing PnP problems in general and detecting ambiguities caused by a poor problem statement (i.e. the target pattern). The results should be relevant at least for PnP solvers which use the pinhole camera model and minimize the reprojection error like done here.

## 5.6 Measuring the pose from a headset display

To find out how the newly developed linear algorithm and OpenCV's algorithm perform in practice, they were tested with instrumented test content on the HTC Vive headset. In the test content, a QR code is placed in front of the observer in the virtual world. Normally, when the user moves around in the operating space, the pose of the virtual camera is updated according to the user's motion. However, for testing and characterization purposes, a mode was implemented where the camera pose could be set manually to a known pose. This way, the behaviour of the pose estimators can be monitored in a scenario as close to real measurements as possible.

The first step in a measurement where the calibration matrix is not known beforehand (unlike the simulations), the camera must be calibrated. In this case, the calibration is performed by assuming the initial camera pose as the "zero" pose with $\mathbf{t} = \mathbf{0}$ and $\mathbf{R} = \mathbf{I}$. As a further assumption, the optical center point of the image is taken to be the arithmetic center point — for a $188 \times 120$ pixel image, this means $(c_x, c_y) = (94, 60)$. Thus, only the focal length $f$ is left to be found out.

Ideally, the focal length should be calibrated such that the location of the optical

center point does not affect the result. In this case, at least two image-model point pairs are required to find out $f$. Several images of the target pattern are captured in a pose where $\mathbf{t} = \mathbf{0}$ and $\mathbf{R} = \mathbf{I}$. In this case, the camera model equation (4) simplifies to

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

where it immediately follows that $s = z$, reducing the system to two equations:

$$z \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Writing this for the world-image point pairs with equal $z$ coordinates, $(\mathbf{x_1}, \mathbf{q_1}) = \left(\begin{pmatrix} x_1 & y_1 & z \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} u_1 & v_1 \end{pmatrix}^{\mathrm{T}}\right)$ and $(\mathbf{x_2}, \mathbf{q_2}) = \left(\begin{pmatrix} x_2 & y_2 & z \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} u_2 & v_2 \end{pmatrix}^{\mathrm{T}}\right)$, and subtracting the respective camera equations, one has

$$z \begin{pmatrix} u_2 - u_1 \\ v_2 - v_1 \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \end{pmatrix} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z - z \end{pmatrix} = f \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}.$$

Assuming $f$ and $z$ as positive, one may write

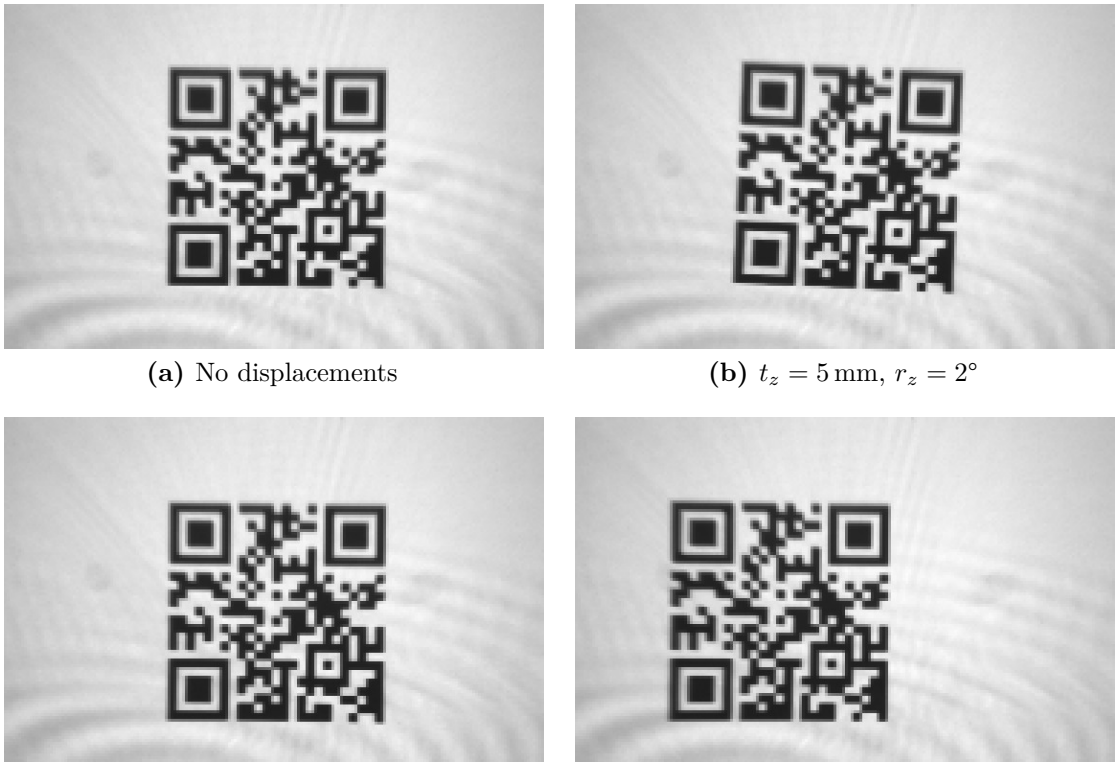
$$z \left\| \mathbf{q_2} - \mathbf{q_1} \right\| = f \left\| \mathbf{x_2} - \mathbf{x_1} \right\|,$$

