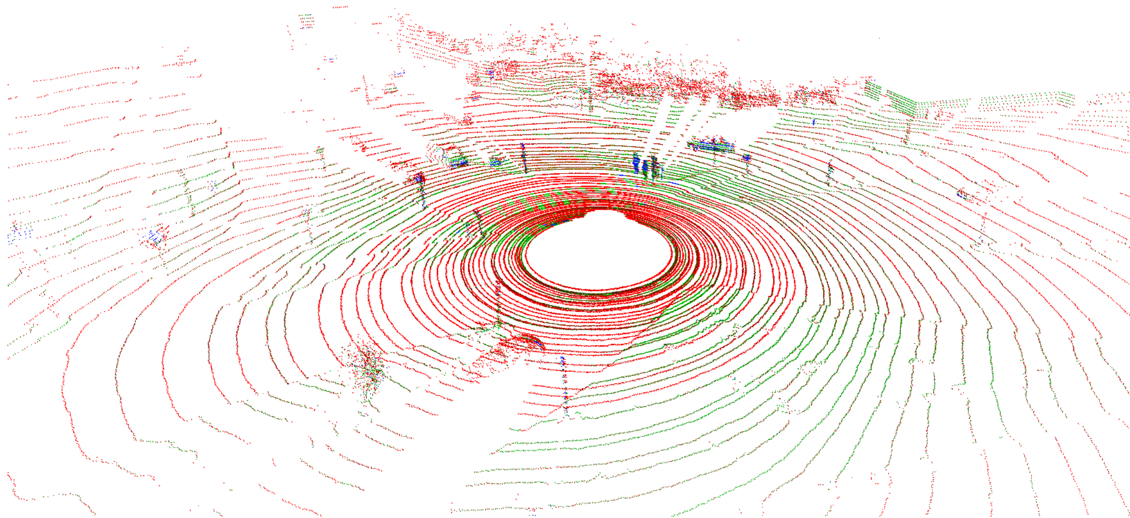
CHALMERS
UNIVERSITY OF TECHNOLOGY



Vehicle sensor data real time visualizer

Master's thesis in Systems, Control and Mechatronics

Erik Karlsson
Henrik Olsson

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:EX007

Vehicle sensor data real time visualizer

Erik Karlsson
Henrik Olsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Vehicle sensor data real time visualizer
Erik Karlsson
Henrik Olsson

© Erik Karlsson, Henrik Olsson, 2016.

Supervisors:
Micael Axestål, Consultant at ÅF - Product Development
Petter Falkman, Department of Signals and Systems

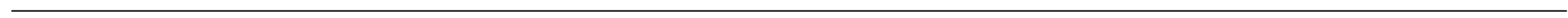
Examiner:
Petter Falkman, Department of Signals and Systems

Master's Thesis 2016:EX007
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Abstract

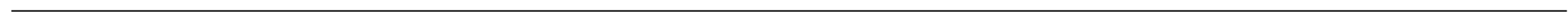
This report seeks to determine if virtual or augmented reality is suitable for use in a sensor verification tool. The purpose of the tool is to verify the output of sensors reading surroundings of autonomous cars, as well as the corresponding object detection algorithms. This was done through research and by implementing such a tool with features that could not be evaluated through research. Verification is performed by comparing a reference with the output from the sensors. The resulting tool uses two modes, off-line for desktop verification and on-line for verification during test drives.

The off-line mode shows a virtual reality in Oculus Rift where the reference is shown as a point cloud stream produced by a Lidar scanner. The on-line mode mimics augmented reality by having a stereoscopic camera feed in Oculus Rift which acts as the reference. The objects detected by the sensors are shown as color coded cuboids for both off-line and on-line. The tool itself was developed in the game engine Unity. An interesting area of discussion is the simulator sickness, induced by the virtual reality tool. There are a lot of factors affecting simulator sickness, e.g. frame rate, latency, field of view, and how and if the view is changed by other than bodily movements such as head rotations. As long as the simulator sickness is sufficiently suppressed, virtual reality could be beneficial for an off-line verification tool since the user gets a reliable overview of the vehicle's surroundings faster than by using an ordinary monitor. Augmented reality for on-line verification has potential, but the mimicked augmented reality implemented in this report require improvements for real verification purposes.



Acknowledgements

First and foremost, we would like to thank Micael Axestål, our supervisor at ÅF, who provided us with guidance and support throughout the project. We would also like to thank our supervisor and examiner at Chalmers, Petter Falkman, who gave us much appreciated feedback. Furthermore, we show our appreciation to Jonas Kurol, our sponsor at ÅF, for all the support. We would also like to thank ÅF, as a whole, for proposing this Master's Thesis and for having us. All the test data in this project and the car in which the on-line tests were performed were provided by Volvo Cars, for which we are very grateful. Finally, we thank everyone who took the time to test our visualization tool and gave important feedback and tips regarding further development.



Contents

1 Introduction **1**
1.1 Background 1
1.2 Purpose 1
1.3 Objectives 2
1.3.1 Primary objectives 2
1.3.2 Secondary objectives 2
1.4 Project overview 3
1.5 Methods 4
1.5.1 Literature study 4
1.5.2 Iterative process 4
1.5.3 Testing with test subjects 4

2 Related work **7**
2.1 Head mounted displays 7
2.1.1 Field of view 7
2.1.2 Simulator sickness 8
2.2 Simulator sickness questionnaire 9
2.3 Augmented reality 10
2.4 Virtual reality 11
2.5 Human - technical system interface 12
2.5.1 System output 12
2.5.2 System input 13
2.5.2.1 Game controller 13
2.5.2.2 Leap motion 13
2.6 Game engine 14
2.6.1 GPU versus CPU programming 14
2.7 Point clouds 14
2.7.1 Point Cloud Library 15
2.8 Virtual augmented reality using cameras 15
2.8.1 Camera placement 16
2.8.2 Image distortion 18

3 Shared aspects for off-line and on-line verification **19**
3.1 Visualization of data and reference 19
3.2 Sensor configuration 20

3.3 Sensor data for off-line verification . . . 21
3.3.1 Reference . . . 21
3.3.2 Object data . . . 22
3.4 Sensor data for on-line verification . . . 23

4 Design of off-line system **25**
4.1 Choosing VR glasses . . . 25
4.2 Choosing game engine . . . 26
4.3 Implementation of graphics and functionality . . . 27
4.3.1 Reference: Point clouds . . . 27
4.3.1.1 Dynamic Link Library . . . 27
4.3.2 Reference: Mesh . . . 27
4.3.2.1 Normals . . . 28
4.3.2.2 Indexing limitations . . . 29
4.3.2.3 Time required to generate mesh . . . 29
4.3.2.4 Point placements . . . 29
4.3.3 Sensor to verify: Radar-camera . . . 29
4.3.4 Control functions . . . 30
4.3.5 Program structure . . . 30
4.4 User input to visualization tool . . . 31
4.5 Performance . . . 32
4.6 Testing . . . 33
4.6.1 Evaluation of the off-line system . . . 34
4.6.2 Results of the simulator sickness questionnaire . . . 35
4.6.3 Further development of the verification tool . . . 35

5 Design of on-line system **37**
5.1 Stereoscopic cameras mounted on Oculus Rift . . . 37
5.1.1 Choice of cameras . . . 38
5.1.2 Camera mounting . . . 38
5.2 Camera feed in Unity . . . 39
5.2.1 Properties with Unity's VR compatibility . . . 41
5.2.2 Stereoscopic visualization . . . 42
5.3 Object data and simulation of on-line verification . . . 42
5.3.1 Additional settings . . . 43
5.4 Performance . . . 43
5.5 Testing . . . 43

6 Results **45**
6.1 The resulting tool . . . 45
6.2 Comparing objectives with results . . . 45
6.2.1 Primary objectives . . . 45
6.2.2 Secondary objectives . . . 46

7 Discussion **51**
7.1 Usability of the tool for verification . . . 51
7.1.1 Selection of test subjects for off-line verification . . . 51

7.1.2 Selection of test subjects for on-line verification 51
7.1.3 Performance during on-line test drive 52
7.2 Simulator sickness 52
7.2.1 Temperature 52
7.2.2 Quality of computer 52
7.2.3 Quality of Oculus Rift Development Kit 2 53
7.2.4 Cameras for on-line verification 53
7.3 Choice of 3D development environment 53
7.4 Alternative approach and project content 54
7.5 Extended use of the verification tool 54

8 Conclusion 55

A Appendix I
A.1 Survey I
A.2 SSQ results V
A.3 Possible future implementations VI
A.3.1 Cameras on Oculus Rift VI
A.3.2 Color overlay on point cloud or mesh/surface VI
A.3.3 Static mesh VII
A.3.4 Mesh from point cloud cluster VIII

1 Introduction

This report is a Master's Thesis at Chalmers University of Technology performed in cooperation with ÅF, a Swedish consultant company. The project aims at creating a verification tool for sensors and object detection algorithms in autonomous cars. This chapter contains the purpose and problem description of the report. The chapter ends by explaining the various methods used in the report.

1.1 Background

Autonomous vehicles is a popular area of development for many vehicle companies [1]. By using various sensors, such as Lidars and ultrasonic sensors, the cars are able to orient themselves and be aware of their surroundings which allow them to drive in a safe and efficient way [2]. The possibility also exists to have communication between vehicles, which enable them to match each other's velocities as well as communicate about obstacles on the road for a better planned driving. However, to be able to phase out the current vehicles and increase the usage of self-driving vehicles, both drivers as well as other road-users have to be able to trust the vehicles' performances and their ability to drive safely. This means that it is important to verify the reliability of the cars' sensors together with the associated object detection algorithms and their ability to detect various objects in different environments.

1.2 Purpose

The purpose of this report is to investigate if it could be beneficial to use virtual or augmented reality as a tool for verification of sensors and object detection algorithms in autonomous cars.

1.3 Objectives

This project aims at creating a visualization tool to verify that a self-driving car is detecting all objects of interest in its current surrounding that it is supposed to detect. This will be done by displaying for an operator, with the help of virtual reality (VR) and/or augmented reality (AR), what the car is seeing as well as displaying a reference, the real environment. The report is aiming at proving a concept, therefore as many features as possible of such a tool will be tested.

The visualization tool shall be evaluated and used as a basis when answering the question, is it beneficial to use VR or AR technology to verify the output of sensors and object detection algorithms in self-driving cars. In order for the tool to be usable it will be developed to fulfil the primary objectives listed below. If time permits, the tool will also be developed to meet the secondary objectives. The secondary objectives are more related to having a practical solution that is usable than just proving the concept.

1.3.1 Primary objectives

- The tool shall in a clear manner visualize the close surroundings of the car in stereoscopic 3D. At each point in time in the data input stream, the user of the tool shall be able to distinguish all, for the car, relevant objects within the user's near peripheral field [3].
- The tool shall mark detected objects in the 3D space so that a user easily can see which objects that are detected and which are not. The user shall be able to see all markings if they are within the user's near peripheral field.
- The tool shall perform the visualization in real-time and with a sufficiently short time delay as to not induce simulator sickness.

1.3.2 Secondary objectives

- If several sensors are used to detect an object, it shall be clear by which sensors an object is detected. This means that a user with limited experience with the tool shall, by looking at a detected object, be able to tell by which sensors the object was detected.
- The interface shall include a method for logging misdetections. An example of possible log information is a time stamp together with the coordinates of the non-detected object's location.

1.4 Project overview

An overview of the parts included in the project can be seen in Figure 1.1. The project started with a study of related work, which can be found in chapter 2, to gain knowledge about the state of the art and to be able to design a feasible solution. The system was then implemented through an iterative process.

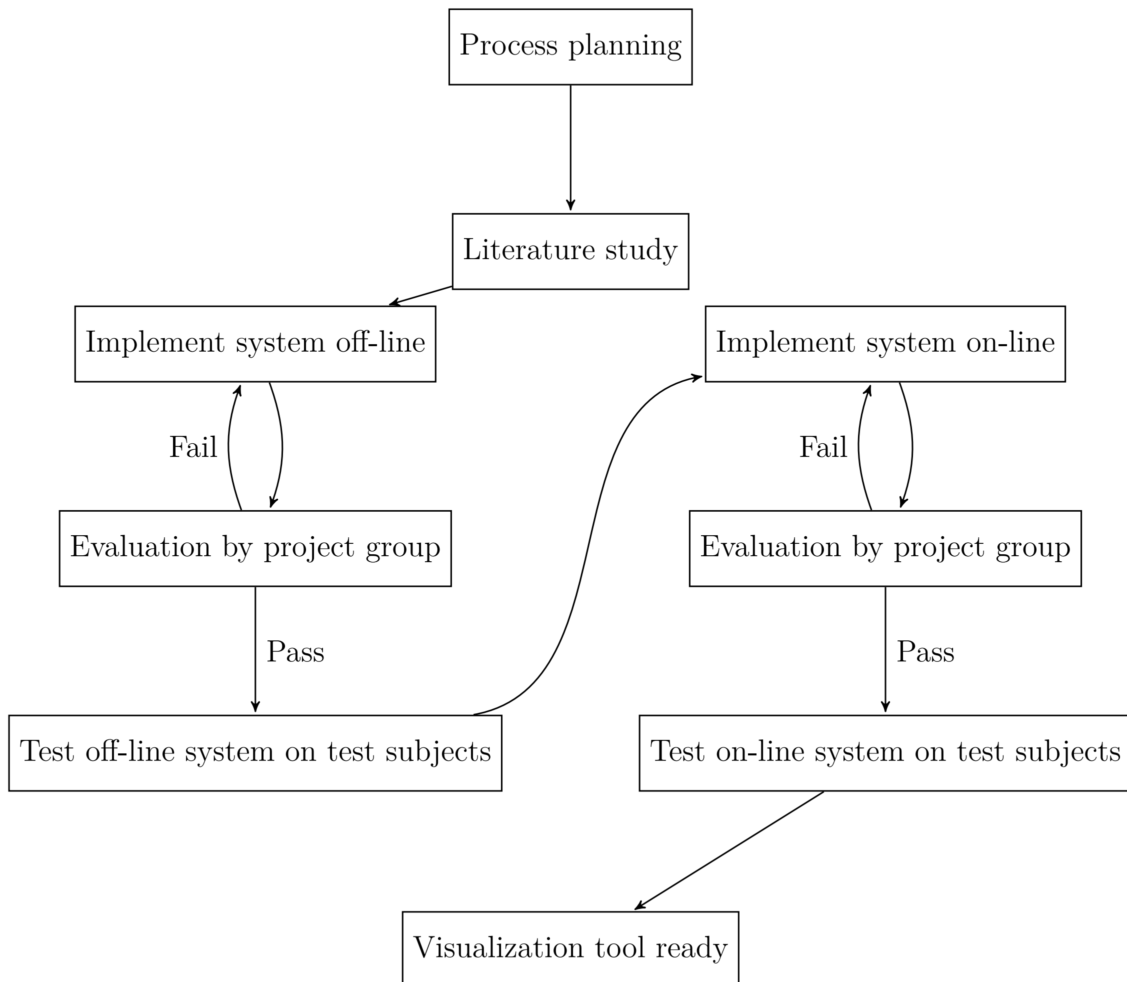


Figure 1.1: An overview of the project

The tool was designed to work with, but is not limited to, the Volvo cars from the project Drive me[4]. The Drive me project has as goal to have 100 self-driving cars of model XC90s equipped with IntelliSafe Autopilot driven by ordinary people on Swedish roads by 2017 [5]. Firstly, a prototype of the tool working for off-line verification was developed. This means that the verification will be performed at the desk with pre-recorded data. Both high resolution sensor data about the environment, referred to as the reference, and data from the sensors that shall be verified is accessible. After necessary functions for off-line verification were implemented, the tool was developed for on-line verification, which can be performed directly in

the car during test drives. All sensor data is recorded during a test drive and made available for off-line verification afterwards.

