



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG



Visualization of tests for future active safety and self-driving cars

Master's thesis in Software Engineering

MATTHIAS PERNERSTORFER
RAJESH THANGASWAMY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

MASTER'S THESIS 2016

**Visualization of tests for future
active safety and self-driving cars**

MATTHIAS PERNERSTORFER
RAJESH THANGASWAMY

Department of Computer Science and Engineering
Division of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Visualization of tests for future active safety and self-driving cars
MATTHIAS PERNERSTORFER
RAJESH THANGASWAMY

© MATTHIAS PERNERSTORFER, 2016.

© RAJESH THANGASWAMY, 2016.

Supervisors: Hang Yin and Christian Berger, Computer Science and Engineering
Examiner: Miroslaw Staron, Computer Science and Engineering

Master's Thesis 2016
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Volvo Car using sensors to identify objects in front of it [1].

Gothenburg, Sweden 2016

Visualization of tests for future active safety and self-driving cars
MATTHIAS PERNERSTORFER
RAJESH THANGASWAMY
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

This thesis investigates how sensor data of autonomous vehicles can be visualized in order to make it easier for humans to understand and analyze the data. To achieve this goal, the requirements of a software architecture of a tool that generates such visuals are elicited together with the details that should be included in said visuals. The result of this elicitation phase is used to develop such a software architecture.

Furthermore, different technologies are compared that can be used to implement a tool that uses the created architecture. For this, a feasibility study is done that investigates which technologies are suitable to fulfill the elicited requirements.

This thesis also finds a way how the deviations in the data of different sensors measuring the same object can be visualized. The objects recognized by the different sensors should be at the same place, however, it is possible that the measurements differ. Those differences have to be visualized.

The results of this thesis for the software engineering body of knowledge include a software architecture that is able to fulfill the elicited requirements, as well as the results of the feasibility study, that shows what is and is not possible when using different technologies to implement a similar tool. Furthermore, a reference solution based on some of the technologies is implemented and described in this thesis.

Keywords: Autonomous Vehicles, Sensor Data, Visualization, Active Safety

Acknowledgements

First of all, we would like to thank Volvo Cars for providing us the opportunity to work on this thesis. We would like to thank our academic supervisors, Hang Yin and Christian Berger, for their help, support and expertise that greatly assisted us while working on this thesis. We would also like to express our deepest gratitude to our industrial supervisors Josef Nilsson and Yury Tarakanov for their continuous encouragement, support and valuable feedback throughout the thesis. We would also like to thank all the participants in the focus groups, survey and validation process for their participation and constructive discussions and feedback, as well as the students that worked on other related thesis topics with us at Volvo Cars for their feedback in the weekly review meetings. Finally we would like to thank our family and friends for their constant encouragement and support.

Matthias Pernerstorfer, Gothenburg, June 2016
Rajesh Thangaswamy, Gothenburg, June 2016

Contents

List of Figures	vii
List of Tables	ix
Glossary	1
1 Introduction	2
1.1 Problem Domain and Motivation	4
1.2 Research Goal and Research Questions	4
1.3 Scope	5
1.4 Document Structure	6
2 Background	7
2.1 Autonomous Vehicles and Safety	7
2.2 Data Visualization	8
3 Related Work	11
4 Methods	15
4.1 Literature Review	16
4.2 Architecture Requirements and Visualization Details	18
4.2.1 Focus Groups	18
4.2.2 Survey	19
4.2.3 Analysis of Similar Systems	20
4.3 Architecture Requirements	20
4.3.1 Creation and Prioritization of Requirements	20
4.3.2 Validation	22
4.4 Parameters and Features	22
4.5 Technologies	23
4.6 Deviation between Sensors	24
5 Results	26
5.1 Architecture Requirements	26
5.1.1 Functional Requirements	26
5.1.2 Non-Functional Requirements	27
5.1.3 Use Cases	27
5.1.4 Architecture	27

Contents

5.2	Parameters and Features	30
5.2.1	Sensor Data	32
5.2.2	Identified Parameters and Features	34
5.2.3	Included Parameters and Features	35
5.3	Technologies	35
5.3.1	Feasibility Study	35
5.3.2	Chosen Technology	40
5.4	Implementation	41
5.5	Deviation between Sensors	50
5.5.1	Iteration 1	50
5.5.2	Iteration 2	51
5.5.3	Iteration 3	53
5.6	Validation	53
6	Discussion	57
6.1	Elicitation phase	57
6.2	Architecture creation	59
6.3	Technology selection	60
6.4	Development	62
6.5	Deviation between Sensor Data	65
6.6	Validation	65
6.7	Threats to Validity	66
6.7.1	Construct Validity	66
6.7.2	External Validity	67
6.7.3	Internal Validity	68
7	Conclusion	69
7.1	Future Work	70
	Bibliography	72
A	Focus Group Questions	I
A.1	Opening Question	I
A.2	Introductory Question	I
A.3	Transition Question	I
A.4	Key Questions	I
A.5	Ending Questions	II
B	Survey Questions	III
C	Statement of Contributions	VI