and consequently,

$$f = z \frac{\left\| \mathbf{q_2} - \mathbf{q_1} \right\|}{\left\| \mathbf{x_2} - \mathbf{x_1} \right\|}. \tag{46}$$

This way, $f$ is conveniently calculated from the quotient of distances between two points in the world and image coordinates. The two diagonals of the QR code are good choices for point pairs used in the calibration. Further, it is desirable to capture several samples of all the points and average over these to yield a better estimate of the focal length.

In the measurements that were made with the HTC Vive, the QR code had



**(a)** No displacements

**(b)** $t_z = 5\,\text{mm}$, $r_z = 2°$

**(c)** $t_y = 5\,\text{mm}$

**(d)** $t_x = 5\,\text{mm}$, $t_y = 5\,\text{mm}$, $r_y = 2°$

**Figure 24.** Images of the QR code displayed on HTC Vive's display. The images were taken with the smart camera, using different manually set virtual camera poses. Pose displacements from origin are marked under each image.

50 mm side length and was situated 300 mm away from the observer. Several poses with different amounts of translations and rotations were tested. Examples of the camera output at different poses are shown in figure 24.

After positioning the headset on the robot so that the camera can see the display directly, the QR code was displayed. Based on the initial observations of the QR code with no displacements from the origin pose, the focal length was measured according to (46). The average length of the two diagonals in the image was 106.3 pixels while in the world coordinates the diagonal length is $\sqrt{2} \times 50\,\text{mm} \approx 70.7\,\text{mm}$. The model points were positioned at $z = 300\,\text{mm}$, and thus equation (46) gives $f = 451\,\text{px}$.

After calibration, a series of different camera poses were shown. QR code corner locations were captured and the pose was estimated in each situation based on this information. Both linear method and OpenCV method were used for the analysis, accompanied with a 4 degree-of-freedom version of the linear estimator that doesn't account for $x$ and $y$ rotations. The pose estimation errors for the samples are plotted

in figure 25.

The linear method shows a linear relationship between the translational and rotational error components. This is natural since any estimation error in a rotational component has its effect on the respective translational coordinate. This is a consequence of the covariance of rotations and translations. The OpenCV method shows generally similar behaviour.

The 4 degree-of-freedom linear method performs well with poses that do not have rotational displacements. This agrees with the simulation results in figure 20 where reducing the pose estimation to 4 DOF increased the accuracy of translational parts.

These observations confirm again the ambiguities caused by the planar target. To address this, a new target pattern with an off-plane point will be developed and utilized. According to the simulations, the noise performance would benefit from it due to the ambiguities being resolved.

A concern that rises with the concrete measurement setup is the reliability of the calibration. The focal length is relatively easy to determine, but it is harder to position the headset on the robot such that it points perpendicularly at the center of the display. Deviations from this setting can generate estimation errors in all of the methods that use the initial pose as a reference. A good improvement to the linear pose estimation algorithm would be taking the initial pose in account and compensating for it before presenting results.

## 5.7 Using pose estimation in tracking accuracy measurements

Tracking accuracy measurements can be performed using the pose estimation algorithm. Accuracy problems such as drifting and stationary jitter, described in section 2.2.2, can be characterized and identified by optically measuring the pose from the displayed content.

Given the design of the linear pose estimation algorithm, it is best to perform the accuracy measurements with respect to an initial pose — deviations are likely small and thus the linear approximation holds to reasonable precision. The robot, described in 3.1, wears the headset and is driven to a specified pose. A QR code is generated in the content in front of the virtual camera, the distance and size of the code set at specified values.