Evaluations of the system were performed iteratively throughout the project. These were conducted by the project group. A test with test subjects took place after both the off-line and on-line implementation. The outcome of the tests at the end of the off-line implementation were to influence the development of the on-line implementation. The tests at the end of the on-line implementation was a part of the final evaluation of the system, to check if the objectives were fulfilled or not.

1.5 Methods

In this section a description of the methods used in the project are given. The first method is literature study, a way of obtaining information. The second method, iterative process, helps making product development more agile. The final method, testing with test subjects, is used for evaluating the visualization tool.

1.5.1 Literature study

A literature study is performed in order to gain knowledge about work and theories related to a project. This knowledge is then used as basis when decisions are made. Usually the study is performed at the start of a project and allows a project to take advantage of the work of others.

1.5.2 Iterative process

Iterative process is the technique of developing incremental changes to a product repeatedly. This is most commonly used in agile software development where changes to the specification can occur during the development phase [6].

1.5.3 Testing with test subjects

Using test subjects to evaluate a system can often give a very wide view of the system's properties. After the test some sort of survey usually takes place. Two examples of surveys are questionnaires and interviews [7], which both have advantages and disadvantages for different applications.

Questionnaires:

- Possible to gather data from a large number of people in a short amount of time.
- A relatively good way to gather data from people who are difficult to reach.
- It is not a direct method, no personal contact with people.

Interviews:

- A versatile method which can obtain knowledge about people's experiences and their opinions, and gives an understanding about their reasoning.
- Mainly a qualitative method but could be quantitative depending on the structure.
- Requires a large amount of resources when performed in a large scale.

2

Related work

This section contains work related to the project that stood as a basis for decisions made during the development of the verification tool. Firstly, aspects of head mounted displays are described. This is followed by a simulator sickness questionnaire, a way of calculating how nauseating a virtual experience is. The following sections explain the terms augmented and virtual reality. The next section contains information about how to design a human - technical system interface. Furthermore, a description is given of what a game engine and a point cloud is. The chapter ends by describing how an augmented reality can be achieved by using virtual reality along with two cameras.

2.1 Head mounted displays

A head mounted display (HMD) is a device attached to the head and contains a screen used to show imagery to a user. A HMD can also include a tracking system that gives the user the ability to control the imagery by head movements. Important aspects of HMDs include the field of view and, if a tracking system is used, time lag between bodily and visual movement, which affect the user experience.

2.1.1 Field of view

Field of view (FOV) is a measurement of the angular dimension of the vision, often measured in degrees. When nothing else is mentioned, the measurement represent the diagonal FOV, for example the angle between the upper left and the lower right corner of an image. It is, however, important to be aware of the two different types of FOVs regarding VR. The screen field of view describes the perceived size of the screen and the virtual field of view describes the perceived size inside the 3D environment [8] captured by a virtual camera, see Figure 2.1 and 2.2.

It is shown that in reality a narrow FOV has a negative effect on the human navi-

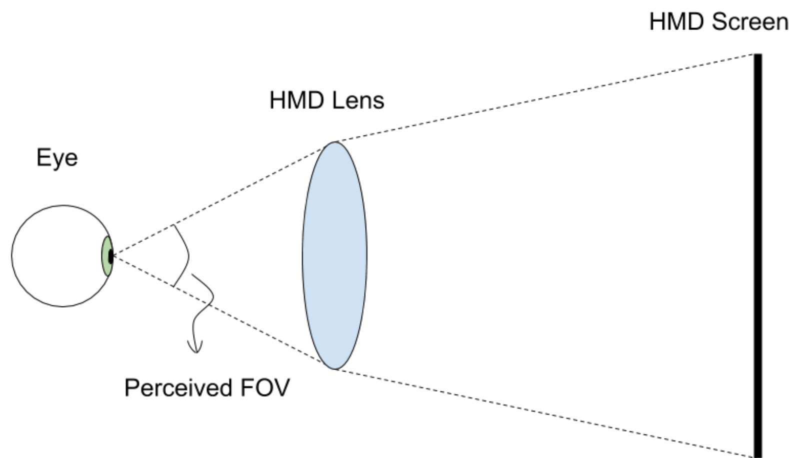
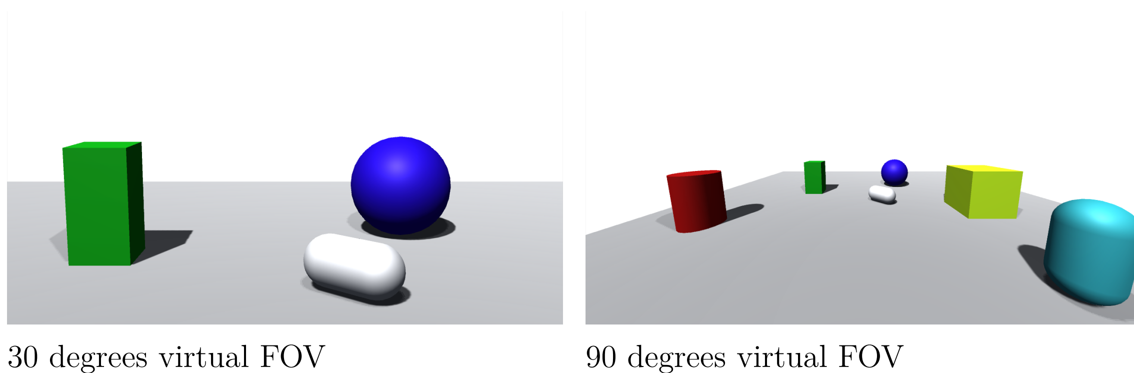


Figure 2.1: Illustration of the perceived field of view



30 degrees virtual FOV   90 degrees virtual FOV

Figure 2.2: Illustration of the virtual field of view. Both images are taken from the same position

gation, spatial awareness, and visual search tasks [9]. Even if the visual periphery is not used to analyze objects, it is still important due to the possibility to discover visual events that can attract the gaze.

2.1.2 Simulator sickness

The horizontal FOV of human vision is approximately 170 degrees. This vision represents around 80 percent of all daily impressions from the senses [7]. When a large part of the area in our visual field is moving while the body is stationary, the illusion of self-motion, or vection, can appear [9]. The mind thinks that the body is moving while it is not. This is one of many causes of simulator sickness [8]. This means that inconsistency between bodily and visual movement, such as time lag in a HMD, can cause simulator sickness. In order to not notice time lag in a HMD, the latency should not exceed 15 ms, preferably it should be as low as 7 ms [10]. Examples of other possible causes of simulator sickness are flickering, age, and unusual maneuvers such as "flying" backwards [11]. The symptoms vary from person

to person but some examples are headache, eyestrain, nausea, blurred vision, and general discomfort. Simulator sickness is not to be confused with motion sickness which can be caused by bodily movements alone.

2.2 Simulator sickness questionnaire

An evaluation of the degree a device induces simulator sickness can be configured in many ways. The article Simulator Sickness Questionnaire: An Enhanced Method for Quantifying Simulator Sickness [12] describes the development of a simulator sickness questionnaire (SSQ) based the previous Pensacola Motion Sickness Questionnaire (MSQ). Symptoms that did not change in frequency or severity between pre- and post-exposure during simulator sickness tests were removed from the original MSQ. The resulting SSQ symptoms can be seen listed in Table 2.1. However, even if these symptoms can be evaluated through a questionnaire, they may not give a fair estimation of the provocation degree to simulator sickness since several symptoms can be effects of the same cause. Kennedy et al. performs principal-factor analyzes, followed by a normalized verimax rotation with the result shown in Table 2.1. This resulted in three distinct symptom clusters: Oculomotor (eyestrain, difficulty focusing, blurred vision, headache), Disorientation (dizziness, vertigo) and Nausea (nausea, stomach awareness, increased salivation, burping). These three symptoms are weighted to be equally represented in the questionnaire. The weights in the table are discretized to either 1 or 0, see [12] to follow the whole procedure.

The questionnaire should consist of questions asking the test subjects to rate the sensation of each symptom with zero to three points, where a lower rate means lower effect of the symptom. Scores for each symptom cluster as well as a total score are calculated in Equations 2.1. The final result does not give an absolute value of how good a simulator device is, but should give the possibility to compare experiences in a good and reliable way, before and after exposure as well as comparing different devices.

$$
\begin{aligned}
N &= [1] \cdot 9.54 \\
O &= [2] \cdot 7.58 \\
D &= [3] \cdot 13.92 \\
Totalscore &= ([1] + [2] + [3]) \cdot 3.74
\end{aligned}
\tag{2.1}
$$

Table 2.1: Computation of SSQ Scores (Kennedy et al. [12] p. 212)

	Weight		
SSQ symptoms	Nausea (N)	Oculomotor (O)	Disorientation (D)
General discomfort	1	1	
Fatigue		1	
Headache		1	
Eyestrain		1	
Difficulty focusing		1	1
Increased salivation	1		
Sweating	1		
Nausea	1		1
Difficulty concentrating	1	1	
Fullness of head			1
Blurred vision		1	1
Dizzy (eyes open)			1
Dizzy (eyes closed)			1
Vertigo			1
Stomach awareness	1		
Burping	1		
Total	[1]	[2]	[3]

2.3 Augmented reality

Augmented reality is when computer generated information is added to the human perception of the reality [13]. This includes all the senses, but this report only address the visual aspect of AR, when computer generated images are shown on top of the reality. When it comes to visualization of computer images there exist a variety of different solutions; ordinary displays, head mounted displays (HMDs) that can track head motions [14], and small displays built into contact lenses [15] are three examples. The products could also be divided into two groups: mono- and stereoscopic, where stereoscopic is necessary for a 3D augmented experience.

Many AR products are introduced to the market in 2016, such as Microsoft HoloLens [14] which is released in the first quarter, but restricted to United States and Canada. Atheer AiR is another product with similar specifications [16], also released in the first quarter 2016. The most significant difference is the field of view. The HoloLens FOV is about 35 degrees diagonally [17], to be compared with 50 degrees for Atheer AiR. All observed AR models have a very narrow field of view which could be a result of physical limitations that comes with the implemented technology [18].

An example of a low cost AR system is the Moverio BT-200, created by Epson [19], with the price of approximately one fourth of the price for HoloLens. Unlike

HoloLens and Atheer AiR, Moveiro uses an external device for control and computation. This device is similar to a smartphone without a screen, which runs Android. However, the drawback is that the computational power is no better than a lower price ranged smartphone. It also suffers from a narrow field of view of approximately 23 degrees.

2.4 Virtual reality

A virtual reality is an artificially created experience for the senses, usually for sight and hearing [20]. The most common way to create a virtual reality is by using virtual glasses. These are devices that cover part of your field of view with screens, and by displaying imagery on the screens give you an impression of being someplace else. Developers strive to make virtual reality as immersive as possible, and creating the feeling of being someplace else is a key factor for a good experience. This factor is dependent on field of view and latency, among others. Several attempts at creating an immersive virtual reality experience have been made throughout the years, but a common problem with the head mounts has been a limited field of view; most consumer products have had a field of view of only 30-40 degrees [21]. It was not until 2012 that promises of overcoming the limited field of view were made, when a Kickstarter campaign for a product called Oculus Rift was launched.

Since then, development of VR devices has increased as large companies like Facebook, HTC, Samsung, Google, and Sony have invested in VR. The Oculus Rift is one of the most commonly known devices, which in 2014 was bought by Facebook. Competition has intensified as several end consumer devices are due to hit the market in the first quarter of 2016 [22]. The different devices have a large variation when it comes to computational power and price. The lower end of the scale uses smartphones as processors and screens, such as the Google Cardboard and Zeiss VR One. Then there is the Samsung Gear VR which also uses a smartphone for screen and processing, but has additional sensors for determining the position of the head gear, such as accelerator, gyrometer, and geomagnetic sensors.

One of the upcoming devices is the Fove VR, which sets itself apart from its competitors by having eye tracking sensors. This allows the user to interact with his environment with his eyes [23]. Another interesting set of glasses, that currently can only be ordered in the US, is the Razer OSVR (Open Source Virtual Reality) which aims at making development of content to the glasses as easy as possible by having everything be open source [24]. Then there is the HTC Vive and Oculus Rift which are, on top of using accelerometers and gyroscopes, using IR emitters on their head gears which are being tracked by external IR sensors [25, 26]. The tracking of the Vive allows the user to move around in a 4.5x4.5 m space (15x15 feet) [27], whereas Oculus is primarily designed to be used while sitting [28].

2.5 Human - technical system interface

The usage of human - technical systems can usually be described with a cyclic flow of information. One example is a cashier using a cash register. The cashier scans an item, sending information to the register which sends an auditory confirmation signal. A display also shows information to the cashier how to continue and choose payment method. The information on the display is interpreted and will be the basis of the cashier's following interaction with the system.

The system interface can be divided into two parts:

- system output - how information is transferred from the system to the user
- system input - how the user can control the system

2.5.1 System output

To design a good system interface it is not enough to look at the technical design, human abilities and limitations must also be taken into account. A high cognitive work load for example will lower the quality of perceived information from the senses [7]. This section concerns the visual feedback from the system.

Contrast sensitivity is a notation for the human ability to distinguish between light and dark areas, which is a part of the ability to discover shapes and contours of objects. This ability is degraded during bad lighting conditions as well as if the object in sight is moving relative to the observer. Another aspect to consider when designing visual interfaces is that approximately eight percent of all men and around one percent of all women has some kind of color blindness. The most common one is the difficulty to distinguish green from red. Therefore, commonly used systems such as traffic lights often use redundant information; both the position of the light as well as the color gives the same information.

Interpretation of the information given from the eyes consists of two processes, bottom-up and top-down. Bottom-up is a data driven process which is only based on what the eyes perceive, while top-down uses previous experiences. Both bottom-up and top-down processes are important when searching for objects. An eye-catching object for example will use a bottom-up process while knowledge and expectations of an object is part of the top-down process.

The possibility for a human to sense depth is based on both accommodation, physical changes in the eye to tune the focal distance, and binocular vision, the ability to see an object from two slightly different positions with our two eyes. This works efficiently for distances up to around five meters. After that the depth interpretation

is mainly based on a top-down process where the object's size is an important aspect. A VR system which does not have eye tracking abilities has to have an infinite depth of field, i.e. all objects have to be in focus. The result of this will be an impaired sense of depth within the distance of around five meters.

2.5.2 System input

The interface for controllability of the system should be designed in such a way that the user understands what to do to get the desired result. For instance, a menu system should be intuitive and standard formulations and symbols should be used as far as possible. It is also preferable to minimize the number of actions and the required effort necessary to get to the desired result. When the user interacts with the system, it should give feedback directly as confirmation. The input can occur through a number of ways. However, as the user will not be able to see the input device with the head mount on, not all ways are feasible in VR. Three solutions that work in VR which are listed below.

2.5.2.1 Game controller

Despite being designed for playing games, a controller could be used with the visualization tool. A game controller, like the Xbox controller, could be plugged in to a computer through an USB port. The advantage with a game controller is that they are designed to be comfortable to use and the many buttons would allow for a lot of interaction with the visualization tool. For instance, one button could call a function for a logging system or change between a predefined set of point of views in the virtual environment.

2.5.2.2 Leap motion

Leap motion is a small device used for hand tracking. It can be plugged into a computer through USB. Leap motion uses IR LEDs and cameras to track the hands in 3D space [29]. The device can be attached to the front of a VR HMD which allows the user to see a virtual version of his/her own hands. There are games that use the position and orientation of the hands and fingers as part of the gameplay. This could be used in order to give commands to the visualization tool. Although this interface would be difficult to make intuitive for new users.

2.6 Game engine

A game engine is a software framework which contains the basic components required to create a game. By using an existing game engine, instead of creating their own, game developers save money as well as reduce the time required to develop a game. Each component of a game engine handles different parts of a game. Some of these components are the rendering engine, audio engine, physics engine and artificial intelligence. The rendering engine transforms code into objects. Everything that has to do with sound is handled by the audio engine. The physics engine determines what laws of physics are to be applied in a game, making interaction between objects feel realistic. Artificial intelligence provides behavior to all computer controlled characters. Commonly used game engines include Cryengine, Unreal engine, Frostbite, Source and Unity.