List of Figures

1	The AstaZero test track [2].	3
2	Example setup of sensors on an autonomous vehicle [1].	3
3	Car autonomously breaking for a cyclist [1].	8
4	Visualization of sensor data in an automobile context [1].	9
5	The different phases of this thesis.	15
6	The research questions and methods used to answer them.	16
7	The literature review approach as proposed by Kitchenham [3].	16
8	The design science approach as described by Hevner et al. [4].	24
9	The created architecture.	31
10	The area sensed by the first sensor.	32
11	The area sensed by the second sensor.	34
12	The main window of the application.	42
13	The bird's eye view of the visualization.	43
14	The text view of the visualization.	44
15	The time series view of an object.	44
16	The objects along with its ID in the visual.	45
17	The bird's eye view showing map, lane and textures.	46
18	Apply filter window.	47
19	Create filter window.	47
20	The same object recognized by two different sensors. Iteration 1.	51
21	Comparison of different objects in the scene. Iteration 2.	52
22	Comparison of different objects in the text view. Iteration 2.	52
23	Comparison of different objects in the text view. Iteration 3.	53
24	Comparison of different objects with transparent color.	54
25	Validation question 1.	54
26	Validation question 2.	55
27	Validation question 3.	55
28	Validation question 4.	56
29	First survey question.	III
30	Second survey question.	III
31	Third survey question.	IV
32	Fourth survey question.	IV
33	Fifth survey question.	IV

List of Figures

34	Sixth survey question.	V
35	Seventh survey question.	V
36	Eight survey question.	V

List of Tables

1	The four research questions to be answered in this thesis.	5
2	Prioritization process example 1.	21
3	Prioritization process example 2.	21
4	An example of a functional requirement.	26
5	An example of a non-functional requirement.	27
6	An example of a use case description.	28
7	Object information in sensor 1.	33
8	Traffic sign information in sensor 1.	33
9	Object information in sensor 2.	33
10	Results of the Feasibility Study on functional requirements.	37
11	Results of the Feasibility Study on non-functional requirements.	38

Glossary

- .pcap** Packet CAPture. A file associated with the wireshark application that contains a log of network traffic. 22, 41, 48, 71
- data model** The structure of the data stored in the tool. 22, 29, 30, 48, 60, 63, 71
- feature** An object. For instance a car or a pedestrian. 4, 5, 22, 29, 30, 32, 34, 35, 57, 58, 63, 67, 69, 71
- offline** One of the modes of the tool. The data is read from a local file, and processed before the visualization starts. 41, 42, 48, 50, 63, 64, 67
- online** One of the modes of the tool. The data is received over a UDP socket and visualized live as it arrives. 41, 42, 48–50, 63, 64, 67, 71
- parameter** A mathematical value. For instance speed or distances. 4, 5, 22, 29, 30, 32, 34, 35, 42, 43, 57, 58, 63, 67, 69, 71
- scene** The entity that contains all objects in the visuals together with their parameters. The camera captures the scene and creates a visual of it. 69, 71
- UDP** User Datagram Protocol. A connectionless network protocol. Messages are sent without establishing a communication channel first. 22, 29, 30, 39–41, 48, 49, 63

1

Introduction

Autonomous vehicles (AVs) provide the possibility for fundamentally changing the current way of transportation. Making new generation cars and other light vehicles with this technology can reduce the accident rate, energy consumption, pollution and cost of congestion. The change in technology is a step by step process from human controlled to self controlled cars [5]. Currently making the car control itself is one of the biggest challenge for the automobile industry and also a major area of research. Intelligent Transport System (ITS) is a global phenomenon, which applies advanced communication, information and electronics technology to solve the transportation problems [6]. Through intelligent active and passive safety systems, the comfort and safety of the passengers and drivers have been improved very much in the past decade. The former helps to mitigate or avoid accidents and the latter reduces the severity of accidents [7].

Volvo Cars, a pioneering company in Active Safety, will bring self-driving functionality in their cars to customers within a few years. Volvo has access to the AstaZero test track [2], as shown in Figure 1. On this track, the self-driving cars are tested and a lot of data is generated through different sensors in the cars. An example setup of sensors is shown in Figure 2. The test data is made available online in a remote test command center. Visualization and logging of all incoming data in real time will allow for efficient test monitoring and “on spot” test data collection and analysis, and to perform this a tool is very much essential. Since self-driving cars have many sensors, all the data may not be needed for humans to analyze. Therefore an empirical study using the generated data needs to be conducted to find the level of detail necessary for human use. This will help to better evaluate the data and benefit the research of truly AVs.

This thesis is part of the iTRANSIT (intelligent TRAffic maNagement System) project, a national research project on the development and testing of intelligent transportation systems [8]. Researchers and engineers from Volvo Cars, SP, Chalmers, ÅF, and AstaZero are part of this project. The goal of this project is the possibility to develop components of an ITS, merge the data from different actors to a global dynamic map and to create a cost efficient positioning system based on GPS and data fusion [8].