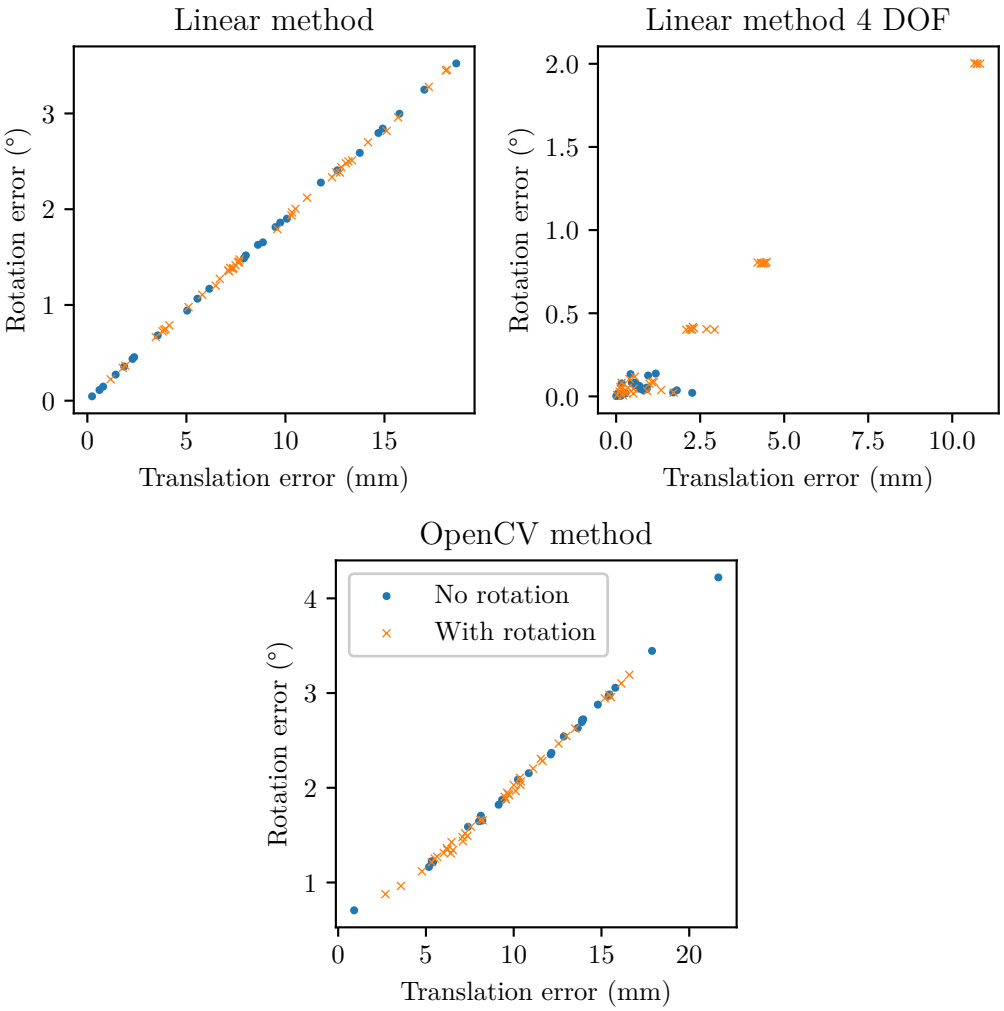



**Figure 25.** Pose estimation measurements from QR code target on HTC Vive. Each point represents an individual pose, its coordinates being the translational and rotational errors. Poses with and without rotational displacements are plotted with different markers.

Stationary jitter can be described by the variances and covariances of the pose variables in a situation where the headset stands still in a fixed pose. The covariances can be directly observed by taking a set of samples of the pose variables when the robot is not moving. In another measurement case, the tracking system could be simulated by performing a spontaneous, small move with the robot and quickly returning to the initial pose. In this case the samples could be taken immediately when the robot has stopped after the move, which can be observed from encoder signals.

The covariances can be estimated from a sequence of samples using the formula

$$\sigma_{xy} = \frac{1}{N} \sum_{i=1}^{N} (x - \overline{x})(y - \overline{y}), \tag{47}$$

where $x$ and $y$ are any of the pose variables and $N$ is the number of samples. In the measurements, the intrinsic noise of the QR code detection has to be taken in account. Jitter estimates close to the intrinsic jitter values are not reliable and should be interpreted with care. The intrinsic noise of the target detection and pose estimation can be observed either by direct measurements of a static target (e.g. QR code printed on paper) or by using the general law of error propagation, as described in section 5.5.

Drifting of the pose variables can be observed from a similar time series of pose samples. Tracking estimation error could start accumulating in some of the pose variables and be seen as a decreasing or increasing curve. Another interesting aspect of drifting can be probed by simulating the tracking system by a sudden sequence of motions performed by the robot, returning to the initial pose. The sequence could be performed many times, estimating the pose each time after the sequence. This is a good way to find out if the operation of the tracking system is deterministic, yielding the same pose each time, or if the error accumulates in some of the pose variables.