2.6.1 GPU versus CPU programming

A GPU (graphical processing unit) can, despite its name, handle a lot more applications than just graphics. The biggest difference between a GPU and a CPU (central processing unit) is the architecture. The CPU is designed for programs with a low number of threads, programs with their instructions in more or less one sequence. This means that there are only a small numbers of cores, around two to eight, each with relatively high processing power. A GPU instead has a very large number of cores, could be thousands [30], but each with lower processing power. This is well suited for parallel programming, programs with many threads. Graphical applications often have many opportunities for parallel programming when performing vector and matrix calculations and therefore take great benefit out of a GPU. Game engines, for example, use so called shaders which are programs to calculate color and light intensities. These shaders are written in specific shader languages and are often executed on the GPU.

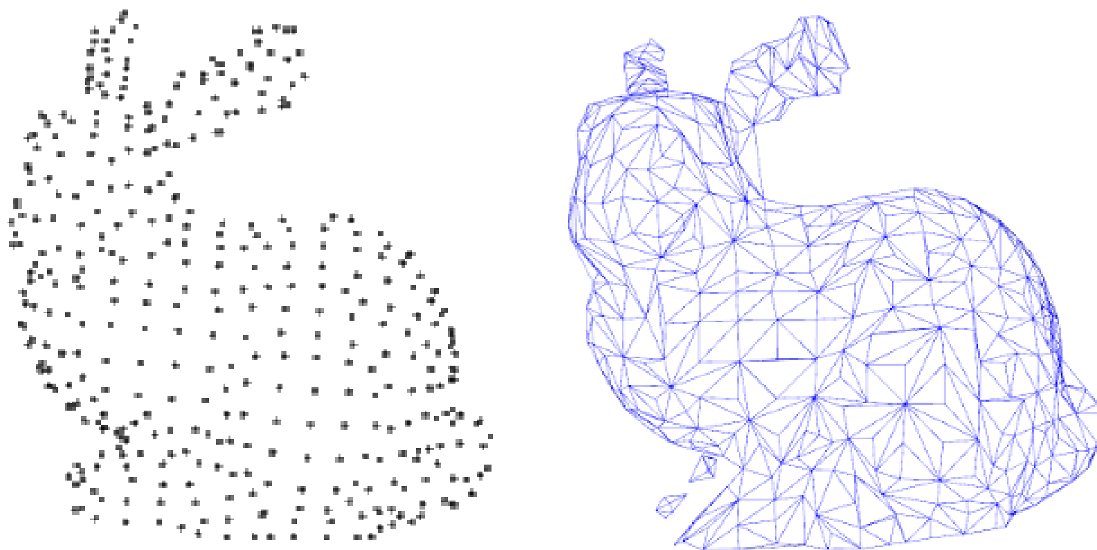
2.7 Point clouds

A point cloud is a set of points in a coordinate system, commonly the 3-dimensional XYZ-coordinate system. A point cloud is used to create a map, or model, of certain objects or an environment. It is a common type of output from laser scanners such as a Lidar [31]. The file format of a point cloud varies from sensor to sensor, some formats only contain XYZ values but there are formats that contain color for each point as well, XYZRGB [32]. An example of a XYZ-point cloud can be seen in Figure 2.3a.

2.7.1 Point Cloud Library

Point Cloud Library (PCL) is an open source library written in C++ that contains algorithms for processing point clouds [33]. The library is split into sub-libraries which allow users to compile only algorithms of interest. The algorithms include pre-processing, such as filtering and resampling, and segmentation, which is the procedure of dividing the cloud into natural and intuitive parts. Each part, or segment, can then be analyzed separately. Another algorithm is used to reconstruct the surfaces from which the point cloud was generated into a mesh.

A mesh is a combination of vertices, edges, and faces. Vertices are vectors defining the position of points, edges are lines drawn between the vertices, and the faces are surfaces enclosed by the edges. The vertices, edges, and faces are then an approximation of the original surface of an object. An example of a mesh with invisible surfaces can be seen in Figure 2.3b. Furthermore, in case of noisy measurements, unwanted points can be removed using PCL through filtering or be resampled to better resemble a scanned object.



(a) An XYZ-point cloud in the shape of a bunny **(b)** A mesh in the shape of a bunny

Figure 2.3: Example of a point cloud and a mesh

2.8 Virtual augmented reality using cameras

By mounting cameras onto a head mounted VR display it is possible to mimic the properties of AR glasses. Two cameras can capture the environment of the user which then is displayed live as stereoscopic images. If the cameras are placed and

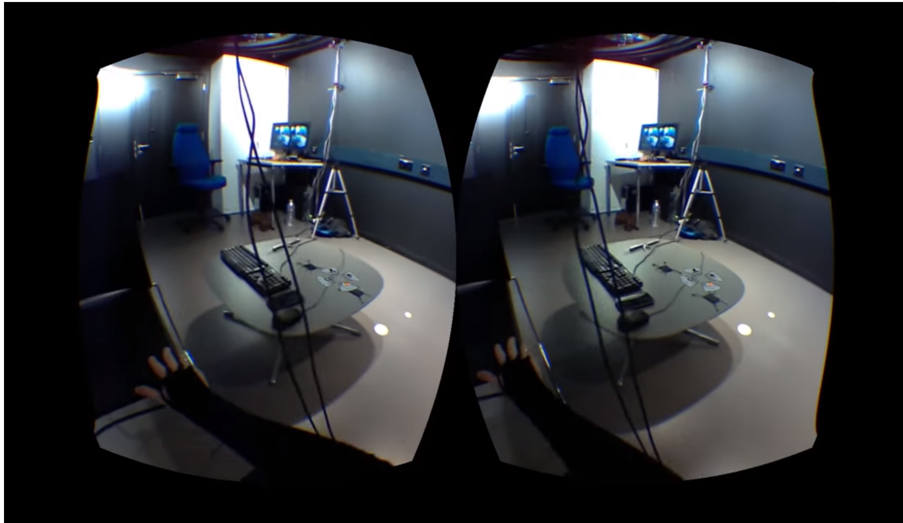


Figure 2.4: Screen capture from William Steptoe's project: AR-RIFT [34]

tuned to match the FOV and the distance between the center points of the human eyes, the interpupillary distance (IPD), the display could almost be perceived as see through. At least if it were not for the limitations in the optics of cameras and the resolution of the HMD. Figure 2.4 shows a screen capture of how it can look like. The main advantages with this solution compared to real see through AR glasses are flexibility and wider FOV. The flexibility comes from the ability to choose between a "see through" display and a completely virtual environment. It is also possible, depending on the choice of camera, to include the ability to zoom, apply color adjustments, or to have night vision.

For this project, the "see through" functionality is useful for on-line verification when the user is seated in the car. The environment would then be perceived with a high level of detail, colored objects and with a sense of depth. Using this technique, together with a live streamed point cloud and an input device, it could be possible to switch from camera view from inside the car, to a 3D view based on the point cloud stream. This would give the possibility to move the point of view around in the point cloud stream, and for example see the surroundings from above.

2.8.1 Camera placement

When mounting the cameras on the HMD, there are two ways to angle them that are worth considering: toed-in and parallel [35]. The way of angling the cameras will affect the distance to the stereo window, which is where the FOV of the two cameras are perfectly aligned, see Figure 2.5 and 2.6. When viewing stereoscopic content on an ordinary screen where the left and right frame are located at the same position, objects that are located in the stereo window will appear to be at the same distance as the screen itself. To be able to make objects appear behind the screen as well, the cameras are often tilted a few degrees toward each other which will move

the stereo window closer.

With a parallel configuration the stereo window will be at infinite distance and an object will never be exactly in the same position in the two frames at the same time. However, there is a way of moving the stereo window closer in parallel configuration as well, namely by applying a lens shift, a sideways translation of the lens relative to the sensor. This is actually preferable to a toed-in configuration since unwanted optical effects like keystoning will then not take place. A lens shift will move the used area of the lens towards the periphery. This means that the lens has to be wide enough to avoid vignetting. A similar effect can be achieved by simply crop out and keep the left part of the left frame and the right part of the right frame from the camera, with the drawback of lost pixels and decreased FOV in the camera images.

However, stereoscopic images viewed in a HMD will be experienced differently. Unlike what William Steptoe writes in his article [35], that parallel configurations should be avoided, they are instead preferable. This since the display of a HMD is divided into separate parts or even two separate units, one for the left eye and one for the right. The eyes will point straight forward when looking at objects that are at infinite distance and a parallel configuration is used together with a correctly calibrated HMD. The implication of a correctly calibrated HMD in this case is that the distance between the centers of the screens/parts of screens in the HMD is matched to the user's IPD. For a good user experience it is also important to match the distance between the cameras with the IPD which will affect the perceived depth and distance to objects. With larger distance, objects will appear to be closer than they really are.

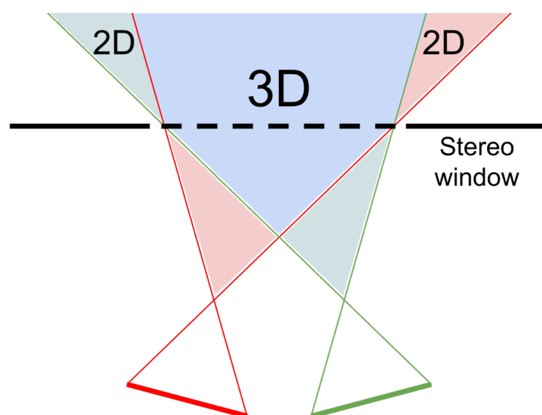


Figure 2.5: Toed-in camera configuration

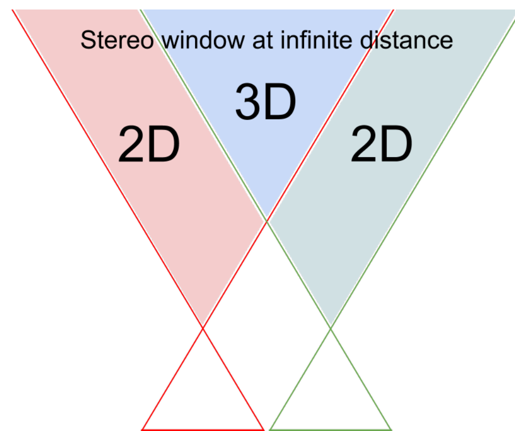


Figure 2.6: Parallel camera configuration

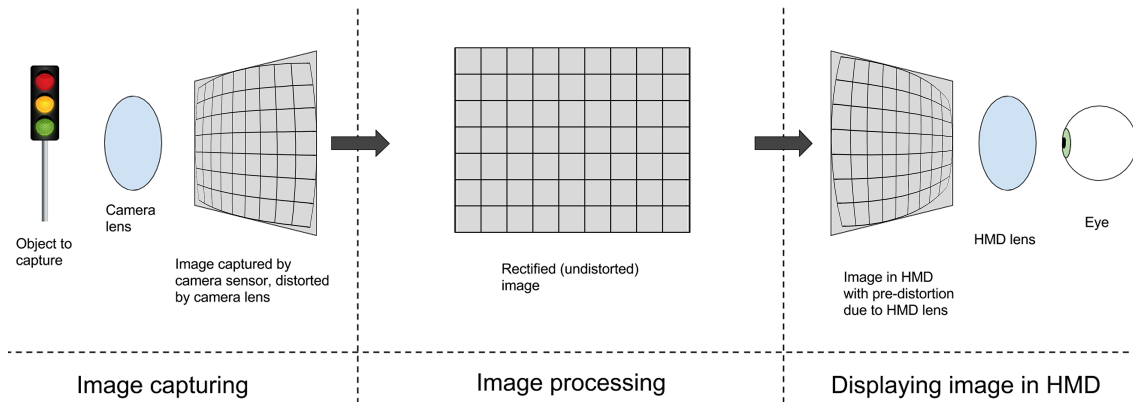


Figure 2.7: The process of capturing and displaying an image in a head mount display

2.8.2 Image distortion

To be able to focus on a screen as close to the eyes as in a HMD, lenses are built in. These lenses will distort the images displayed on the HMD's screen, and to cancel out this effect it is necessary to do an inverted pre-distortion of the images before sending them to the display. Before this is done and the images are imported to the game engine, the camera frames have to be rectified, or undistorted, from the raw data. In what extent this is necessary is dependent of the choice of camera. However, wide angle lenses tend to yield more distorted images. A camera calibration tool for rectifying is included in the open source computer vision library, OpenCV [36]. The process of capturing an image and displaying in a HMD is illustrated in Figure 2.7.

3

Shared aspects for off-line and on-line verification

The overall method for verification is that the user of the tool is seeing a clear view of the surrounding environment from which it is possible to determine which objects are relevant for the car to detect. Overlaid with this reference is the information about which objects are detected by the car's built in sensors. This chapter includes information about available sensors and how sensor data can be visualized in order to take advantage of the properties of VR and AR for the off-line and on-line mode.

3.1 Visualization of data and reference

The visualization shall consist of both data about detected objects, e.g. position, velocity and object type, as well as a reference, to make the verification possible. It is not obvious though how the reference should be displayed, neither what object data is interesting to view. The surrounding environment of the car could during on-line verification be visible through AR glasses or, as discussed in Section 2.8, through VR-glasses with mounted cameras. For off-line verification, the reference has to be pre-recorded. To structure the whole case, the discussion about different types of visualization has been divided into two sections, off-line and on-line. The most important question in the discussion is how to gain as much as possible from the 3D features of a HMD instead of using an ordinary flat 2D screen.

Since the invention of the car, humans have been in charge of driving the cars and even though accidents do occur, we have been found to be quite good "controllers". We can for example detect pedestrians that are largely hidden and still be able to classify them as humans. From this classification we can determine the probability that the pedestrian will move and cross the car's path. With this reasoning it is clear that a visualization of a reference environment in the HMD with similar properties to human perception of the real world would be a suitable solution.

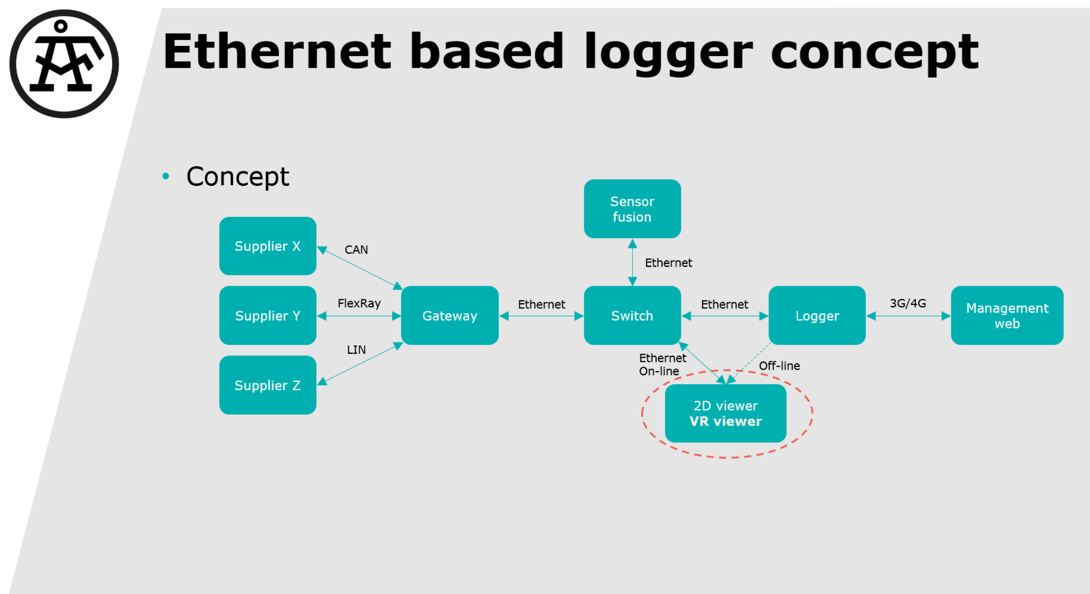


Figure 3.1: Overview of data flow

3.2 Sensor configuration

The cars in Volvo's project Drive me are using a lot of different sensor types and placements [4] to be able to perceive their surroundings. There are also reference sensors attached to test cars during the development phase, of which data can be accessed for verification of the built in consumer sensors. These consist of a 360° Lidar on the roof, a camera pointing forwards and in some cases also a 360° camera.