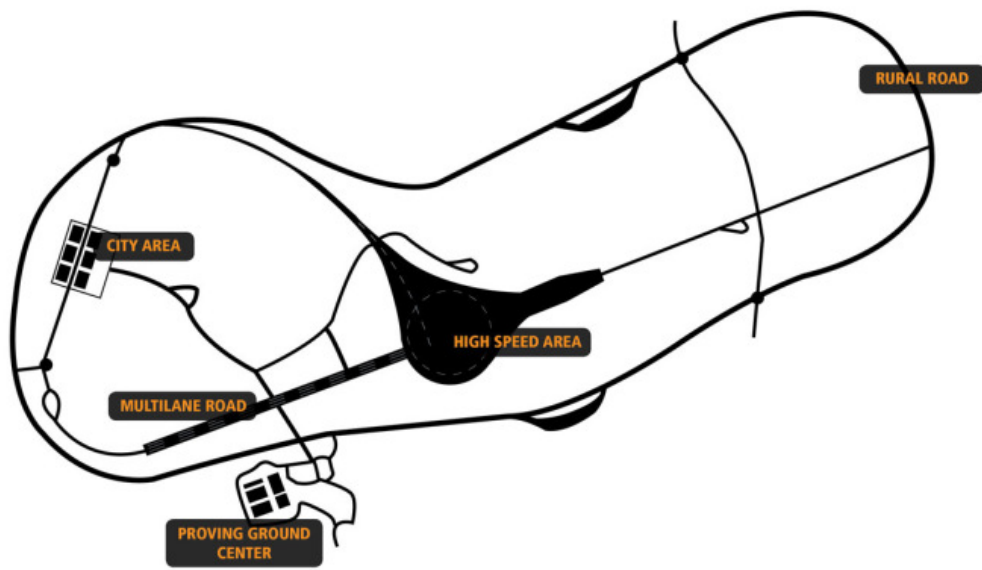


Figure 1: The AstaZero test track [2].

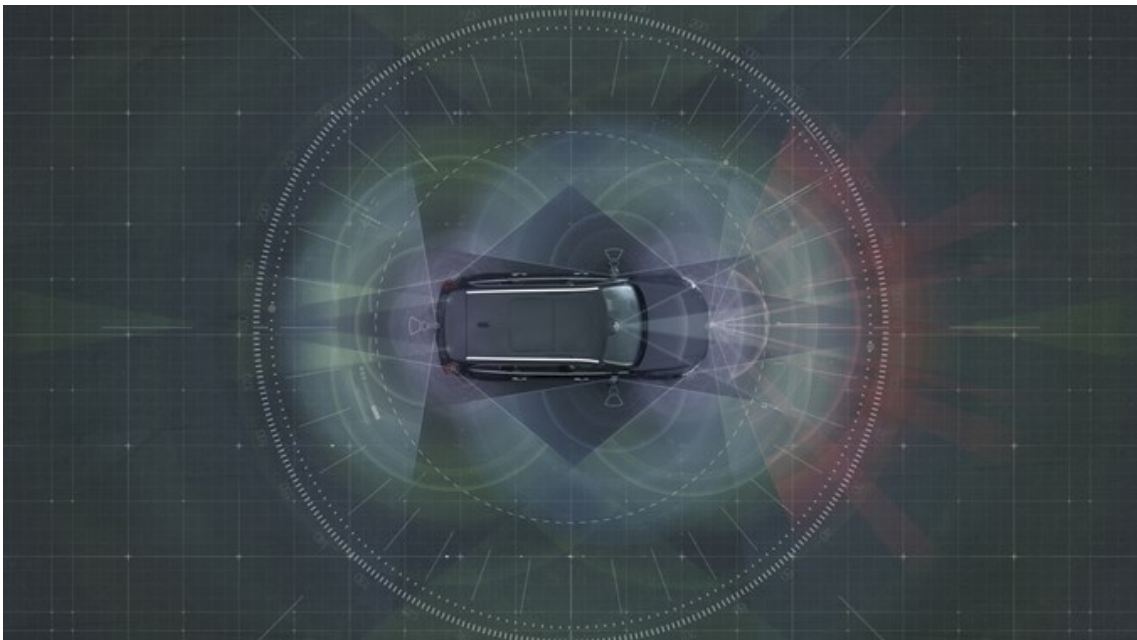


Figure 2: Example setup of sensors on an autonomous vehicle [1].

1.1 Problem Domain and Motivation

AVs have multiple sensors that generate a lot of data. The problem this thesis addresses is how sensor data of AVs can be visualized in order to make it easier for humans to analyze the data and understand the behaviour of the vehicles on the test track. The test track includes city area, rural area, high speed area, roads with multiple lanes etc. In such situations it is crucial to know how the AVs understand these different environments and traffic conditions in order to improve decision making in AVs.

The raw data consists of numbers in different formats and is hard for humans to comprehend. Visualization of this data helps researchers and engineers to understand the behaviour of the AVs during testing. All the generated data will not be needed to visualize at all times, so extraction of the necessary parameters and features at any point of time is essential. Furthermore, some details might be more interesting in some cases, and less useful in other cases, so one of the challenges in this thesis is to find out the details that are supposed to be included.

The visualization of the data while the test happens on the test track is another challenge this thesis tries to solve. This live visualization shall create the visuals within a limited timeframe. Since the amount of generated data is very large and the extraction has to happen fast enough to fulfill this time constraint, it is a challenge to extract the needed information quickly. Visualization shall also be possible at a later point in time from the tests logfiles. It is important that the visuals generated in both modes have the same look and feel to allow efficient analysis. These visuals help the engineers and scientists to gain insight on the behavior of AVs when different events happen while driving and subsequently to be able to develop a system that can guarantee safety for vehicles and passengers.

1.2 Research Goal and Research Questions

The goal of this thesis is to find out how sensor data of AVs can be visualized in order to make it easier for humans to analyze the data and understand the behaviour of the vehicles. To achieve this goal, a tool is developed that generates visuals based on sensor data. The architecture of this tool is important to efficiently generate live visual representations of sensor data. Therefore, it is essential to elicit the requirements of this architecture. Furthermore, it is important that the right parameters and features are included in the visuals and that they are presented in a suitable way. This thesis tries to find out what should be included in the visuals to make them useful for humans to analyze. There are a lot of technologies that can be used to implement a tool that visualizes sensor data. Therefore, technologies have to be found that are suitable to implement a tool that fulfills the elicited

requirements and creates visuals containing the useful parameters and features. It is also important to detect deviations in the data from different sensors measuring the same object and visualize them. These challenges are formulated as the four research questions shown in Table 1.