As of writing this, realizing the actual measurement sequences is still underway. However, some measurements with the robot and HTC Vive have already been performed. In figure 26, a simple sequence of translational $x$, $y$ and $z$ motions was performed with the robot, recording the encoder positions from the axes and simultaneously recording the QR code location. The pose at each sample was estimated afterwards by the linear method in four degrees of freedom, due to an improved target not being implemented yet. The sequence was repeated with both

HTC Vive and a static, printed QR code as targets, the printed code serving as a "ground truth" reference target. The measurement setups are shown in figure 27.

The figures show some interesting behaviour. On the $x$ and $z$ axes, the estimation mainly behaves very well, but the $y$ axis shows a greater amount of noise. This is due to the $y$ axis being parallel with the camera's $z$ axis where the pose estimation is most sensitive to coordinate noise, as indicated by the covariance matrix (45).

Generally, the results look good — the estimated pose follows the encoder curves and returns to zero when the robot returns to the initial pose. The HTC Vive data shows a slight offset from the encoder curve on the $x$ and $z$ axes, probably due to poor initial calibration in the origin pose. In the $x$ and $z$ axes, some deviation from the encoder peak values is seen where the $y$ axis (camera $z$) has moved from the initial pose. This is due to the non-linearity of the camera model with respect to $z$ which causes a change of scale and therefore deviation in the $x$ and $y$ coordinates. If there had been rotational displacements in the sequence, mixing of the coordinates would have been also seen.

The estimated $x$ and $z$ locations for the static target seem to have less noise than the respective measurements with HTC Vive. Stationary jitter of the HTC Vive's tracking algorithm is a possible cause of the noise. After the motion sequence, the Vive $x$ and $z$ signals drift away from the initial value.

There is some noise also in the static target signals, probably due to the QR code location estimate varying with image noise. In the cusps of the motion (with a high acceleration), there is some oscillatory behaviour which could originate from the robot mechanics.

The core measurement principle seems to work, and the initial data already reveals behaviour which is likely caused by stationary jitter and drifting. The linear pose estimation method will likely work well in the measurement cases at hand (when extended to six degrees of freedom using the improved target). If more demanding cases arise, OpenCV's solver can be used or the linear algorithm can be extended to a non-linear one. An interesting topic of further research is the use of another rotation formalism for the non-linear optimization — there are several rotation representations to choose from. Different representations and an alternative, on-manifold optimization formalism are presented in [49].