Table 3.1 shows information about the current sensor configuration for Volvo cars in the project Drive me. The sensor readings are processed and used with sensor fusion to estimate for instance position, velocity and size of objects. The data is then accessible through a CAN-bus in the car. The CAN-bus (Controller Area Network) is a communication network in the car.

An overview of the systems and data flow is illustrated in Figure 3.1 where "VR viewer" symbolizes this project. Data from the different sensors, the suppliers, are gathered and saved as PCAP files (Packet CAPture) locally in the Logger unit. PCAP is a file format for capturing network traffic. In on-line mode, the data can be accessed directly through Ethernet.

The cars will also have access to a global high definition static map containing information about the cars' surroundings. The map has a very high precision with an accuracy of around a couple of centimeters. The cars will then use GPS, odometry sensors, and the local map created by the other sensors in Table 3.1 to be able to position themselves in the global map in real time.

Table 3.1: Sensor configuration for cars in Volvo's Drive me project [4]

Sensor type	#	Placement	Description
Combined radar and camera	1	In the windscreen	Reads traffic signs and road curvature. Can detect objects on the road
Surround radars	4	Behind the front and rear bumpers (one in each corner of the car)	Monitoring the 360° surroundings and locates objects
360° surround camera vision	4	Two under the outer rear-view mirrors, one in the rear bumper and one in the grille	Detecting objects at close range and monitor lane markings
Multiple beam laser scanner	1	In the front of the vehicle, below the air intake	High angle resolution, 150 m range, 140° field of view
Trifocal camera	1	Behind the upper part of the windscreen	Three cameras in one with the field of views 140°, 45°, and 34° long range
Long-range radars	2	In the rear bumper	Detects fast-moving objects from far behind which is suitable when changing lane
Ultrasonic sensors	12	All around the car	Detecting objects close to the car

3.3 Sensor data for off-line verification

Only pre-recorded data is available during off-line verification, there is therefore no point in using neither AR nor VR with cameras, instead the environment has to be shown completely virtually.

3.3.1 Reference

Below is a list of feasible solutions regarding reference visualization. All are possible to visualize in a VR head set.

- **Point cloud** - Possible to apply a color overlay from e.g a 360° camera.

- **Polygon mesh derived from point cloud** - Possible to apply a color overlay from e.g a 360° camera.

- **360° video stream** - Would not contain any depth information.

Desirable is, as previously mentioned, to get a view as similar as possible to the human perception of the real world. An optimal case in that sense would therefore be to have a 360° stereoscopic camera feed. However, this is not available. A mesh with color overlay is a good alternative, this means that a 360° camera feed is projected onto the surface from the scanner location. Important to have in mind are the consequences of an offset in location between the scanner and the camera. The closer the objects are the more inaccuracy will be present in the overlay between geometric shape and its color. It is naturally the same case when color overlaying a point cloud as well. Also, meshes have to be derived from the point cloud before visualization. This requires grouping neighboring points three and three, which is relatively computational intense. Another solution is to view bare point clouds. The points in a point cloud visualization are usually very simple graphically compared to geometric shapes, just a single point. This is to minimize the computational requirements. If the points are sparsely located it can be difficult for the viewer to cognitively merge points into groups to see shapes and objects. These points, unlike more complex 3D objects such as cubes, spheres or a mesh, have no ability to cast shadows. As explained in Section 2.5.1, shadows are one of the inputs for a top-down process to be able to determine depth. This method however is the least computationally demanding since the output from the Lidar is in the same format which makes this method a good starting point for this project.

The big difference between a 360° camera feed and using a point cloud stream is that a point cloud is a 3D model which enables the possibility to change the point of view. During playback it could therefore be possible to change between the driver's perspective, view from above, and to simply walk around in the virtual environment. Worth to mention is that the Lidar's point cloud will cast shadows behind each object. Objects will thus only be visible on the sides facing the Lidar.

The global map used in the Drive me project could potentially also be used in the visualization of the reference. However, since it is static it will not include updated information about movable objects, but by merging the dynamic point cloud from the car's Lidar with the static global map it is possible to eliminate shadows behind the movable objects and possibly get extra detailed information about the static environment.

3.3.2 Object data

All sensors with associated algorithms producing data in form of detected objects will reasonably have the same visualization interface. It is therefore only necessary

to implement visualization for one such sensor for a proper evaluation of the tool. For verification purposes it would be interesting to also visualize the specification for each implemented sensor. For example, the individual field of view for each sensor or the distance within which it is accurate.

3.4 Sensor data for on-line verification

During on-line verification it is possible to make use of see through displays which would mean that the reference would be obtained by the user's own eye which, as mentioned in section 3.1, is a desirable solution. The primary option is therefore AR glasses. Taking versatility into account, it might be more interesting to use cameras on a VR glasses instead as discussed in Section 2.8. This will unlock some extra features, with the price of lower quality on the image which is highly dependent on the choice of cameras. VR glasses with cameras makes it possible to record exactly what the user has seen during on-line verification. It would also be possible for the user to for example pause the image feed, rewind, change visualization to a point cloud based material and replay an earlier event from a different angle. This configuration makes it possible to use the same HMD for off-line verification as well.

4

Design of off-line system

Off-line verification will only use pre-recorded data from PCAP files that contain stored Ethernet data traffic from a test car. Both the representation of the reality, the reference, and object data from the sensors of the cars will be accessible through the PCAP files. Thanks to the data being pre-recorded, there will be no need for see-through AR glasses. Although AR glasses are still able to produce a virtual reality, they have a much more restricted field of view and will for that reason not be considered in the design of the off-line system.

After choosing a pair of VR-glasses a game engine will be chosen and used to speed up the development of the visualization tool. Ways of processing point clouds will be added by including PCL in the game engine.

4.1 Choosing VR glasses

Important aspects to consider when choosing VR glasses are the field of view, performance, software support, and availability. Considering this, the two main candidates are the Oculus Rift Development Kit 2 and the HTC Vive Development Kit. Since a specification for the HTC Vive Development Kit is not available, the specification for the HTC Vive and Oculus Rift (consumer edition) are listed instead for comparison. Specifications for the glasses can be found in Table 4.1.

Although the Vive has the extra feature of a frontward facing camera, the Oculus will be used in this project. This is because more tutorials and guidance can be found for the Oculus since the development kits has been on the market for a longer period of time.

Table 4.1: Specifications for Oculus Rift (consumer edition), Oculus Rift Development Kit 2 (DK2) and HTC Vive [25, 28, 37, 38, 39]

	Oculus Rift	**Oculus Rift DK 2**	**HTC Vive**
Display	OLED	OLED	OLED
Resolution	2160x1200	1920x1080	2160x1200
Field of view [degrees]	110	100	110
Refresh rate [Hz]	90	60	90
Connections	1xHDMI 1.3, 3xUSB 2.0	1xHDMI 1.3, 2xUSB 2.0	1xHDMI 1.3, 2xUSB 3.0
Game engine support	Unity, Unreal Engine	Unity, Unreal Engine	Unity, Unreal Engine

4.2 Choosing game engine

As can be seen in Table 4.1, the Oculus Rift has official support for two game engines, Unity and Unreal Engine [38]. Therefore only these two engines will be considered.

Unreal Engine is a game engine developed by Epic Games and was first showcased in 1998 [40]. The latest version, Unreal Engine 4, was unveiled in 2012 and is considered one of the top engines today. One of its features, Blueprint, allows for game logic to be developed graphically which makes it easier for people unfamiliar with programming to develop games. Unreal Engine is associated with AAA-games (high quality and critically successful games). The engine is publicly available from the Unreal web page and uses the programming language C++.

Unity is a game engine developed by Unity Technologies. One of its main features is its wide platform support, it allows for games to be developed to over 15 platforms [41]. Unity is known for its relatively user friendly interface as well as its wide online support with video tutorials and guides [42]. Unity uses the programming languages C# or JavaScript.

Both engines would be usable for the verification tool. The superior graphics of Unreal Engine, along with the graphical game logic development tool will not be a requirement for the tool, although these properties are preferable. The decisive factor was the opportunity to use work done by others in the area of the project. In Unity there exists open source code for visualization of point clouds, something that did not exist for Unreal. Therefore Unity will be used as development platform.

4.3 Implementation of graphics and functionality

The implementation section will cover how the reference came to be shown, as well as what features and functions were added. An explanation of the structure of the code will also be given.

4.3.1 Reference: Point clouds

The output from the Lidar is a stream of point clouds which is saved in PCAP files. A natural start is then, as mentioned in Section 3.3.1, to visualize this stream in Unity. The files are divided so that each file consists of approximately one minute of recorded data.

The point cloud from the Lidar has, after a conversion from spherical to Cartesian coordinate system, data consisting of x, y and z coordinates and a light intensity value for each point. The intensity values are then mapped to different colors. This makes it possible to, for example, distinguish lane markers in the visualization due to the difference in reflectivity between the markers and the surrounding asphalt.

4.3.1.1 Dynamic Link Library

The point cloud library, PCL, has built in functions to grab point clouds from PCAP files. To be able to use PCL, which is written in C++, together with Unity, all PCL related code was grouped and compiled into a DLL, Dynamic Link Library. The DLL file includes the necessary functions which can be called directly from Unity.

4.3.2 Reference: Mesh

As a complement to the point cloud reference, it is possible to show a mesh, generated from a down sampled version of the point cloud stream. The point cloud and mesh views can be activated/deactivated individually in the tool. The mesh generation uses functions from PCL. It searches for neighbors to each point and groups the points in groups by three, Unity then renders a filled triangle spanned by the points of each group. There are several important aspects to consider when implementing the mesh generation to get a desired output and the capability to display it in Unity.

Figure 4.1: Left: Each triangle has a copy of the vertices
Right: Vertices are reused between triangles

4.3.2.1 Normals

The meshes in Unity include a normal vector for each vertex in the mesh. These normals are used by the shader to calculate the desired lighting of the surface. With standard shaders it is usually only the front facing side of each triangle that gets rendered. Triangles with a normal pointing directly toward the light source will be fully lit. The greater the angle between the light source and the normal, the less light received. If the normal is perpendicular, or pointing away from the light source, the triangle receives no light at all.

Unity includes a function for calculating normals based on the vertices and the defined triangles. If a vertex is used in several triangles, i.e. several triangles share a point, Unity will render a soft transition of the lighting between the triangles after normal calculations. Sharp edges and individually calculated lighting for each triangle, which is desired for example when rendering a cube, are obtained by copying the vertices so that each triangle has its own vertices and its own normals. Figure 4.1 shows the resulting difference. For a good looking representation of the reality by meshes in this project, the soft transitions are used.

PCL has a parameter in the mesh calculation to get consistency of the vertex ordering. This means that each triangle has its vertices ordered in the same direction, clockwise or counterclockwise and Unity's normal calculation is dependent on the ordering of the three vertices in a triangle. An inverse order of the vertices leads to an inverted direction of the normal vector. This in turn leads to the triangle being rendered in the wrong direction. However, the parameter to get consistent ordering of the points in PCL does not work completely as desired for all point clouds. This is probably caused by the lacking definition of clockwise/counterclockwise which is dependent of the point of observation. A custom shader for the mesh was therefore implemented. It works as the standard shader described above, with the difference that both sides of the mesh are rendered. Also, the light calculation for triangles with normal vectors angled between 90 and 180 degrees from the light source get a mirrored lighting compared to triangles in the 0 to 90 degree span, i.e. a 45 degree surface will be equally lit as a 135 degree surface. This means that the mesh rendering is no longer dependent on the ordering of the points in the triangles.

4.3.2.2 Indexing limitations

The indexing variable included in Unity's mesh structure is only two bytes large. This means that the number of vertices building up the mesh is limited to 65,536 (2^{16}, due to 16 bits index variable) while the raw point cloud from the Lidar has approximately 200,000 points. The point clouds are therefore, before mesh conversion, down sampled to match the limitation. Point clouds from the PCAP files are unordered, which means that there is no structure in the list of points. This makes the otherwise possible alternative to divide the mesh into several parts very difficult to realize.

4.3.2.3 Time required to generate mesh

The mesh generation is performed on the CPU and is very time consuming and cannot be performed in real time; therefore it has to be performed beforehand. The pre-generated meshes are saved as binary files which allow them to be loaded quickly.

4.3.2.4 Point placements

Another important aspect regarding the point clouds from the Lidar is the distribution of points. The design of the scanner, a vertical column of lasers which perform a scan between small increments in rotational angle, gives the cloud certain properties. Surfaces with a small angle of incidence, such as flat ground, will be represented by points formed in circles around the scanner.

The binding of the triangles can be tuned by several parameters. One is how long distance is allowed between two points. To make the mesh able to reach between the circular lines of points and make the ground solid, this parameter has to allow for a relatively great distance. This is not desired where the points have neighbors close by, and a point should not be connected to points other than its closest neighbors due to the resulting loss of detail.

4.3.3 Sensor to verify: Radar-camera

For the proving of concept and because of time constraints, only one sensor is implemented for verification. This is the combined radar and camera listed in Table 3.1, known as the radar-camera. All sensor data is available through the CAN bus in the vehicle. Data on the CAN bus is, as previously mentioned, saved during the test drive to PCAP files, through which the CAN-data can be accessed afterwards. The stored information for each detected object is among other:

- Object id
- Distance from sensor (X and Y distance)
- Velocity of object (X and Y velocity)
- Width of object
- Height of object
- Object type (Unknown, Car, Motorcycle, Pedestrian, Animal etc.)
- Time stamp

Each object is visualized in the tool as a semitransparent cuboid. To simplify the perception of information, much of the object information is visualized in the properties of the cuboid. This minimizes the need of text fields and makes a faster intake of information possible. The distance is transformed to coordinates of the position. Width and height are applied directly as size of the cuboid and the object type is coded by color. The semitransparent property makes it possible to see reference data behind and inside the cuboid.

4.3.4 Control functions

For the application to be a usable verification tool it is essential to have a set of carefully chosen functions. The implemented functions are chosen to give the user the ability to closely observe all available details in the recorded material and in a easy way bookmark interesting events. The different functions are listed in Table 4.2.

4.3.5 Program structure

The program is structured to be able to handle the functions in Table 4.2 in a clear and flexible way. Each reference type, sensor type, and the logic for on screen controls has individual C# classes. The reference and object data classes read all the data and store it to variables for quicker access when needed. Additionally, there is one class controlling the playback time. At each frame update, all classes display the data with the time stamp closest to the current playback time. This makes it possible to implement functions like play, pause, rewind, or to set an arbitrary playback time with ease. An overview of the structure can be seen in Figure 4.2.

Table 4.2: Implemented functions in the verification tool

Function name	**Description**
Position	Control the x and y coordinate in a horizontal plane of the observing virtual camera
Camera rotation	Yaw control of the observing virtual camera (in addition, Oculus Rift can control Roll, Pitch and Yaw)
Up/down	Height control of the virtual camera
Play/pause	Play and pause the playback of recorded data
Slow motion	Slow motion playback of recorded data, forward and backward
Rewind/forward	Switch playback direction
Log/unlog	Log current time instance of the playback, also saves the current pose of the virtual camera
Go to logged pose	Jump to the saved pose if this time instance is logged
Next/previous log instance	Jump in time between saved log instances

4.4 User input to visualization tool

To be able to view the reference and data from the sensors properly for verification, as well as log instances of time, the user must be able to give input to the system. Since the input device will not be visible to the user while the head mount is equipped, a keyboard is not a valid option. Also, using hands and gestures for input might be a good solution if one only uses the tool for short periods of time, however, for long periods of time this solution could be exhausting. Therefore a game controller was chosen as a user input device. A game controller is also a well-tried type of control device and is familiar for a lot of people. The buttons of a game controller are easy to locate even though they cannot be seen during use of the HMD. More specifically, the choice of game controller became a wireless Xbox controller. The functions related to each button can be seen in Figure 4.3.