Number	Research Question
RQ 1	What are the functional and non-functional requirements for an architecture model to visualize data from automated vehicles testing?
RQ 2	How can the necessary parameters and features be identified and automatically extracted from the sensor data without loss of necessary information in order to generate a visual representation?
RQ 3	What existing technologies can be used to develop a tool that implements an architecture to visualize data from AV testing?
RQ 4	How can the generated visuals be used to identify the deviation in the data of the same object in different sensors?

Table 1: The four research questions to be answered in this thesis.

1.3 Scope

The requirements are elicited from selected people at Volvo Cars, Chalmers University of Technology and from literature. Based on those requirements, suitable technologies are chosen to implement the tool. There are a lot of available technologies that can be used for development. For this reason, a pre-selection for further investigation based on certain criteria is necessary. The most feasible technologies are chosen for implementation.

A self-driving car uses multiple sensors to gather all necessary data to make autonomous decisions. This thesis only considers visualizing the data from some important sensors. The decision, which sensors are included depends on the data made available from Volvo Cars. To be able to select the parameters and features to include in the visuals, the necessary information has to be available in the provided sensor data.

The tool that is developed as a part of this thesis is a proof of concept to show that an architecture that satisfies most important requirements can be implemented using the chosen technology. In cooperation with the main stakeholders, requirements are chosen for implementation based on their priority and the result is used to demonstrate the concept. The developed tool can then be used as a base for future development.

Multiple sensors can cover the same area and therefore identify the same objects. The tool is also able to visualize the deviation in the data of two sensors measuring

objects in the same area. The exact way that the overlapping is visualized will be identified during this thesis.

This thesis gives the software engineering community knowledge about the requirements of a software architecture that is used to provide data visualization online (live) and offline, as well as a reference architecture which can serve as a model for future visualization tools with similar goals. Furthermore, this thesis gives the reader understanding about different available technologies that can be used to develop similar tools, as well as their advantages and disadvantages.

Handling large amounts of data generated from multiple sensors in a short time is a challenge in automotive and other industries [9, 10]. This thesis investigates how the necessary information can be extracted and visualized both live with time constraints, as well as from a logfile. Furthermore, it shows that both ways can create visuals with the same look and feel and provide a common way for humans to analyze data.

1.4 Document Structure

This thesis starts with a description of the background in chapter 2, followed by an analysis of related work in chapter 3. This gives the reader insight in the domain and shows how other researchers have solved similar problems. Chapter 4 describes the methods used in this thesis and shows how the research questions are answered. The results of those methods are described in chapter 5, which provides detailed explanation of the insights gained from conducting the steps in chapter 4. A discussion of the results is provided in chapter 6, including an analysis of threats to validity. This thesis is then concluded and summarized in chapter 7.

2

Background

This chapter introduces the needed background knowledge to understand why automated driving improves the safety of vehicles and passengers and how visualization of test data help engineers to build safer cars.

2.1 Autonomous Vehicles and Safety

Active safety is a vehicle's ability to avoid accidents (as opposed to passive safety that reduces the impact of accidents) [7]. To be able to provide active safety, intelligent vehicle safety systems are needed [7]. Cars driving autonomously will reduce accidents and improve safety for the passengers [7, 11, 12]. Autonomous vehicles do not consciously break traffic laws, never drive drunk or after having taken drugs and will react quicker than humans. According to a study in the United States in 2013 40 % of all car crashes involve alcohol, drugs, distraction or fatigue [12]. None of those factors affect autonomous vehicles. Cars today are able to do more and more tasks autonomously. Many new cars include features like adaptive cruise control, lane control and parking assistance, as well as automated breaking, for instance for a cyclist as shown in Figure 3. [11, 12]. By 2020 many large automobile companies hope to release fully autonomous vehicles [11, 12].

There are still open issues concerning autonomous vehicles [11, 12]. Governments have to allow the operation of self-driving cars and provide a way to certify autonomous vehicles as safe for public roads. Furthermore, the liability in case of an accident has to be clear. As of now, it is not clear if it is the car manufacturer or the company who built the sensor or someone else's fault if a car crashes. This becomes even harder to decide if an autonomous vehicle crashes with a human driven one. Some people are also sceptical to autonomous vehicles and do not want to give a computer control over their vehicle, or simply like to drive and prefer to continue doing so themselves. Privacy is also an issue, because autonomous vehicles are able to gather data about where they go.



Figure 3: Car autonomously breaking for a cyclist [1].

By having a high enough market share of autonomous vehicles that have the possibility to communicate with each other and preferably also the public infrastructure, congestion can be reduced [12]. This is because the safety gap between cars can be reduced in this scenario and as a result, a better flow management is possible. Furthermore, cars can be parked further away from the destination of the passengers, by dropping them off and then proceeding empty to a nearby parking lot and picking the passengers up when they want to drive somewhere else. This will eliminate cars circling around the block in wait of an available parking space, which results in reduced exhaust fumes which is good for the environment.

Autonomous vehicles will allow persons to drive that are not able to do so today [12]. Old people that can't drive anymore as well as young people that are not allowed to get a drivers licence yet can use self-driving cars for transport. Furthermore, humans with disabilities will benefit from this advantage. Autonomous vehicles will make transportation available to a wider audience while improving safety for everyone.