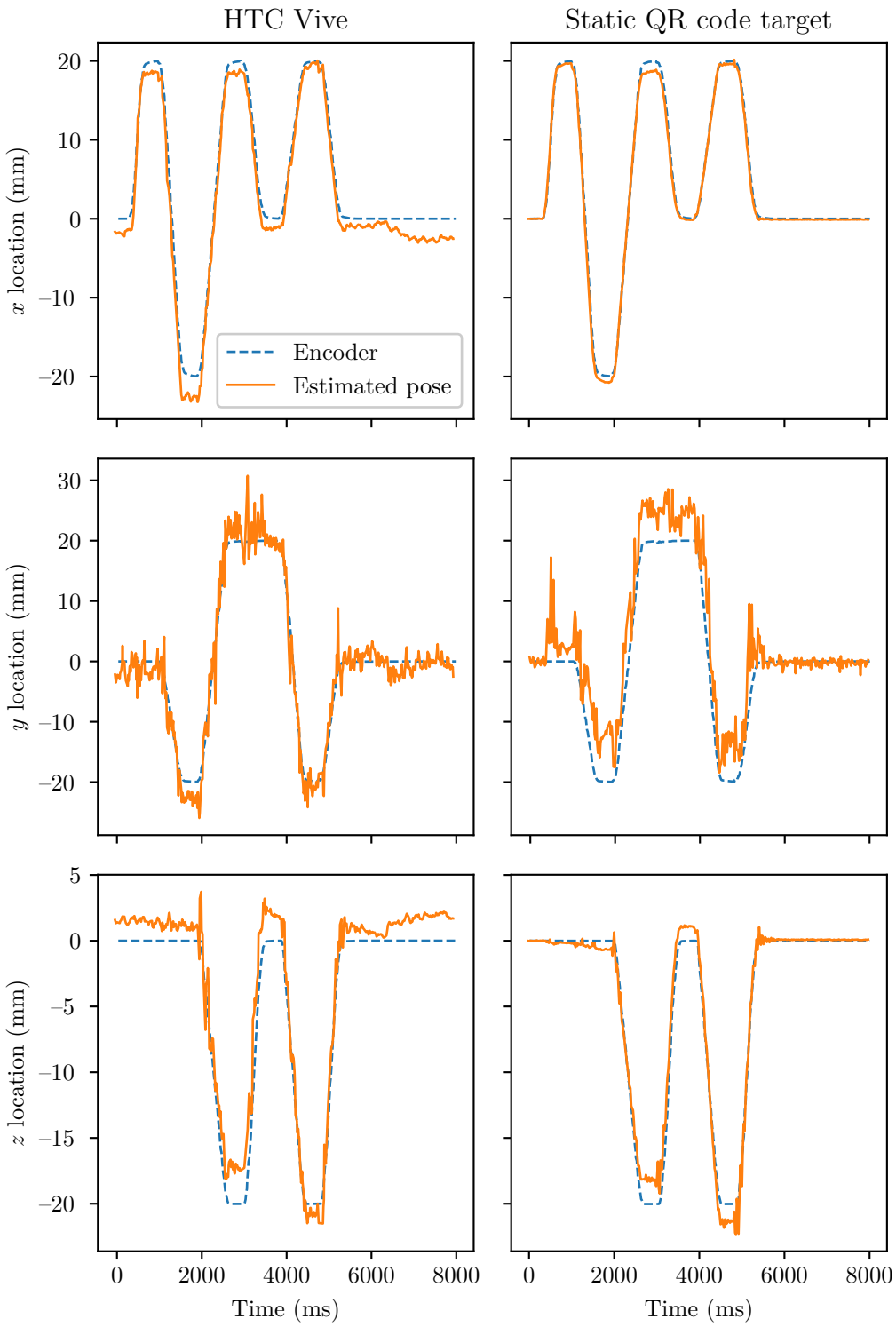
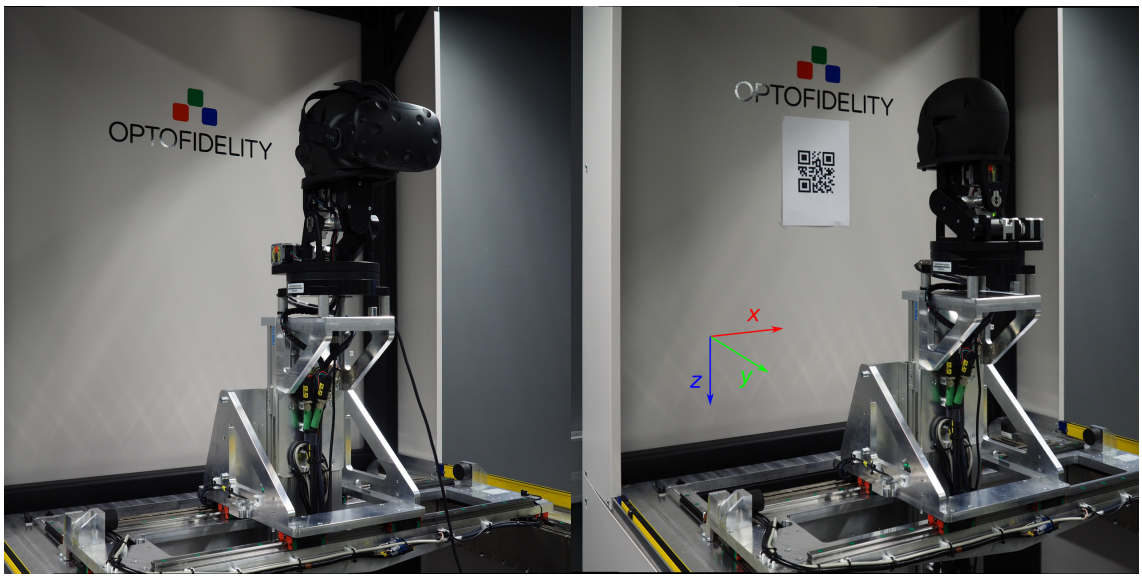



**Figure 26.** Measurements of the estimated pose versus robot position (encoder signal) with HTC Vive and a static QR code target. The translational pose components were transformed to the robot coordinate system (see figure 27.)




**(a)** HTC Vive **(b)** Static QR code

**Figure 27.** Measurement setups used for pose estimation measurements with HTC Vive and the static QR code. The robot's Cartesian coordinate system axes are also indicated.