A Leap motion device is also used, but only to visualize the user's hands during verification. The purpose is to make the user more comfortable in the VR environment while it also works as a source of inspiration during use, for future development.

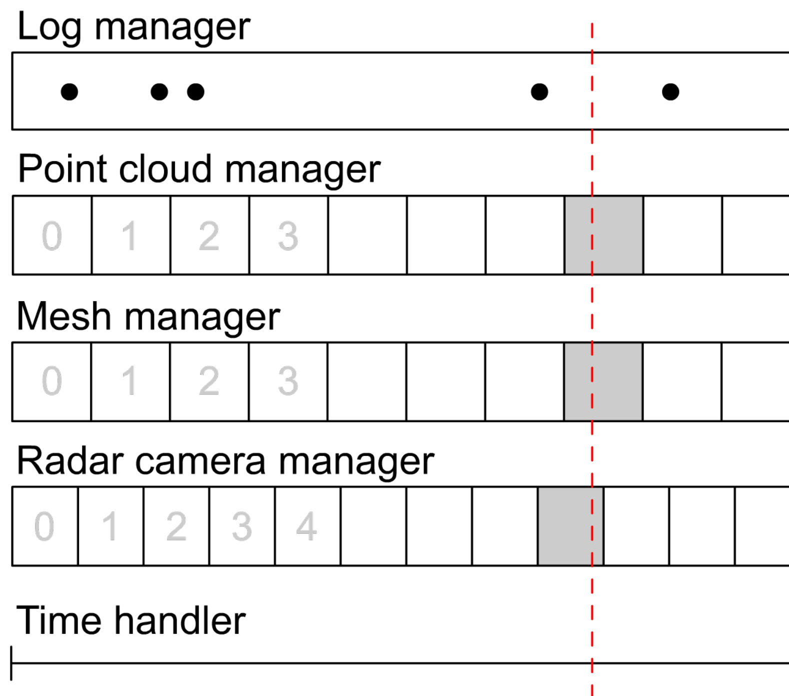
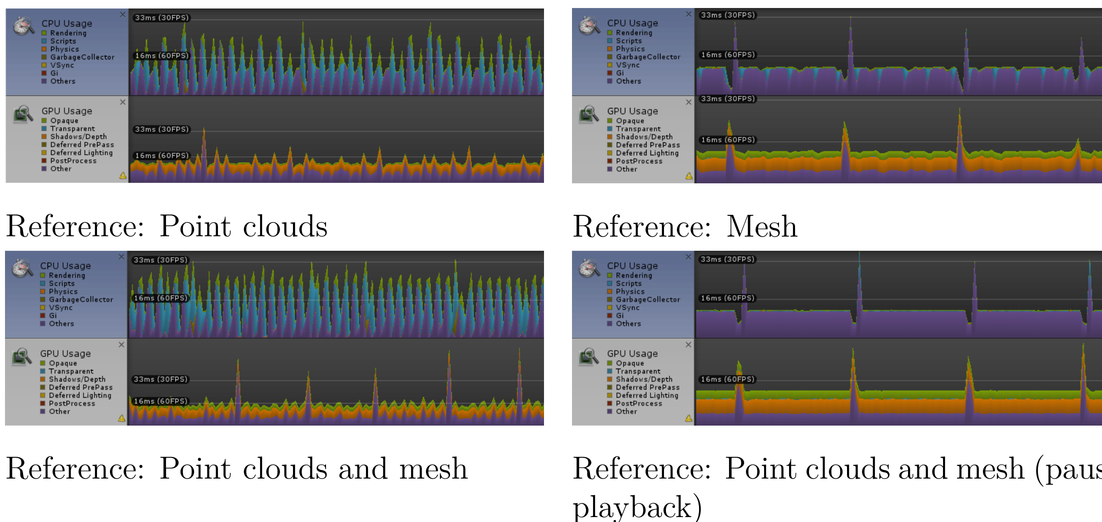


Figure 4.2: Overview of program structure. Each box represent one data instance, e.g. one point cloud or one message of detected objects. Time stamps are in the illustration saved in the middle of each box. The dashed line illustrates the current time instance and the selected grey boxes are the currently visualized data instances for that time.

4.5 Performance

Since delay between control input, such as head movements, and an insufficient frame rate are factors that can lead to simulator sickness, it is important to look into the performance. Figure 4.4 shows a profile of the CPU and GPU activity. The sparsely located spikes that are mostly visible for the mesh reference and the paused mode are caused by the game engine editor and are not present when building the application to an external executable file. Worth knowing is also the effect of using Oculus Rift. Oculus' runtime software is syncing the frame rate with the Oculus screen refresh rate which is the reason why it is quite steady around approximately 60 FPS (frames per second) for mesh and in paused mode. From this it can be concluded that frame rate, with Oculus connected, is only effected when it is under 60 FPS which occurs during reading and updating point clouds. During the frames where a new point cloud is set, the frame update time is approximately 27 ms, with some variations, which corresponds to 37 FPS. When comparing the performance of the point cloud reader and the mesh reader it is important to consider that the meshes are down sampled to approximately 65,000 vertices and saved to binary files, this compared with the point clouds which are varying with around 200,000 vertices. Recommended computer specification to use with Oculus and the specification of the computer used for the implementation and testing are shown in Table 4.3.

Figure 4.3: Image showing the functions related to each button



Reference: Point clouds

Reference: Mesh

Reference: Point clouds and mesh

Reference: Point clouds and mesh (paused playback)

Figure 4.4: Unity profiler

4.6 Testing

Evaluation of the off-line system was made through tests with 13 test subjects. However, five of these only filled out the SSQ and not any other questions seen in Appendix A.1. The test subjects were mainly engineers, and two have experience with developing similar tools. However, few have experience of using or testing verification tools. The purpose of the tests can be divided into two parts. One to evaluate how useful the tool was and the degree to which the usage in VR-environment led to simulator sickness. The second part was to take the further possibilities with the off-line part of the tool into account as well as indicate what could be useful for the implementation of the on-line part.

The tests took place in an isolated room with one test person at the time. Each test, which lasted for about 30 min, started off by the test subject filling out a SSQ form. Then the test person got to use the verification tool, as it was in its current state, and try out the different functions. Afterwards the test subject filled out a

Table 4.3: Oculus Rift recommended computer specifications vs the specifications on the PC used for implementation and testing

	Oculus Rift recommended	**PC used for implementation & testing**
Video Card	NVIDIA GTX 970 / AMD R9 290 equivalent or greater	NVIDIA Quadro 4000
CPU	Intel i5-4590 equivalent or greater	Intel Xeon E5-1620
Memory (RAM)	8 GB+	32.0 GB

SSQ form again. Various comments from the test subjects were written down as the test progressed, and afterwards the test subjects were given a question form, see Appendix A.1, to answer after exiting the room. It should be noted that the temperature of the test room was slightly higher than that of the normal comfort zone, and thus might affect the result of the SSQ.

4.6.1 Evaluation of the off-line system

The majority agreed that a stream of point clouds was a good representation of the car's environment. Furthermore, the majority also thought that the mesh, in its current state, was far too inconsistent and flickered too much. The flickering is a result of differences in the surface approximation between frames, which points were grouped and which were not. However, large objects captured close to the Lidar are visualized more clearly than by the point clouds regarding shape and depth. Smaller objects, or objects too far away lost too many details. This shows the potential of the mesh, although the generation of the meshes requires some improvement.

As for the graphics and functionality of the tool, there were a lot of small suggestions for improvement, such as a function for resetting the camera position, changing the playback speed, and making the menu marker more visible (see Appendix A.1 for full list). However these are details compared to the main objective of the project and are therefore mostly interesting only in a commercialized version of the tool.

A couple of questions were regarding the purpose of the tool, if it was useful as a verification tool and if VR was to prefer over a standard monitor. The majority thought that the tool was functional, but the answers regarding the benefit of VR were more evenly spread even though they weighed more towards that VR is beneficial because of the extra dimension which made the distinguishing of objects faster and more reliable.

4.6.2 Results of the simulator sickness questionnaire

The SSQ (simulator sickness questionnaire) is a relative measurement and was therefore performed both before and after each test with 13 participants. The result for each test was calculated according to Section 2.2. Tables 4.4 and 4.5 show the result of the questionnaire before, respectively after, tests of the tool in VR environment.

Table 4.4: SSQ result from 13 test subjects, before tests

Before	Nausea (N)	Oculumotor (O)	Disorientation (D)	**Total score**
Mean	11.7	20.4	16.1	**19.0**
Std. dev.	14.1	21.1	24.0	**19.5**

Table 4.5: SSQ result from 13 test subjects, after tests

After	Nausea (N)	Oculumotor (O)	Disorientation (D)	**Total score**
Mean	35.6	31.5	50.3	**43.0**
Std. dev.	21.0	26.4	39.6	**29.4**

The increment of simulator sickness symptoms can clearly be seen between the tests. However, the distribution is also widely spread which indicates that the effects are very individual. It can also be seen that the most common symptoms are linked to nausea and disorientation. Complete SSQ results can be seen in Appendix A.2.

4.6.3 Further development of the verification tool

The answers from the tests also include opinions regarding possible future implementations, see Appendix A.1. One suggestion was to differentiate between static and non-static objects, which could be done by implementing another suggestion, using a global static mesh and have movable objects be illustrated as point clouds. However, the three most popular ideas were color overlay on point cloud from camera feed, mounting cameras onto the Oculus Rift, and adding a video stream as part of the reference.

Due to one of the objectives of the project being to test different features in the verification tool, color overlay will not be implemented due to its similarity to a point cloud with colors from intensity. As for a camera feed from a car mounted camera compared to stereoscopic cameras on the rift, two implementations that are also similar, stereoscopic cameras would yield more visual information thanks to the added dimension of depth. Adding stereo cameras also give an indication as to if augmented reality could be beneficial for the verification tool or not. The cameras would only be usable for on-line verification.

5

Design of on-line system

As discussed in section 3.4, VR glasses with mounted cameras is a versatile solution with advantages such as allowing the same HMD to be used for both on-line and off-line verification. It would also be possible to implement a point cloud viewer for on-line equally immersive as the one for off-line. Section 4.6.3 explains why implementing stereoscopic camera feed with mounted cameras on the Rift is the preferred alternative for on-line compared with a point cloud stream.

A potential extension of the functionality in on-line verification would be to enable the possibility to switch between a stereoscopic camera feed and point cloud feed. In the point cloud view it would then be possible to change the point of view, and also pause and rewind. This functionality will however not be implemented since it would not further aid the purpose of this report.

The game engine Unity was also used for the on-line mode since this makes the merging of the two modes a straightforward process and many software components can be reused. The same controller for input will be used, however, the set of functions will be more limited in on-line mode. Since the stereoscopic cameras are attached to the Oculus, it is not possible to change the point of view with the hand controller. Also the playback time will not be controllable during on-line verification since the latest captured object data always will be used to match the stereo camera reference.

5.1 Stereoscopic cameras mounted on Oculus Rift

This section handles the hardware for the stereoscopic camera solution. The mounting and positioning of the cameras are based on the theory presented in Section 2.8.1.

5.1.1 Choice of cameras

There are several important aspects to consider in the choice of camera. The most important ones are:

- Field of view
- Resolution
- Aspect ratio
- Time lag
- Frame rate

The FOV is one of the most important aspects. The view of the camera feed in the Oculus has to have the same scale as the virtual cameras in the game engine through which the virtual environment is seen. Otherwise the position of real world objects would not match the position of virtual objects. This means that if the physical cameras have a too large FOV compared to Oculus the images would be cropped, too narrow and the images would not make use of the Oculus' entire FOV.

The display of the Oculus Rift Development Kit 2 has a resolution of 1920 x 1080 which means that the individual eye's screen area has a resolution of 960 x 1080 and thereby an aspect ratio of 1:1.125. This is the screen area that preferably should match the camera frames after image distortions have been applied. Lastly, aspects like time lag between capturing a frame and when it is accessible in the virtual world, and the frame rate of the camera feed will have impact on the user experience. These aspects will affect motion sickness as well as immersion.

There are only a few available cameras on the market with a specified frame rate higher than 30 FPS, so the choice of cameras was instead based on the field of view. The chosen cameras are of the model Genius WideCam F100 [43] with specifications shown in Table 5.1. Worth mentioning is that many web cameras are designed to make it possible to replace the optics. This makes it possible to widen the field of view from the original set-up depending on the choice of lens. Changing the optics, however, induce a risk of dirtying the exposed camera sensor, and there might be compatibility issues with the socket and potential auto focus. With the WideCam F100 no modifications are necessary.

5.1.2 Camera mounting

To match the aspect ratio of each eye's field of view, it is recommended to mount the cameras vertically. However, for this application, where sensors that mainly detect

Table 5.1: Specifications of Genius WideCam F100

	Genius WideCam F100
Resolution	Maximum 1080p Full HD
FOV	120 degrees diagonally in full HD, slightly less in lower setting
FPS	Up to 30 frames/s
Interface	USB 2.0
Aspect ratio	1:1.78 or 1:1.33 depending on chosen resolution

objects close to the ground are verified, the horizontal FOV is more important. Therefore the cameras were mounted horizontally. With the horizontal mounting the cameras' horizontal FOV almost match each eye's horizontal FOV in Oculus perfectly which then also results in a maximized field of overlapping between left and right eye, i.e. the stereo vision field.

The cameras should preferably be mounted in a way that enables the possibility to tune the angles to allow for small manufacturing differences between the cameras, as well as changing the distance between the cameras. The tilting of the cameras in up and down direction has to be almost identical in order for the human brain to be able to merge the left and right images. The IPD, the interpupillary distance, is individual and should for the best VR experience be set to match the user in the Oculus configuration utility which is a part of Oculus runtime. Most important is, however, that this setting matches the distance between the cameras. This means that the distance between the physical cameras should match the distance between the virtual cameras in Unity.

To attach the cameras to the Oculus, a 3D model was designed and 3D printed to allow for the previously mentioned camera mounting, see Figure 5.1. It is designed to replace the original cable door on the Oculus Rift Development Kit 2. The model also includes a holder for Leap motion. Figure 5.2 shows the Oculus HMD with mounted cameras as well as the Leap motion. The calibration of the cameras was done by rendering an overlaid cross sight in each camera feed. The cameras were then tuned until a far distant object right in front of the HMD (approximately 10 meters away) was located in the centre of each camera feed.