2.2 Data Visualization

Data visualization is a fast growing field [13], as it serves as a great communication tool [14]. It finds application in business, automobiles and most other fields. An example of data visualization in the automobile field is shown in Figure 4. With respect to the field of application, the following four things have to be considered [14]:

1. What will be visualized?

2. How is it visualized?
3. How much is visualized truly?
4. What has been visualized exactly?



Figure 4: Visualization of sensor data in an automobile context [1].

The first question finds out the necessary information to be visualized. The second question identifies a visualization technique or the need to develop a new technique, which can be used to visualize the data. The third question validates the used technique and the fourth question explains the results obtained from visualization.

Along with the data, the users are also important and it is essential to know their preferences. Generally, for any kind of visualization the user expects the following features [12, 15]:

1. Overview: Have an overall view of the system.
2. Zoom: Zoom in on the interested things.
3. Filter: Filter out uninterested things.
4. Details-on-demand: Select an item or a group and get details when needed.
5. Relate: View the relationship among the items.
6. History: Replay, undo, refine, etc.
7. Extract: Extract sub-collections of the query parameters.

AVs have many active safety features that are achieved through multiple sensors. Data streams generated by sensors are continuous and can have very high data rates and volume, so analyzing them through visuals makes it easier for the users to acquire information and knowledge [13, 16]. In an environment where a lot of continuous data is generated, it is almost impossible to visualize all the data, so the user needs have to be considered in order to extract the relevant data [15]. Understanding and extracting the information from the data for further analysis is challenging [17]. Data arrives in a bursty mode and it is hard to process all data in real time. Handling the data rate, noises in the data, order of the data and missing data is also challenging [13] and moreover, the data has to be sampled at a suitable rate [17]. The data can have different parameters, and changes over time can affect one or multiple parameters, while others such as geometry remains constant [16]. For instance, if the sensor data has the information of a car, the parameters like height, width etc., of the car is unchanged while its position can change. So an efficient way of handling the data can improve the performance and can make the rendering happen in (near) real-time.

Visualising the sensor data in a browser based client increases the accessibility, but web based 3D visualization of continuously changing data is challenging. Moreover, transmitting the data directly to a browser has performance and scalability issues, so the data streams have to be processed in an efficient way. For fast 3D rendering, the system requires significant processing power and an optimized data structure like Binary Space Partitioning [16].

3

Related Work

This chapter presents related work in the field of sensor data visualization. To identify related work, the literature review approach described in section 4.1 was used. Since automated driving is a relatively new concept, not many papers have been published on visualizing tests of AVs. Instead, this chapter presents work that has been done on developing visualization tools for vehicles and geospatial sensor data visualization in general.

Sedlmair et al. [9] have developed Cardiogram, a visual analytics system that supports engineers in analyzing large data sets. The data that is analyzed comes from in-car communication networks and contains millions of messages. The tool was created as a result of a three year field study in a large automotive company. The goal this tool aimed to achieve was to integrate it directly in the industrial work context. To be able to do that, the creators worked closely with domain experts to learn their everyday work. They tested various prototypes to ensure that the tool fulfilled the industrial work requirements.

Cardiogram was developed in close collaboration with the end users. This was done as an in-depth long-term case study. A field study of current practices included interviews with single engineers, user observations, as well as multiple focus groups, that aimed to bring a common understanding between the users themselves. Furthermore, the tool itself was validated multiple times using design workshops and personal feedback as well as lab studies and field studies.

The people behind Cardiogram claim that the key to success was understanding the available data, the tasks the engineers want to perform and the tools currently available. Based on this knowledge the requirements of Cardiogram were identified. The tool has to be able to handle masses of data. To be able to do this, it filters the messages automatically and supports the error detection. Furthermore, Cardiogram should provide new insights. This is achieved by analyzing timing aspects and message propagation, detecting outliers and viewing correlations between messages.

The tool is implemented using a two step design. The first step contains data pre-processing and storage. Here data filtering, error detection and data abstraction

takes place. The second part is the visualization. Here the results of the first steps are visualized and the correlation between time and logic becomes clear.

The process used to develop cardiogram satisfied specific factors that the authors claim were necessary for the tool to get adopted. The tool has to be simple to use, so that the users don't have to spend a lot of time learning the tool. Furthermore, strong user integration is needed through the entire design processes to make sure that it satisfies the needs of the end users. Lastly, a tight integration to existing tools and workflows is needed.

M. Tönnis et al. [18] have built a visualization system for spatial sensor data. The system has two different setups; the laboratory setup and the car setup. In the laboratory setup, the perception sensor data provided by pre-recorded scenes is rendered in a down-scaled form. In the car setup, the sensor data is visualized in real-time, which allows for direct inspection. In both setups, two visualization devices have been used; a video see-through LCD flat panel (TFT) and an optical see-through head-mounted display (HMD). Visualization schemes for spatial sensor data and for geometric models that outline recognized objects were developed. The main intention for the development of the system is to reduce the gap between the interface designers and the sensing engineers during the development phase. The idea behind this system is to check the functionality of the sensor, debug and understand the sensor data and serve as a platform for future driver assistance systems.

Llorach et al. [10] have built a visualization system for a Massive Multiplayer Online (MMO) virtual regatta using a virtual globe. A regatta is a sailing race around the world in the shortest amount of time. A MMO virtual regatta is an online game, which allows members of the public to sail a virtual boat around the world, competing in real-time with the real boats of the regatta. The virtual globe is the 3D representation of the globe. The work shows some of the possibilities of the browser for visualizing large data sets of trajectories on the client.