# 6 Conclusions

Developing a smart camera software is an involved process. In addition to the algorithms and the architecture described in the thesis, the vast majority of the work consisted of firmware development and debugging. However, the requirement of synchronizing image capture to a low persistence display is not readily fulfilled by most conventional machine vision cameras. Given the existing measurement technology, backlight synchronization algorithms and experience with signal processing on microcontrollers, building a smart camera was a natural choice for OptoFidelity. The work has been successful: copies of the smart camera have already been integrated in systems delivered to customers. The camera also has potential for use outside of the VR measurements.

The architectural choice of using USB and Ethernet emulation for communication between the master and slave devices has made test sequence development pleasant. Sending HTTP requests to subnetworks is very easy in e.g. Python, and it has made the development the master side software straightforward, not having to implement any low-level drivers or communication protocols on PC. Synchronization of the clocks of several USB devices has been proven to work correctly and provides a solid basis for measurements where data from different devices is combined.

The microcontroller STM32F427 and the camera sensor MT9V034 have been found to work well together. So far, the sensor has been used at $188 \times 120$ resolution and eight bit depth. An immediate requirement to upgrade either of these is not predicted, and there is more resolution and bit depth to be utilized if necessary.

The implementation of the autofocus algorithm was straightforward. The Sobel operator magnitude for local sharpness estimation, combined with the percentile method for global sharpness estimation, provides a reliable sharpness estimator which is sensitive to small sharp areas but also tolerates relatively high amounts of noise. It has also worked very reliably since the first implementation.

A new algorithm for pose estimation, to be used in measuring the tracking accuracy of VR headsets, was developed from first principles using the pinhole camera model and linear least squares optimization. The algorithm calculates the

rotation and translation of the camera based on the points of a three-dimensional model that are projected onto the image plane. As the algorithm is a critical part of the optical measurement system, it was analysed thoroughly by statistical methods and Monte Carlo simulation. In the analysis it turned out that the planar QR code target caused ambiguities between translations and rotations.

Simulations using the camera model made it possible to experiment with various target geometries and the algorithmic properties in the simulations. It turned out that planar targets are especially prone to ambiguities, and adding a single off-plane point to the target greatly reduces the risk of ambiguities and improves the accuracy of results. This finding led to improving the target pattern in the measurement setup.

With the improved target pattern, the linear pose estimation method performed as well as OpenCV's non-linear, iterative PnP solver in the region of small displacements. Both algorithms showed similar response to coordinate noise, indicating that the noise sensitivity is likely intrinsic to the problem statement, i.e. the construction of the target model. Outside the region of small displacements, especially on the translational $z$ and the rotational degrees of freedom, the linearization began to cause large errors. It is, however, reasonable to assume that the errors in the tracking system stay within boundaries where the linear approximations hold. A clear advantage of the linear pose estimated method is the ease of analyzing it statistically. Error estimations could be easily derived using the general law of error propagation.

The six degree of freedom VR measurement project at OptoFidelity is still ongoing. The next steps are implementing the actual measurement sequences, performing ground truth characterization of the system and doing tests with several devices such as HTC Vive. Beyond objective end-to-end measurements, an interesting field of research would also be the psychovisual effects, briefly described in 2.2.4, that take place when experiencing virtual reality content. Ideally, a model could be developed which would accurately quantify the perceived quality of the virtual reality experience.

# References


[1] C. Cruz-Neira, D. J. Sandin and T. A. DeFanti. 'Surround-screen projection-based virtual reality: the design and implementation of the CAVE'. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM. 1993, pp. 135–142. DOI: 10.1145/166117.166134.

[2] HTC Corporation. *VIVE™ | VIVE Virtual Reality System*. 2017. URL: https://www.vive.com/us/product/vive-virtual-reality-system/ (visited on 11/10/2017).

[3] Acer Inc. *AH101-D8EY | Virtual & Mixed Reality - Tech Specs & Reviews - Acer*. 2018. URL: https://www.acer.com/ac/en/US/content/model/VD.R05AP.002 (visited on 08/04/2018).

[4] *File:HTC Vive (2).jpg - Wikimedia Commons*. 2015. URL: https://commons.wikimedia.org/wiki/File:HTC_Vive_(2).jpg (visited on 08/04/2018).

[5] *VR Multimeter*. OptoFidelity. 2017. URL: https://www.optofidelity.com/wp-content/uploads/2017/10/OF_VR-Multimeter.pdf (visited on 07/11/2017).

[6] Sony Interactive Entertainment LLC. *Tech Specs*. 2017. URL: https://www.playstation.com/en-us/explore/playstation-vr/tech-specs/ (visited on 11/10/2017).