5.2 Camera feed in Unity

Using OpenCV to rectify the images as a preprocessing step is very computational demanding according to Steptoe [44]. The distortion of the camera images from

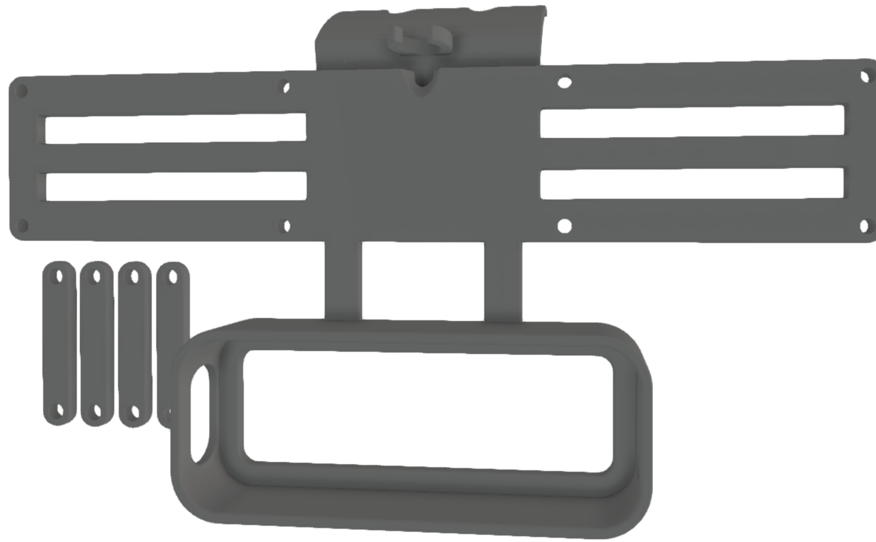


Figure 5.1: 3D model for Oculus door mounting

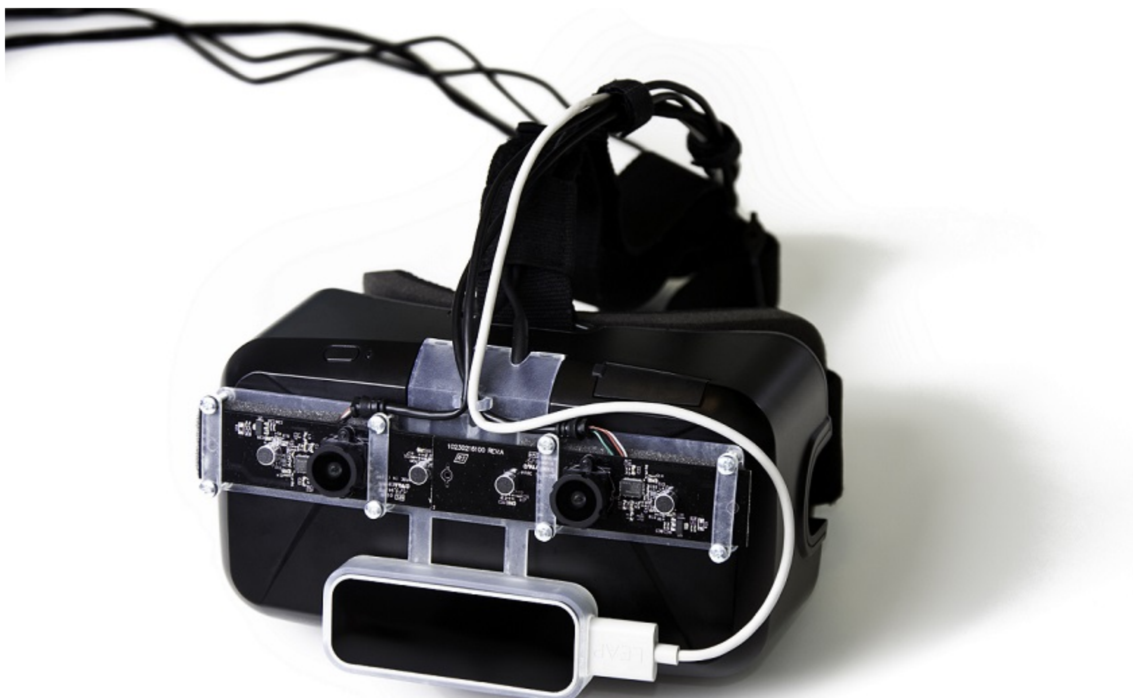


Figure 5.2: Oculus Rift with mounted cameras and a Leap Motion

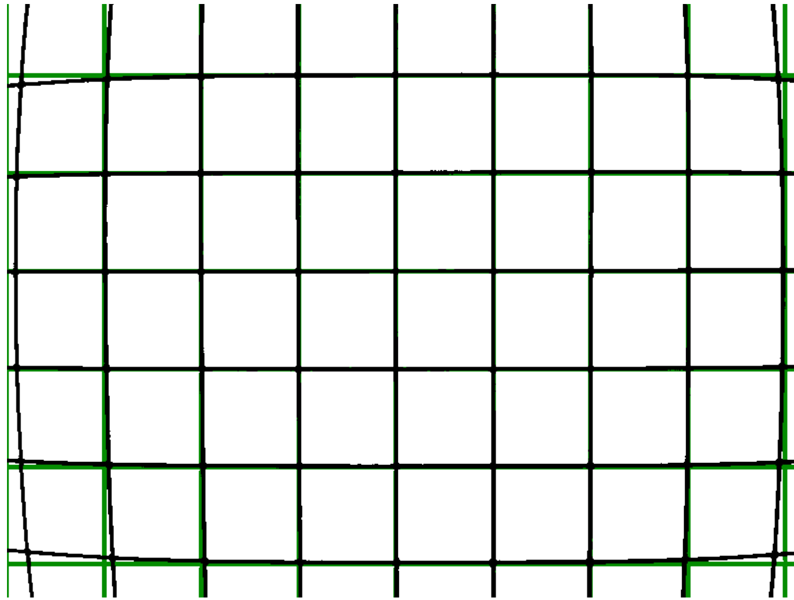


Figure 5.3: Visualized distortion of the WideCam F100 camera (black) compared with an undistorted grid (green)

the WideCam F100 is visualized in Figure 5.3. The images from the WideCam are deemed to be sufficient rectilinear, or undistorted, so that a rectifying step could be omitted.

There are several possible methods for viewing the feed from the web cameras in Oculus, requirements are however that camera feed is always positioned right in front of the viewer independent of the orientation of the users head. The left and right camera also needs to be separated so that the left eye is unable to see the right camera feed and vice versa.

5.2.1 Properties with Unity's VR compatibility

To create a suitable solution, Unity virtual camera setup for VR has to be considered. The build in VR compatibility for certain devices is a new feature since Unity 5.1. VR support can easily be enabled, Unity then duplicates the virtual camera during runtime to catch frames for both left and right eye. The virtual cameras are controlled in six degrees of freedom by the head movements recorded by the HMD's sensors. The inter camera distance, the distance between the two virtual cameras, is as mentioned controlled by the settings in Oculus configuration utility.

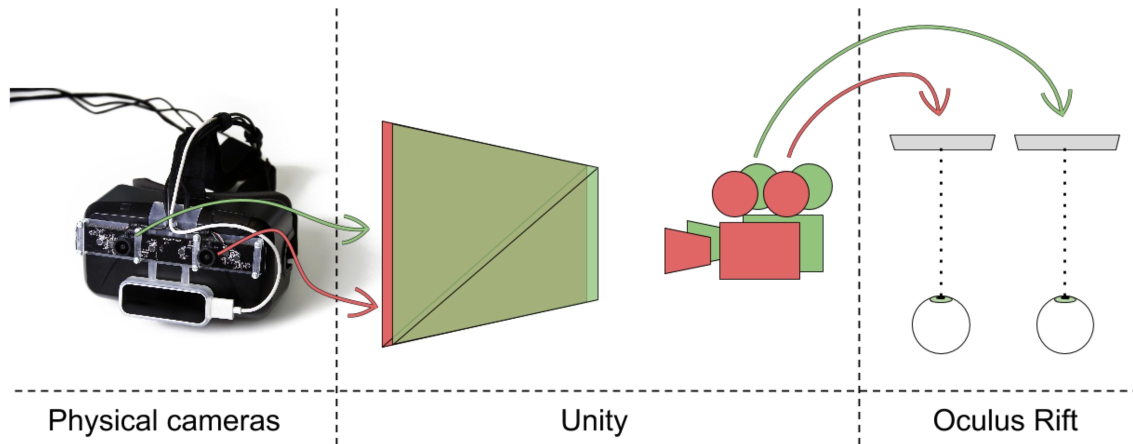


Figure 5.4: Setup of virtual cameras and quads in Unity. The imagery from the physical cameras are shown on the quads in Unity. Virtual cameras are filming the quads, and the films are shown on the screens of Oculus Rift

5.2.2 Stereoscopic visualization

The tool uses a web camera texture stream for each camera which continuously updates Unity textures with what is captured by the cameras. The textures are applied to two rectangular stripped down planes, called quads, which are set to always be located in front of the virtual cameras. An illustration of the camera setup with the quads, each consisting of four vertices and two triangles, can be seen in Figure 5.4. The textures are applied to the quad by a custom shader to always be rendered in the background so objects are visible even if they are behind the quads in the 3D environment.

By using two virtual cameras before the duplication during runtime, but only use the left camera instance of first and the right instance of the second during runtime, it is possible to use culling masks to exclude the left quad from the right camera's view and vice versa. This setup with dual cameras and quads enables easy accessible methods for changing for example the image size by just changing the quad size.

5.3 Object data and simulation of on-line verification

The object data is still limited to the output from the radar-camera sensor and are visualized in the same manner as in off-line mode. The colored cuboids are, as previously mentioned, rendered in front of the quads but are semitransparent to not conceal important information from the real world. The sensor data reader listens through a User Datagram Protocol, UDP, to the output from the network

unit seen back in Figure 3.1 during the test drive. This network traffic can also be simulated with a PCAP playback program, which sends data to the visualizer through a network socket, for test of the on-line system while off-line.

5.3.1 Additional settings

The estimated position of the user's head is controlled by the IR-tracker which comes with Oculus Rift. The pose of the tracker in Unity's 3D environment must however be correct relative to the sensor data in order to align the object markers with reality. An estimation of the passenger seat is preset but can be changed during runtime with the game controller.

5.4 Performance

The image feed time delay was measured by recording the reality and the captured camera feed in Unity momentarily with a 60 FPS video camera. Counting the frames between an event's occurrence in reality and the captured camera feed in Unity resulted in an estimated delay of 125 ms.

5.5 Testing

A test of the on-line verification mode was performed with a Volvo XC90. Unfortunately, the tool was only evaluated by the authors of this report since there were no experienced test subjects available. The test started with a manual calibration of the position, as described in Section 5.3.1. The test showed the possibilities with virtual augmented reality and that the on-line verification worked. The main drawback is currently the quality, referring to the resolution and optical quality, and performance of the cameras such as the image latency and refresh rate.

A problem that has to be solved in order to use the tool properly during test drives is the effect of the vehicle's acceleration. The built in accelerometer in Oculus Rift, which is used to estimate the orientation of the user's head, is affected by the accelerations of the vehicle. Usually the resulting acceleration vector on Oculus can be approximated to be the gravitational vector. However, during for example forward acceleration, the resulting vector will be pointing slightly backwards which Oculus interprets incorrectly as the head being tilted forward. This results in the radar-camera objects not aligning with the stereo camera feed. On-line verification with stereo mounted cameras has potential but is not ready with the currently implemented hardware and software.

6

Results

In this chapter images of the tool are shown. This is followed by a comparison between the objectives set in the beginning of the project with the actual properties and performance of the tool.

6.1 The resulting tool

Images from the resulting tool can be seen in Figures 6.1 to 6.4. Note that the referenced figures appear brighter in Oculus Rift compared to an ordinary monitor. Also note that the boxes are manually generated and are not drawn based on the data from the radar-camera due to confidentiality. Due to the confidentiality of the radar-camera there are not any figures from the on-line verification.

6.2 Comparing objectives with results

In this section the results are matched to the objectives of the project. The objectives are repeated below to simplify the comparison between objectives and result.

6.2.1 Primary objectives

The tool shall in a clear manner visualize the close surrounding of the car in stereoscopic 3D. At each point in time in the data input stream the user of the tool shall be able to distinguish all, for the car, relevant objects within the user's near peripheral field. [3].

In off-line mode:
By using a stream of point clouds or meshes if desired, the user is able to distinguish all, for the car, relevant objects within the near peripheral field.

In on-line mode:
The user can see the environment captured by a stereoscopic camera set, which has approximately the same horizontal FOV as Oculus Rift. The on-line test showed that it was easier to distinguish and classify objects using stereoscopic cameras compared to a point cloud in the off-line mode.

The tool shall mark detected objects in the 3D space so that a user easily can see which objects that are detected and which are not. The user shall be able to see all markings if they are within the users near peripheral field.

In off-line and on-line mode:
The objects that are detected by the sensor are marked with a box to inform the user that the car is aware of the objects. The box is clearly visible as long as it is within the user's near peripheral vision. However, since the field of view and the range of the sensor to verify is not being shown in the tool, it is not possible to determine which objects that the car should have detected.

The tool shall perform the visualization in real-time and with a sufficiently short time delay as to not induce simulator sickness.

In off-line mode:
The tool does to some extent, more if careless maneuvers are performed, induce motion sickness. However, the effect is highly individual and might be reduced with extended use of the tool.

In on-line mode:
The delay of the cameras is 125 ms which does result in simulator sickness being induced.

6.2.2 Secondary objectives

If several sensors are used to detect an object, it shall be clear by which sensors an object is detected. This means that a user with limited experience with the tool shall, by looking at a detected object, be able to tell by which sensors the object was detected.

In off-line and on-line mode:
Only one sensor, the radar-camera, has been used for verification. It was therefore not necessary to implement a way of seeing by which sensor an object was detected.

The interface shall include a method for logging misdetections. Example of possible log information is a time stamp together with angular coordinates where the non-detected object is located.

In off-line mode:
With a resolution of 10 milliseconds, it is possible to log the current time instance during playback. When logged, the current pose is logged as well to make it easier to go back to the pose where the misdetection occurred.

In on-line mode:
The on-line mode does not currently contain any functions to log misdetections.

Figure 6.1: Menu screen

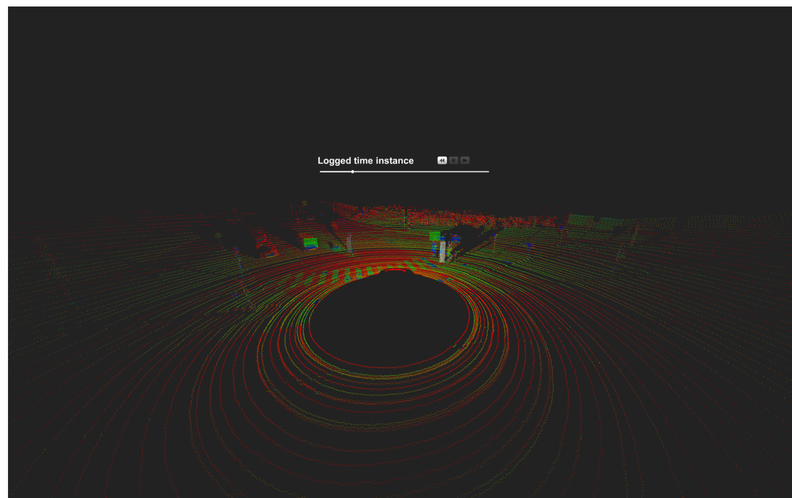


Figure 6.2: Off-line: Point cloud. Note that the boxes are manually generated and are not drawn based on the data from the radar-camera due to confidentiality

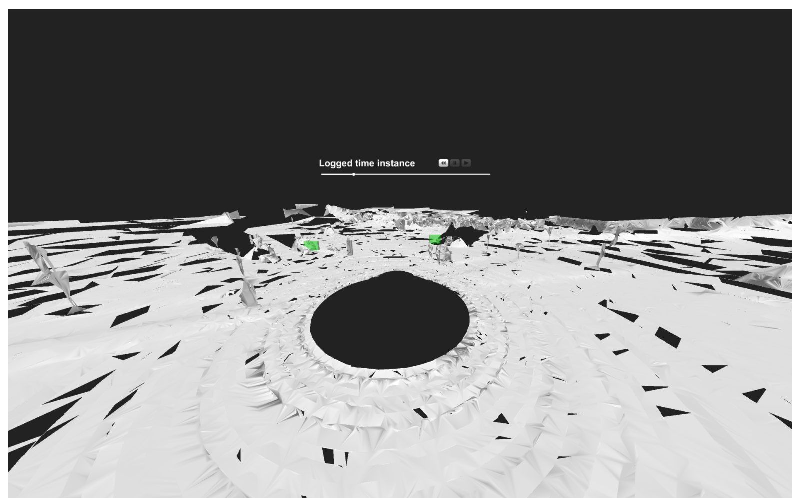


Figure 6.3: Off-line: Mesh. Note that the boxes are manually generated and are not drawn based on the data from the radar-camera due to confidentiality

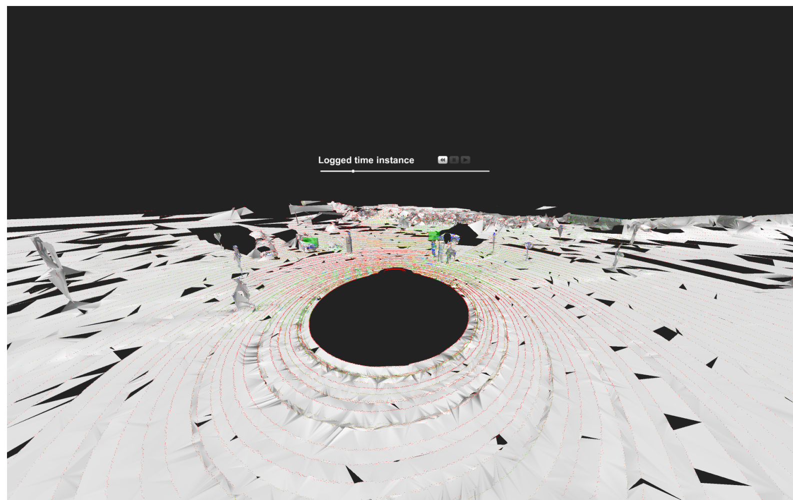


Figure 6.4: Off-line: Point cloud and
mesh. Note that the boxes are manually generated and are not drawn based on the data from the radar-camera due to confidentiality

7
Discussion

During the project a lot of decisions that affect the result have been made. In this chapter a selection of these decisions are discussed along with other matters of interest.