An Itinerary of 17,000 boats sailing over the ocean around the world, featuring over 20 million points is visualized in a web browser environment. The data is presented and drawn on the top of a virtual globe, which allows users to navigate, interact and focus on any area and is also useful for visualizing information on the globe from different perspectives and scales. For efficient storage, samples are added to the data set based on some strategy and each data sample contains the latitude, the longitude and a timestamp.

Two different approaches are proposed to visualize the data. The first approach shows the evolution of the race over time, which renders only a part of the trajectory of each boat. This is achieved by taking only the last samples where each boat has gone through and a line is drawn, like a tail, behind each boat. In the second approach, the full trajectories are progressively rendered into a framebuffer object, resulting in a density map of the areas where users passed more often. This density

map can be created both in real-time, as well as offline, where the latter achieve high resolutions and detail. The path visualization shows precisely the trajectory of each individual boat over time. The density map is useful to understand where the players thought the winds would be better to win the race and which routes they preferred.

Stampach et al. [19] have built a context based dynamic visualization system for sensor data that varies in space and time. Context is defined as the synthesis of conditions which can influence an adaptive map. Each context is a combination of parameters, describing a user or his actual situation. Cartographic techniques are used for monitoring, predicting and analysing the situation in space and time. Geoinformation technologies allow users to examine and represent spatial information on a pre-existing map through interactive cartography, described as adaptive cartographic visualization.

A sensor network consisting of multiple sensor units serves as the source for near real time data. Each sensor unit measures multiple characteristics (temperature and humidity of both air and soil). A GPRS based mobile communication unit serves as a relay station to communicate the measurement to the remote server. The remote server stores the received data in a database. These measurements are then sent to the web-based client applications. The clients have two different views, the spatial (map) view and the temporal (chart) view. The user can switch between the views, but can't view them simultaneously.

The spatial visualization is in the form of a map, which contains predefined hierarchical levels of detail (from an overall view of the area of interest to the detailed view of the immediate surroundings of the selected group of sensors). In the temporal visualization, an interactive chart is drawn where the user is allowed to choose the sensor and the time range. The user can also select multiple sensor measurements and compare them to find the variations between different measurement points. Below the chart, the user can read the statistical information about the currently displayed selection. This view supports features like exporting the chart and print, zooming and specifying the level of aggregation (week, month).

As multiple measurements were involved, the complexity of symbols was a big issue and the information that was not relevant to a particular context was omitted to manage the simplification of the symbols. Implicit symbols with reduced minimal size without any numerical or textual content were used and by hovering the mouse on the symbol they were enlarged and the numbers and texts in the symbol got visible. In this way, an overview and details-on-demand were provided to the users. The system supports the basic features as described by Fagant et al. and Zankl [12, 15].

C. Roessig et al. [20] developed a framework that visualized information about speed, distance and potential danger in the rear-view mirror of a car. This was done using backward-looking cameras to gather information about approaching vehicles.

Overlaying the collected information over all vehicles would clutter the image too much, so the work aimed to find a better visualization technique.

Multiple ways were presented how the human perception of speed, distances and danger can be influenced. An artificial depth-of-field can be used to draw the user's attention towards a specific part of the visual by manipulating depth information. Motion blur makes the user subconsciously believe that an object moves at high speed. Furthermore color can be used to draw the user's attention to important objects. A color scale from red (danger) to green (safe) can be used to classify threat.

A camera was used to identify lanes and objects. This information was then used for distance and velocity estimation, as well as for calculating the risk potential, which was the estimated time to impact. The result was then visualized in different ways. The lanes, the own vehicle can switch to can be colored in green if it is safe to switch and red if another vehicle is too close. The artificial depth-of-field can be used to make an approaching car look close to make the user aware of it while making the background seem very far away. Lastly motion blur can be used to give the user information about the speed of other vehicles.

All related work mentioned in this chapter addresses handling and visualizing huge data sets. Most of the work covers either an online mode with live or real-time data or an offline mode using logfiles. M. Tönnis et al. [18] have developed both an offline mode for usage in the lab and an online mode for the car itself. These two modes use different setups and approaches to visualize the data. The research didn't reveal any work where online and offline modes were realized with the same setup and approach. This gap is addressed in this thesis.

4

Methods

The overall approach was divided into four different phases, as shown in Figure 5. In the first phase, the requirements of the architecture were elicited. In parallel, the details that needed to be included in the visuals were identified. In the second phase, the results from the first phase were used to identify the suitable technologies to implement a tool based on the architecture. In the third phase, the requirements selected for implementation from the first phase were implemented using the technologies identified in the second phase. After generating the visuals, the differences in the data of different sensors measuring same object were visualized. Once the tool was developed, a user validation was done to understand whether the tool satisfies their needs.

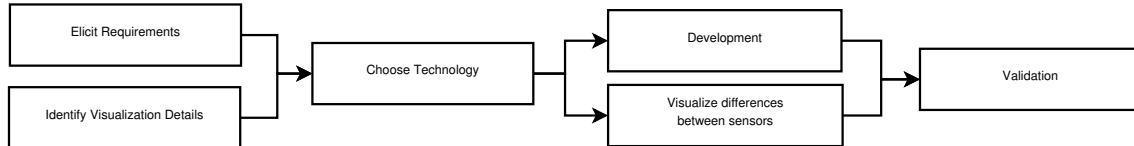


Figure 5: The different phases of this thesis.