[7] *Visual acuity measurement standard*. Tech. rep. International Council of Ophthalmology, 1988. URL: http://www.icoph.org/dynamic/attachments/resources/icovisualacuity1984.pdf (visited on 17/10/2017).

[8] S. M. LaValle. *Virtual reality*. 2016. URL: http://vr.cs.uiuc.edu/ (visited on 12/04/2018).

[9] H. Kolb. *Facts and Figures concerning the human retina*. 2017. URL: http://webvision.med.utah.edu/book/part-xiii-facts-and-figures-concerning-the-human-retina/ (visited on 19/10/2017).

[10] J. Bergquist. 'Display-induced motion artefacts'. In: *Seminar at the Media Technology Laboratory, Helsinki University of Technology*. 2007. URL: https://www.researchgate.net/profile/Johan_Bergquist/publication/268003997_Display-induced_motion_artefacts/links/546bf50a0cf2397f7831cab1.pdf (visited on 17/10/2017).

[11] Microsoft. *HoloLens hardware details*. 2017. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/hololens_hardware_details (visited on 12/12/2017).

[12] S. M. LaValle et al. 'Head tracking for the Oculus Rift'. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 187–194. DOI: 10.1109/ICRA.2014.6906608.

[13] M. Ribo, A. Pinz and A. L. Fuhrmann. 'A new optical tracking system for virtual and augmented reality applications'. In: *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*. Vol. 3. 2001, 1932–1936 vol.3. DOI: 10.1109/IMTC.2001.929537.

[14] *Hologram stability - Mixed Reality | Microsoft Docs*. 2018. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/hologram-stability (visited on 12/04/2018).

[15] *Oculus best practices*. Tech. rep. Oculus VR, LLC, 2017. URL: http://static.oculus.com/documentation/pdfs/intro-vr/latest/bp.pdf (visited on 02/11/2017).

[16] *Unpredictable movement performance of Virtual Reality headsets*. Tech. rep. OptoFidelity, 2018. URL: https://www.optofidelity.com/wp-content/uploads/2018/03/Unpredictable-movement-performance-of-Virtual-Reality-headsets.pdf (visited on 17/04/2018).

[17] H. Akiduki et al. 'Visual-vestibular conflict induced by virtual reality in humans'. In: *Neuroscience Letters* 340.3 (2003), pp. 197–200. ISSN: 0304-3940. DOI: 10.1016/S0304-3940(03)00098-3.

[18] M. Regan and G. S. Miller. 'The Problem of Persistence with Rotating Displays'. In: *IEEE Transactions on Visualization and Computer Graphics* 23.4 (2017), pp. 1295–1301. DOI: 10.1109/TVCG.2017.2656979.

[19] P. Bakaus. *The Illusion of Motion - The Sea of Ideas*. URL: https://paulbakaus.com/tutorials/performance/the-illusion-of-motion/ (visited on 17/04/2018).

[20] R. VanRullen. *Neuronal Dynamics of Vision: Perception, Attention & Consciousness*. 2017. URL: http://www.cerco.ups-tlse.fr/~rufin/ (visited on 12/12/2017).

[21] *Video Multimeter*. OptoFidelity. 2016. URL: https://www.optofidelity.com/files/uploads/2016/09/OF_leaflet_VideoMultimeter.pdf (visited on 07/11/2017).

[22] P. Aimonen. 'Design of an embedded system for video performance measurements'. MA thesis. Tampere University of Technology, 2014. URL: http://urn.fi/URN:NBN:fi:tty-201402201086.

[23] J. Zhao et al. 'Estimating the motion-to-photon latency in head mounted displays'. In: *Virtual Reality (VR), 2017 IEEE*. IEEE. 2017, pp. 313–314. DOI: 10.1109/VR.2017.7892302.

[24] K. Raaen and I. Kjellmo. 'Measuring Latency in Virtual Reality Systems'. In: *Entertainment Computing - ICEC 2015: 14th International Conference, ICEC 2015, Trondheim, Norway, September 29 - Ocotober 2, 2015, Proceedings*. Ed. by K. Chorianopoulos et al. Cham: Springer International Publishing, 2015, pp. 457–462. ISBN: 978-3-319-24589-8. DOI: 10.1007/978-3-319-24589-8_40.

[25] Oculus. *Latency Tester Pre-Orders Now Open!* 2013. URL: https://www3.oculus.com/en-us/blog/latency-tester-pre-orders-now-open/ (visited on 07/11/2017).

[26] *VRTrek + Libraries - Basemark Products*. 2018. URL: https://www.basemark.com/products/vrtrek-library/ (visited on 12/04/2018).

[27] *ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera*. ST Microelectronics. 2015. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf (visited on 02/02/2018).