7.1 Usability of the tool for verification

For verification, the specification of the sensor, currently the radar-camera sensor, must be clear for the user of the tool. That could be to visualize the field of view and range in which the sensor should be able to detect objects and also what kind of objects it should detect.

7.1.1 Selection of test subjects for off-line verification

The test subjects were selected from coworkers in the office, this means that few had real experience with similar types of tools. To get a complete picture of how useful the tool is regarding functionality and efficiency, a more experienced set of test subjects should be selected.

7.1.2 Selection of test subjects for on-line verification

The test of the on-line system was only performed by the authors of this report. For a proper evaluation of the functionality as well as the simulator sickness, a larger set of objective test subjects should be used. This was unfortunately not possible to realize within the time frame of the project. The latency of the camera feed (125 ms) and the induced simulator sickness was however so significant that one could argue that further testing is unnecessary.

7.1.3 Performance during on-line test drive

One issue during the on-line test was the Oculus Rifts' estimated self orientation. Oculus Rift assumes that gravity is the only long term force that is being applied to the head mount. So during accelerations in the test drive, Oculus Rift misinterpreted these forces as the user tilting his head backwards and forwards respectively. This lead to the position of the radar-camera data being misaligned with the stereoscopic camera feed. However, it may be possible neutralize this effect by reading the vehicle's velocity, steering and acceleration and compensate the Oculus orientation estimation with this.

7.2 Simulator sickness

The main issue with the tool is induced simulator sickness. However, the authors of this report were less affected by the tool than the test subjects. Though, since our opinion is not objective, this was not included in the result section of the report. The reduced simulator sickness is likely an effect of more time spent using the tool than the test subjects. This would indicate that the symptoms are reduced with an increased time spent with the tool. Another possible contribution to resistance to simulator sickness might be low age, that younger people might be more adaptive and therefore are less sensitive to virtual environments.

7.2.1 Temperature

The tests for the SSQ were done in an isolated room to make the test subject feel more comfortable. However, the temperature of this room was slightly above the normal comfort temperature (this temperature was unfortunately not measured). Several of the symptoms in the SSQ might have been effected of the high room temperature. Symptoms like sweating can have been effected directly while symptoms like nausea can have been effected indirectly through a increased sensitivity of the subjects.

7.2.2 Quality of computer

One of the parameters that affect simulator sickness is the frame rate. Oculus Rift DK2 can display up to 60 FPS. However, the tool did from time to time drop below 60 FPS during point cloud updates possibly inducing more simulator sickness in the test subjects, see Figure 4.4. It is hard to say how much the spikes in Figure 4.4, that is the FPS drops, affected the simulator sickness, if any at all. However, it might

have been possible to increase the frame rate by implementing a small buffer. The data from the PCAP files was stored in variables for quicker access, but that data still had to be converted to point clouds before visualization. A buffer could have increased the frame rate by storing data already converted to point clouds that was ready for visualization. A different solution would be to read and visualize the point clouds in partitions. Then segments of the cloud could be read and visualized at different frames, thereby spreading the workload. The drawback with this method is the induced object tearing in objects that are part of two partitions. The tool can of course also be run on a computer with a faster CPU.

7.2.3 Quality of Oculus Rift Development Kit 2

One of the big upgrades between Oculus Rift DK2 and Oculus Rift (consumer edition) is the frame rate that increases from 60 FPS to 90 FPS. This results in a smoother experience, possibly reducing the simulator sickness even further. Important to consider is as previously mentioned that the highest frame rate in Figure 4.4 is limited to 60 FPS due to Oculus DK2 and will be higher with the consumer edition. This has been verified by analyzing the frame rate without Oculus connected.

7.2.4 Cameras for on-line verification

The web cameras in the stereo camera setup have a maximum of 30 FPS during good lighting conditions and there is also an delay of approximately 125 ms between the capturing of an image until it is shown in Oculus. These are both properties that should be improved to minimize the simulator sickness. Fast 60 FPS cameras with USB 3.0 could be an alternative, though market supply is limited.

It could also be interesting to rectify the camera feeds on the GPU through a shader. Rectifying might decrease the induced simulator sickness but also make the user more comfortable in general when the camera images match the reality better. This shader method would have a smaller impact on performance compared to rectifying using OpenCV on the CPU.

7.3 Choice of 3D development environment

An interesting area of discussion is the choice of environment used to build the 3D world with reference and object data. Eventually the game engine Unity was chosen. The arguments were mainly the support for Oculus and the existing point cloud visualizer. It should be mentioned that first an attempt using Unreal Engine was made, since both Unreal and PCL are written in C++. In Unreal there is no

point cloud visualizer so the points were instead illustrated as tetroids (tree-sided pyramids), however, these were far too computationally demanding and caused lag when the number of tetroids exceeded 10,000 (the point clouds consist of about 200,000 points). The solution was to create a wrapper and use Unity instead.

An alternative was to use ROS (Robot Operating System) for communication and the built in ROS GUI for visualization. This option was not explored further, since using a game engine result in a more flexible application for further development. However, other game engines such as Blender Game Engine could have been investigated as well. Blender was neglected due to it not officially supporting Oculus Rift. It is likely that third party support exists for several different alternatives.

7.4 Alternative approach and project content

There are many possibilities with VR and AR and it is difficult to perform a complete evaluation by only implementing a few ideas during a limited time. AR with a real AR head set such as the Microsoft HoloLens or Atheer AiR would be very interesting to compare with VR with mounted cameras. An alternative approach to the project would therefore be to only implement an on-line solution and focus on comparing AR with a VR plus cameras. However, this was not possible due to AR glasses not being accessible.

7.5 Extended use of the verification tool

A further extension of the tool could be to also include sensor data from other types of sensors in the vehicle, for example speed and wheel directions. This data could be used to display a predicted path in the 3D environment. The tool could also be extended to include verification of situation awareness and visualize the possible paths planned by the vehicles' own logic handler, including preferable, non-preferable, and forbidden areas in front of the vehicle marked with different colors.

8

Conclusion

The tests show that the implemented off-line verification tool was operational and fulfilled its purpose. Using virtual reality could be beneficial for verification of sensors in autonomous cars with the following motive: The user of the verification tool perceives a reliable overview of the vehicle's surroundings faster than by using an ordinary monitor. However, the user experience is very individual and the tool has to be optimized in performance in order to minimize the risk of inducing simulator sickness. Functionality could be improved for increased efficiency, and it should also be more adjustable regarding speed of controller in the virtual 3D environment for the tool to be suitable for all users.

The test of the on-line mode was successful, but the precision of the markings was to poor for a real use of the tool. The cameras has to be exchanged to faster cameras, both regarding frame rate as well as latency, and with a higher resolution. When AR devices has a more sufficient FOV, that could be an interesting area to explore further. Even though AR does have potential in a verification tool, whether it is actually beneficial or not for on-line verification could not be determined in this report.

Bibliography

[1] D. Reisinger, "10 automakers, tech firms researching self-driving cars — eweek," 2015. [Online; accessed 29-Jan-2016].

[2] Google, "Google self-driving car project." [Online; accessed 29-Jan-2016].

[3] Wikipedia, "Peripheral vision — Wikipedia, The Free Encyclopedia," 2016. [Online; accessed 29-Jan-2016].

[4] Volvo car group, "Press Release Volvo Cars presents a unique solution for integrating self-driving cars into real traffic," 2015. [Online; accessed 07-Mar-2016].

[5] Volvo car group, "The toughest test for the self-driving car – real life," 2015. [Online; accessed 10-Mar-2016].

[6] Wikipedia, "Iteration — Wikipedia, The Free Encyclopedia," 2015. [Online; accessed 25-Jan-2016].

[7] "Arbete och teknik på människans villkor."

[8] Oculus, "Simulator Sickness," 2016. [Online; accessed 27-Jan-2016].

[9] K. Arthur, "Effects of field of view on task performance with head-mounted displays," tech. rep., Department of Computer Science, University of North Carolina, 1996.

[10] Michael Abrash, "Latency – the sine qua non of AR and VR," 2012. [Online; accessed 3-June-2016].

[11] E. M. Kolasinski, "Simulator sickness in virtual environments," tech. rep., U.S. Army Research Institute, May 1995. Technical Report 1027.

[12] Robert S. Kennedy, Norman E. Lane, Kevin S. Berbaum, Michael G. Lilienthal, "Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness," *The International Journal of Aviation Psychology*, 1993.

[13] Mashable, "Augmented reality." [Online; accessed 26-Jan-2016].

[14] Microsoft, "Hololens," 2016. [Online; accessed 26-Jan-2016].

[15] L. O'Neil, "Lcd contact lenses could display text messages in your eye," 2012. [Online; accessed 26-Jan-2016].

[16] Atheer, "Get air now," 2016. [Online; accessed 27-Jan-2016].

[17] ITPro, "Microsoft hololens release date, rumours, specs & pricing: Microsoft dev shows off hololens streaming halo 5 from xbox one," 2015. [Online; accessed 27-Jan-2016].

[18] Unknown, "HoloLens and Field of View in Augmented Reality," 2015. [Online; accessed 28-Jan-2016].

[19] Epson, "Specifikationer," 2016. [Online; accessed 27-Jan-2016].

[20] Wikipedia, "Virtual reality — Wikipedia, The Free Encyclopedia," 2016. [Online; accessed 28-Jan-2016].

[21] D. Lee, "Oculus Rift virtual reality headset gets Kickstarter cash," 2012. [Online; accessed 28-Jan-2016].

[22] J. Ripton and L. Prasuethsut, "The VR race: What you need to know about Oculus Rift, HTC Vive and more," 2015. [Online; accessed 28-Jan-2016].

[23] Fove, "The world's first eye tracking virtual reality headset." [Online; accessed 27-Jan-2016].

[24] OSVR, "Configure your Hacker Dev Kit," 2015. [Online; accessed 27-Jan-2016].

[25] Wikipedia, "HTC Vive — Wikipedia, The Free Encyclopedia," 2016. [Online; accessed 26-Jan-2016].

[26] Oculus, "Documentation," 2015. [Online; accessed 27-Jan-2016].

[27] A. Davies, "Hands-On With The HTC Vive," 2016. [Online; accessed 27-Jan-2016].

[28] Brandon Widder and Will Nicol, "Spec showdown: Oculus Rift vs. HTC Vive," 2016. [Online; accessed 27-Jan-2016].

[29] Alex Colgan, "How Does the Leap Motion Controller Work?," 2014. [Online; accessed 11-Feb-2016].

[30] NVIDIA, "WHAT IS GPU ACCELERATED COMPUTING?." [Online; accessed 31-May-2016].

[31] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, May 2011.

[32] PCL, "PCLVisualizer." [Online; accessed 2-Feb-2016].

[33] pcl, "Documentation." [Online; accessed 8-Feb-2016].

[34] William Steptoe, "Ar-rift (part 1)," 2013. [Online; accessed 24-May-2016].

[35] William Steptoe, "Ar-rift: Building the stereo camera (part 3)," 2013. [Online; accessed 15-Feb-2016].

[36] OpenCV dev team, "Camera calibration with OpenCV," 2014. [Online; accessed 15-Feb-2016].

[37] Rift Info, "Oculus Rift specs - DK1 vs DK2 comparison," 2016. [Online; accessed 21-Apr-2016].

[38] Oculus, "Overview," 2015. [Online; accessed 27-Jan-2016].

[39] HTC, "The best experiences," 2015. [Online; accessed 28-Jan-2016].

[40] Wikipedia, "Unreal Engine — Wikipedia, The Free Encyclopedia," 2016. [Online; accessed 2-Feb-2016].

[41] Wikipedia, "Unity (game engine) — Wikipedia, The Free Encyclopedia," 2016. [Online; accessed 8-Feb-2016].

[42] Ian Hirschfeld, "Jumping into VR: Unreal vs. Unity in One Weekend," 2015. [Online; accessed 2-Feb-2016].

[43] Genius, "Ultra wide angle Full HD webcam - WideCam F100." [Online; accessed 17-May-2016].

[44] William Steptoe, "Ar-rift: Aligning tracking and video spaces (part 5)," 2013. [Online; accessed 24-May-2016].

A

Appendix

A.1 Survey

Note: The numbers before the answers correspond to the number of test subjects that shared the opinion. There were 8 test subjects. The test subjects had little to no previous experience with sensor verification and using virtual reality devices. The ages were ranging from 20-50 years.