The goal of this thesis was to find out how to visualize sensor data to make it easier for humans to analyze the data. To fulfill this goal, the four research questions described in section 1.2 had to be answered. This was done using different scientific methods: two focus groups, a stakeholder survey, an analysis of similar systems, a literature review, a feasibility study and a data study, as well as different forms of validation techniques. The process is described in Figure 6, which shows how the goal is split up in the research questions. The text next to a research question indicates the methods used to answer that specific research question. This chapter starts with an explanation of methods used in multiple research questions, followed by the individual research questions and the methods used to answer them.

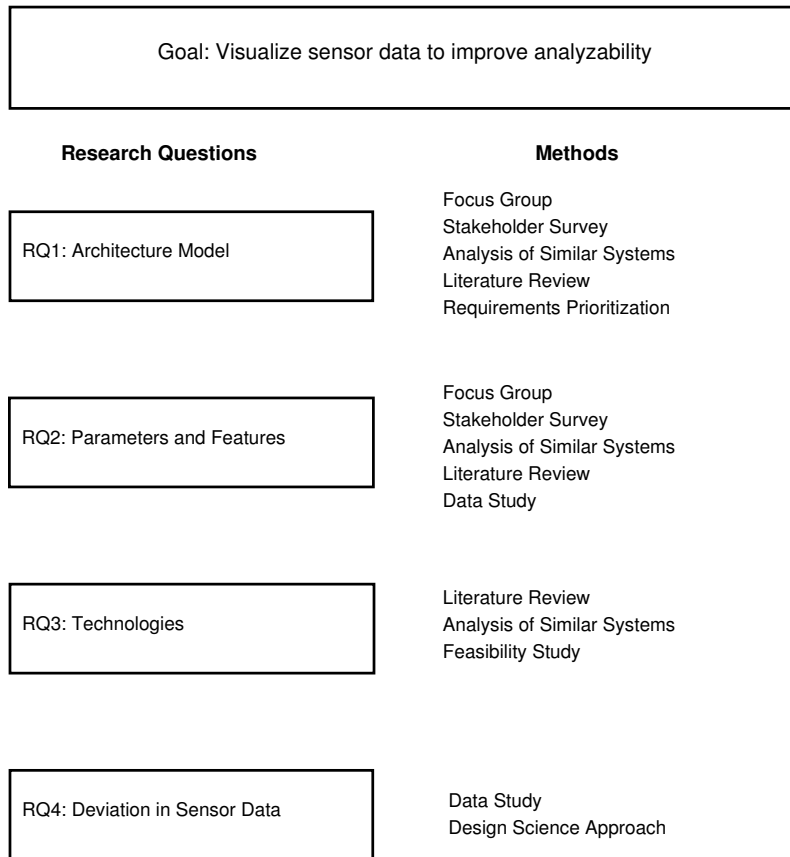


Figure 6: The research questions and methods used to answer them.

4.1 Literature Review

Literature review was part of the methodology to answer the first three research questions. To increase the scientific value of the research, the method was based on the SLR (Systematic Literature Review) approach proposed by Kitchenham in 2004 [3]. The literature review consists of three parts: planning the review, conducting the review and finally reporting the review. The first two parts consist of multiple sub steps [3]. The entire process is shown in Figure 7.

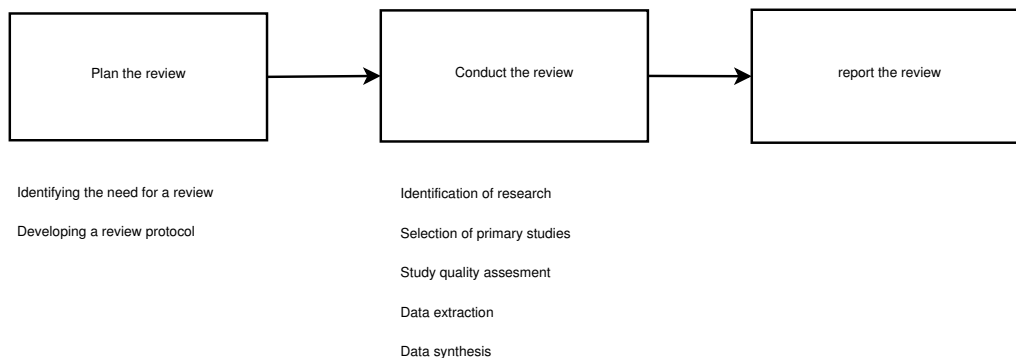


Figure 7: The literature review approach as proposed by Kitchenham [3].

A review method based on the SLR was used to summarize existing information in an unbiased manner. First the review was planned. The rationale for the survey and the research questions to be answered are specified in chapter 1. Since there was a time plan for the thesis itself, no additional time plan for the literature review was created. The review protocol contained the following information:

- Study selection criteria and procedures.
- Study quality assessment.
- Data extraction strategy.
- Synthesis of the extracted data.