[28] D. Honegger et al. *An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications*. Tech. rep. ETH Zürich. URL: https://www.inf.ethz.ch/personal/pomarc/pubs/HoneggerICRA13.pdf (visited on 15/01/2018).

[29] *JeVois Smart Machine Vision Camera*. 2018. URL: http://jevois.org (visited on 15/01/2018).

[30] JeVois. *Camera Sensor Information — JeVois Tech Zone*. 2017. URL: http://jevois.org/qa/index.php?qa=63&qa_1=camera-sensor-information&show=96#a96 (visited on 15/01/2018).

[31] J.-P. Lang. *CMUcam5 Pixy*. 2018. URL: http://www.cmucam.org/projects/cmucam5 (visited on 15/01/2018).

[32] *OpenMV Cam M7*. 2018. URL: https://openmv.io/collections/products/products/openmv-cam-m7 (visited on 15/01/2018).

[33] *NuttX Real-Time Operating System*. 2018. URL: http://www.nuttx.org (visited on 15/01/2018).

[34] MIPI Alliance, Inc. *Camera and Imaging*. 2018. URL: https://mipi.org/specifications/camera-and-imaging (visited on 21/01/2018).

[35] *Reference manual - STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs*. RM0090. ST Microelectronics. 2017. URL: http://www.st.com/resource/en/reference_manual/DM00031020.pdf (visited on 21/01/2018).

[36] *File:Rolling shutter SMIL.svg - Wikimedia Commons*. 2015. URL: https://commons.wikimedia.org/wiki/File:Rolling_shutter_SMIL.svg (visited on 21/01/2018).

[37] *MT9V034: 1/3-Inch Wide-VGA CMOS Digital Image Sensor*. MT9V034/D. ON Semiconductor. 2017. URL: http://www.onsemi.com/pub/Collateral/MT9V034-D.PDF (visited on 21/01/2018).

[38] *LUPA1300-2: High Speed CMOS Image Sensor*. NOIL2SM1300A/D. ON Semiconductor. 2015. URL: https://www.onsemi.com/pub/Collateral/NOIL2SM1300A-D.PDF (visited on 21/01/2018).

[39] *OV9281-OV9282 1-megapixel product brief*. OmniVision. 2017. URL: http://www.ovt.com/download/sensorpdf/207/OmniVision_OV9282.pdf (visited on 21/01/2018).

[40] S. Kuiper and B. H. W. Hendriks. 'Variable-focus liquid lens for miniature cameras'. In: *Applied Physics Letters* 85.7 (2004), pp. 1128–1130. DOI: 10.1063/1.1779954.

[41] Corning Inc. *Auto Focus Lens Modules: C-S-Series*. 2018. URL: https://www.corning.com/worldwide/en/innovation/corning-emerging-innovations/corning-varioptic-lenses/auto-focus-lens-modules-c-s-series.html (visited on 15/03/2018).

[42] H. Mir, P. Xu and P. Van Beek. 'An extensive empirical evaluation of focus measures for digital photography'. In: *Digital Photography*. 2014, p. 90230I. DOI: 10.1117/12.2042350.

[43] J. F. Brenner et al. 'An automated microscope for cytologic research a preliminary evaluation.' In: *Journal of Histochemistry & Cytochemistry* 24.1 (1976), pp. 100–111. DOI: 10.1177/24.1.1254907.

[44] OpenCV developer team. *Camera Calibration and 3D Reconstruction*. 2018. URL: https://docs.opencv.org/3.4.1/d9/d0c/group__calib3d.html (visited on 26/03/2018).

[45] R. M. Murray. *A mathematical introduction to robotic manipulation*. CRC press, 1994. URL: http://www.cds.caltech.edu/~murray/books/MLS/pdf/mls94-complete.pdf.

[46] E. W. Weisstein. *Sphere point picking*. URL: http://mathworld.wolfram.com/SpherePointPicking.html (visited on 02/04/2018).

[47] G. Schweighofer and A. Pinz. 'Robust Pose Estimation from a Planar Target'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.12 (2006), pp. 2024–2030. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2006.252.

[48] K. O. Arras. *An introduction to error propagation: derivation, meaning and examples of equation* $C_Y = F_X C_X F_X^T$. Tech. rep. ETH Zurich, 1998. URL: http://srl.informatik.uni-freiburg.de/papers/arrasTR98.pdf (visited on 03/04/2018).

[49] J.-L. Blanco. *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*. Tech. rep. University of Malaga, 2010. URL: http://ingmec.ual.es/~jlblanco/papers/jlblanco2010geometry3D_techrep.pdf (visited on 17/04/2018).