Graphical interface

Where you able to get a good overview of the functions of the tool? If not, why?
6 - Yes
1 - The text was too small
1 - An oral explanation was required

Was the tool intuitive? E.g., was it easy to increase/decrease altitude or log a time instance? Were the functions easy to use? If not, which function and why?
7 - Yes
1 - The start/pause - forward/backward was not intuitive

Did you find the tool esthetically pleasing? E.g., how was the colors of the point cloud and the illustration of a logged time instance? Was the marking of objects clear?
7 - Yes
3 - Add more contrast to the selection marker of menu options
3 - The marking of objects block the view, replace with a wireframe
1 - Show on the time axis where the logs are at
1 - Misspelling of Camera Control, currently it says "Camera Controll"
1 - Make the surrounding brighter

Functionality

Did you feel that the tool was lacking any functions? If so, which ones and why?
3 - Reset the camera position
2 - Change playback speed
1 - Mark in 3D space
1 - Log information in form of text
1 - Allow for the directional pad to move the menu marker in addition to the left joystick
1 - A button that removed the ground, this would make the objects appear more clearly

Did you feel that any functions were superfluous? If so, which functions and why?
3 - No
2 - The mesh
1 - Pitch rotation of right joystick
1 - I do not know

Where you able to handle/use existing functions? If not, what was unclear?
6 - Yes
3 - A way of showing what the buttons did would be good
1 - A bit trouble knowing if play/pause and/or forward/backward was pressed

User experience

Was the experience smooth? E.g., was the speed of the joysticks moderate? Was the image clear?
5 - Yes
2 - The controller was too fast

How do you think that the mesh represented reality? Were the objects clear?
6 - The mesh was unclear
4 - Objects were clear on and off
1 - It was clear close to the car
1 - Yes, but it was flickered to much

How do you think that the point cloud represented reality? Were the objects clear?
6 - The point cloud was clear
1 - It was clear, but lacked the mesh's sense of depth

1 - It was clear close to the car

Which reference, mesh, point cloud, or both, did you prefer? Why?
3 - Point cloud, easier to interpret objects
1 - Point cloud, same information as mesh with smaller visual impact
1 - Point cloud, the mesh was flickering to much
1 - Point cloud, it was clearer and less motion sickness inducing
1 - A mix
1 - One or the other, point clouds are more exact but requires practice to interpret, while meshes feel more intuitive

Other

Do you think that the tool in its current condition and with the currently implemented functions works well as a verification tool for autonomous cars?
3 - Yes
0 - No

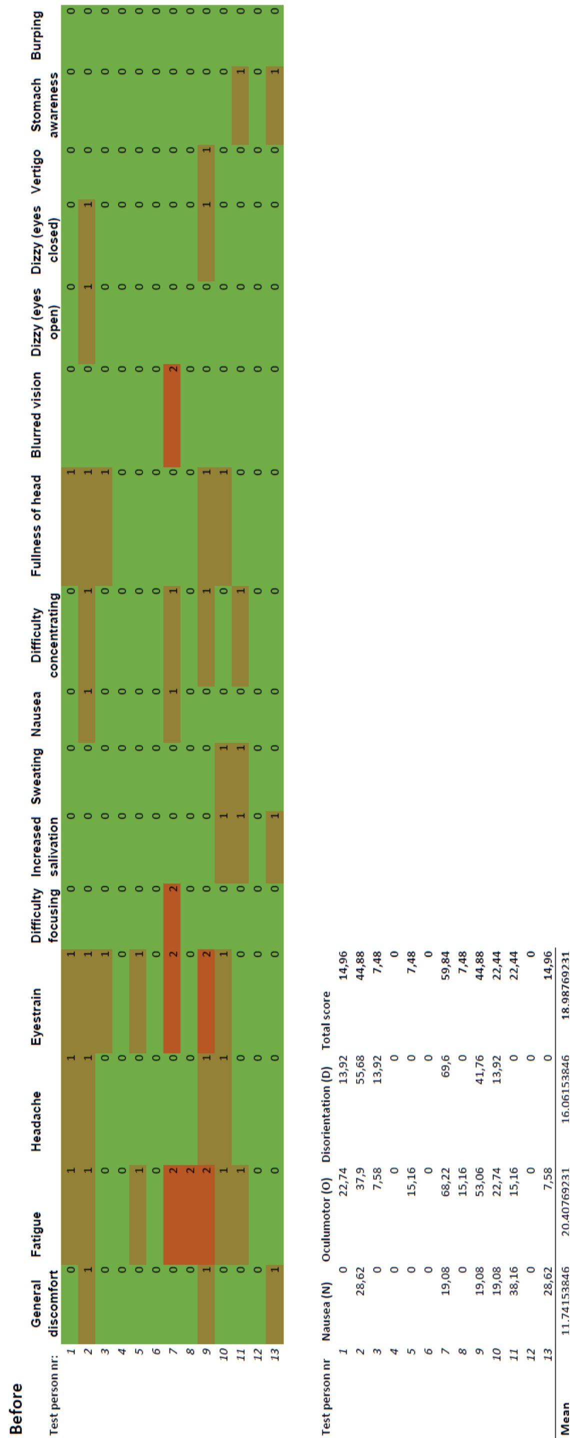
Virtual reality with Oculus compared to an ordinary monitor, do you think that virtual reality is beneficial in a verification purpose?
1 - Yes
1 - Yes, the extra dimension made the distinguishing of objects faster and reliable
1 - Yes, but only during playback in real time while not paused
1 - No

Do you think that there were any objects that the radar-camera sensor should have detected?
3 - Yes
2 - I do not know
1 - No

Which of the ideas on future implementations (see "Possible future implementations") do you think would favor the verification tool the most? Do you have any suggestions for an alternative future implementation?
3 - Color overlay
2 - Add cameras to the Oculus Rift
2 - Add video feed from a car mounted camera pointing forwards
1 - Differentiate between static and non-static objects
1 - Connect neighboring points with lines
1 - "Vacuum packing"
1 - Text information about the object currently being watched
1 - A global static mesh and movable objects as point clouds

Final comments/suggestions
2 - Show the field of view of the radar-camera
1 - Add a virtual car
1 - Show the positions of the sensors on the car, and allow for enabling/disabling them
1 - Mesh within a restricted area of the car since the mesh was more clear close to the car
1 - Show a counter for the number of logged time instances
1 - Show the (x, y, z) position of the user
1 - Filter the point cloud using the intensity values of the points

A.2 SSQ results

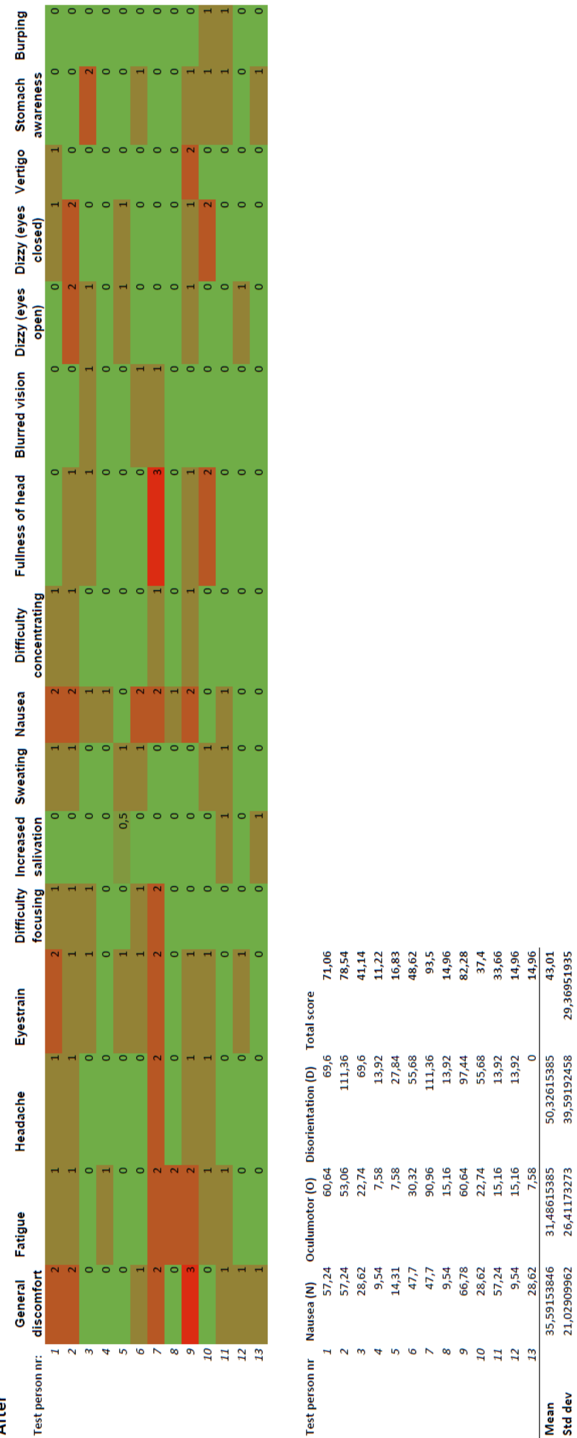


Before

Test person nr:	General discomfort	Fatigue	Headache	Eyestrain	Difficulty focusing	Increased salivation	Sweating	Nausea	Difficulty concentrating	Fullness of head	Blurred vision	Dizzy (eyes open)	Dizzy (eyes closed)	Vertigo	Stomach awareness	Burping
1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0
2	1	1	1	1	0	0	0	1	1	1	0	1	1	0	0	0
3	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	2	0	2	2	0	0	1	1	0	2	0	0	0	0	0
8	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	2	1	2	0	0	0	0	1	1	0	0	1	1	0	0
10	0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0
11	0	1	0	0	0	1	1	0	1	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

Test person nr	Nausea (N)	Oculumotor (O)	Disorientation (D)	Total score
1	0	22,74	13,92	14,96
2	28,62	37,9	55,68	44,88
3	0	7,58	13,92	7,48
4	0	0	0	0
5	0	15,16	0	7,48
6	0	0	0	0
7	19,08	68,22	69,6	59,84
8	0	15,16	0	7,48
9	19,08	53,06	41,76	44,88
10	19,08	22,74	13,92	22,44
11	38,16	15,16	0	22,44
12	0	0	0	0
13	28,62	7,58	0	14,96
Mean	11.74153846	20.40769231	16.06153846	18.98769231

Figure A.1: SSQ results before test



After

Test person nr:	General discomfort	Fatigue	Headache	Eyestrain	Difficulty focusing	Increased salivation	Sweating	Nausea	Difficulty concentrating	Fullness of head	Blurred vision	Dizzy (eyes open)	Dizzy (eyes closed)	Vertigo	Stomach awareness	Burping
1	2	1	1	2	1	0	1	2	1	0	0	0	1	1	0	0
2	2	1	1	1	1	0	1	2	1	1	0	2	2	0	0	0
3	0	0	0	1	1	0	0	1	0	1	1	1	0	0	2	0
4	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0,5	1	0	0	0	0	1	1	0	0	0
6	1	0	0	1	1	0	1	2	0	0	1	0	0	0	1	0
7	2	2	2	2	2	0	0	2	1	3	1	0	0	0	0	0
8	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	3	2	1	1	0	0	0	2	1	1	0	1	1	2	1	0
10	0	1	1	1	0	0	1	0	0	2	0	0	2	0	1	1
11	1	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1
12	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
13	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

Test person nr	Nausea (N)	Oculumotor (O)	Disorientation (D)	Total score
1	57,24	60,64	69,6	71,06
2	57,24	53,06	111,36	78,54
3	28,62	22,74	69,6	41,14
4	9,54	7,58	13,92	11,22
5	14,31	7,58	27,84	16,83
6	47,7	30,32	55,68	48,62
7	47,7	90,96	111,36	93,5
8	9,54	15,16	13,92	14,96
9	66,78	60,64	97,44	82,28
10	28,62	22,74	55,68	37,4
11	57,24	15,16	13,92	33,66
12	9,54	15,16	13,92	14,96
13	28,62	7,58	0	14,96
Mean	35,59153846	31,48615385	50,32615385	43,01
Std dev	21,02909962	26,41173273	39,59192458	29,36951935

Figure A.2: SSQ results after test

A.3 Possible future implementations

A.3.1 Cameras on Oculus Rift

If one were to mount two cameras onto an Oculus Rift, see Figure A.3 below, and playback the image feeds live, one can obtain an AR-like effect. The result is a pair of "see-through" glasses with the advantage that 3D objects can be rendered on the imagery.

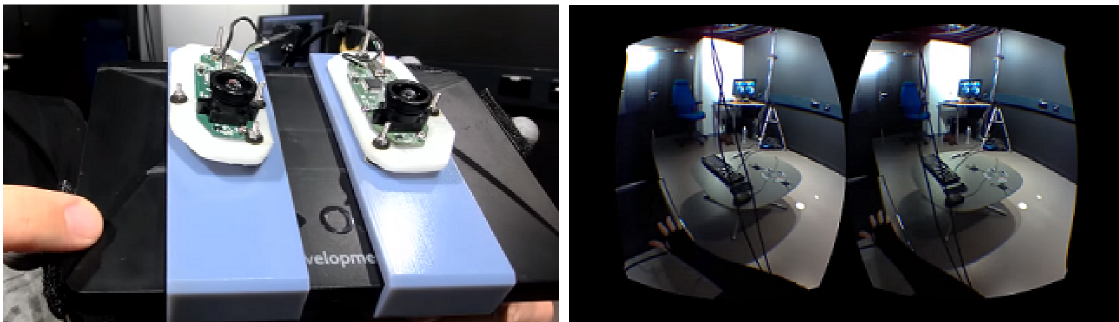


Figure A.3: The images are taken from an example on Youtube by William Steptoe.

With this solution the user would be able to sit in a car and see the environment as the eyes would perceive it, with rendered markers on the objects that the car have detected. It would also be possible to pause and rewind to see something that was within the field of view again.

A.3.2 Color overlay on point cloud or mesh/surface

One idea of improvement regarding showing a point cloud is to overlay the points with a camera feed, that is to give the points color corresponding to the color of an image, see Figure A.4. Though to implement this improvement it is required that a 360 degrees camera is mounted on the car as close to the LIDAR as possible during test drive.

Observe that the image will only be recognised completely if the point cloud is viewed from the centre where the camera and LIDAR are located. If e.g. a tree would be viewed from above the colors would still be mapped to how the tree was looking from the side during the scan.

It would also be possible to overlay a 360 camera feed onto a mesh, a surface recreated from points. Where the mesh does not have its normal angled towards the viewer, i.e. as soon as an environment is scanned that is not a flat wall facing

Figure A.4: An example of color overlay. Observe that this point cloud has a much higher resolution than that of the car's LIDAR.

the LIDAR, the camera image will be stretched out and thus only look good while looking from the location of the LIDAR.

A.3.3 Static mesh

The self driving cars will have access to a global map in the shape of a point cloud which is scanned in advance. This cloud has a much higher resolution than the ones that are live streamed while driving.

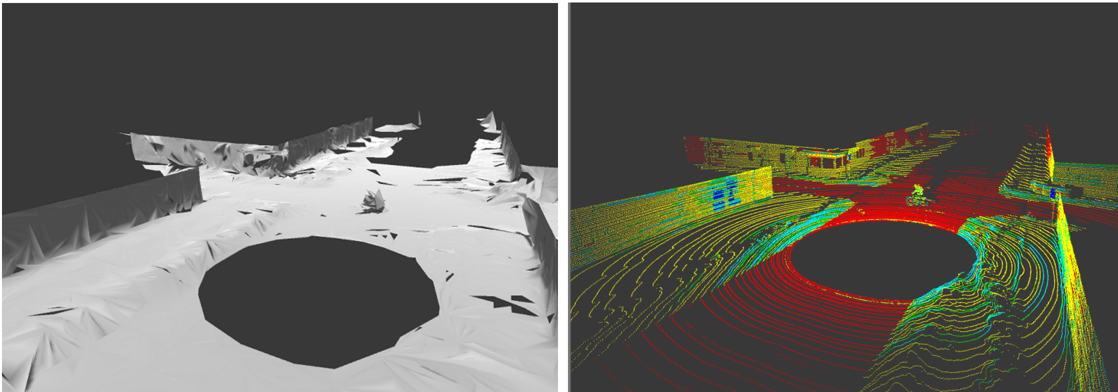


Figure A.5: Print screens from the ongoing project (4/4-16). The large circle is a shadow from the LIDAR, in front of the car there is a cyclist.

An alternative would be to turn this global point cloud into a mesh. All non-static objects in the point cloud of the car can then be masked out and shown together with the mesh. This would use the fact that roads, houses, and other flat surfaces are illustrated well by a mesh and still maintain a level of detail for movable objects with a point cloud.

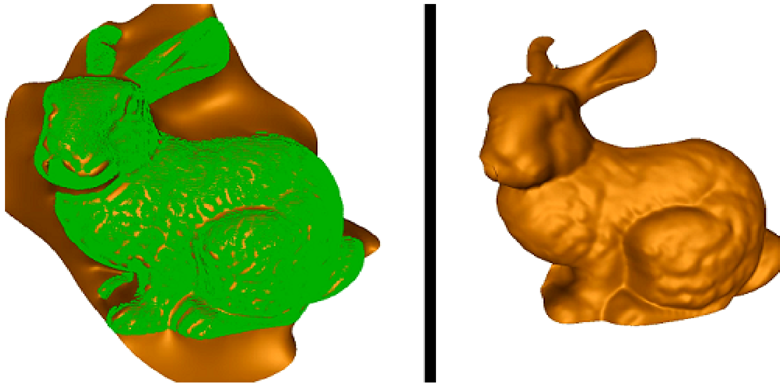


Figure A.6: Illustration of the alternative mesh-algorithm

A.3.4 Mesh from point cloud cluster

As in the previous suggestion, all objects that are moving can be filtered out. These smaller point cloud clusters could then be used to generate meshes using a different technique. Instead of binding points together, this technique can be visualized as "vacuum packing" the points, see Figure A.6.