The literature review was performed mainly using four different scientific databases, namely ACM DL, IEEE XPLORE, SpringerLink and ScienceDirect. Furthermore the Chalmers library was included, which searches in 177 different scientific databases [21]. The databases were queried with certain keywords that depended on the research question. The results were ordered by relevance and chosen based on different criteria. First the title of the publication had to match the research area. If that was the case the abstract was read. If the publication matched, it was included in the research and data was extracted from it. Sometimes it was not enough to read the abstract. In that case, the conclusion was considered as an inclusion criteria as well. Since the number of papers returned by a search query was usually very large, the assessment had to stop before all papers were read. If ten papers in a row were considered not relevant for the research, the assessment was stopped. If a publication was included, the quality of it had to be assessed. To assure quality, only peer-reviewed papers published in scientific journals and conference proceedings were considered. Studies that were still included at that point were read by one of the students. He took notes about everything that was important to the research question being answered. The other student validated the notes by reading the study as well, checking the notes of the reader and adding anything that he considered missing. The joint result was the data extracted from the study. When the relevant papers were read the knowledge gained from the papers was merged to one dataset.

Instead of reporting the results of the review in a journal or conference paper as proposed by Kitchenham, the results were merged with the results from the other methods used in this thesis and documented in chapter 5.

4.2 Architecture Requirements and Visualization Details

Some methods were used to answer both RQ1 and RQ2. This includes the focus groups and the survey. The analysis of similar systems was important for the choice of technology, as well as the architecture requirements and visualization details. Instead of explaining the methods multiple times in different places, all common methods are explained in this section. Furthermore, the literature review described in section 4.1 was used for all the research questions.

4.2.1 Focus Groups

To elicit the requirements the architecture has to fulfill and to identify the parameters and features that have to be included in the visuals, two focus groups were conducted. The first one was held with engineers at Volvo Cars and the second one with academic experts at Chalmers University of Technology. The focus group procedure was based on the book "Understanding your Users (Second Edition)" [22] and consisted of multiple steps. The research problem was defined in the research questions in chapter 1.2. The questions asked in the focus group can be found in the appendix of this thesis. The initial version of the questions were validated by testing them with two other master thesis students. Any questions they thought were unclear were reconsidered.

Participants were invited by the main stakeholders at the Sensor Analysis and Verification group at Volvo Cars and the supervisors of this thesis at Chalmers University of Technology. The participants included engineers and researchers who voluntarily agreed to join this study. In total, four persons participated in the first study, and two in the second.

Each focus group sessions lasted about one hour and consisted of multiple phases. The sessions started with a short presentation of this thesis, followed by an introduction of the participants. An introductory question acted as a starting point of the discussion, before a transition question led the attendees to the topic of AVs. The key questions aimed to give answers to the research questions and were used to elicit stakeholder requirements. The sessions were closed by two ending questions. The first was used to identify further stakeholders that had not been considered yet, while the second one aimed to identify what was most important to the participants.

One of the students working on this thesis acted as the moderator who led the discussion, while the other student took notes and audio recorded the sessions. After the session, the two students had a debriefing session, where observations and thought were exchanged. This was done directly after the focus group to ensure

that no ideas got lost. After that, a transcript of the audio recording was generated and used as a basis for further analysis.

In the transcripts, all the wishes and ideas relevant to the aim of the thesis that were brought up were identified and written down on a sheet of paper. This was done by reading the entire transcript from the beginning to the end and as soon as a point was identified, a note was taken. The notes were then grouped based on categories and duplicates were merged. Each note was then reformulated to a requirement sentence, starting with "The tool shall be able to..." or "The user shall be able to...".

4.2.2 Survey

After the focus groups and the analysis of similar systems, a survey was conducted at Volvo Cars in order to validate and prioritize the known requirements and also to elicit more requirements, as well as to gather more knowledge on the details to include in the visuals. The survey included 8 questions (some questions contained sub-questions) and could be answered within 5 minutes. It was hosted on a Volvo Cars SharePoint site. Initially the survey was sent to the main stakeholder. After that pilot survey, an invitation to participate was sent to three workgroups at Volvo Cars, with a total of 53 recipients. The survey was left online for two weeks and multiple reminders were sent in this timeframe to get a higher response rate. Eleven persons answered the survey.

The survey included questions on how the engineers would like to use a visualization tool that visualizes sensor data. This included if they want an online or offline mode (or both), as well as on what platforms they would like to use the tool. Furthermore, a question was included where the users could choose what features to include in the visuals, with an option to add additional features not in the list. The survey also had a question on what views should be included in the tool. A view in this context is the perspective the scene is rendered from, for instance bird's eye or first person. As before, the participants were able to suggest their own ideas. The final section of the survey covered what the tool should be able to do. It featured a list of requirements gathered from the first focus group and the analysis of similar systems. The requirements that were included in the survey had to be about a user task and not a technical requirement. The participants were able to choose how important they think that requirement is for them. Once more they were able to suggest additional features. The survey ended with a field where the participants could add additional thoughts on sensor data visualization. The questions of the survey can be found in appendix B of this thesis.

Additional wishes and suggestions were transformed into requirements by reformulating them to a requirement sentence, starting with "The tool shall be able to..." or "The user shall be able to...".

4.2.3 Analysis of Similar Systems

On top of the literature review, two presentations of similar tool were visited. The presentations took place at Volvo Cars during the requirements elicitation phase of this thesis. The reason for this was to see how other companies have solved similar problems, as well as to see what technologies they used, what details are included in their visuals and what views they had implemented. After the presentations the engineers from Volvo Cars discussed the advantages and disadvantages about the solutions from their point of view. This gave valuable input for this thesis, as well as better understanding of the stakeholder preferences.

The first presentation was about an application from a company located in the United States. The meeting was held online with the presenter in America and all Volvo Cars participants gathered in a conference room. The second presentation was from a Swedish consultancy company. They had developed a tool to visualize autonomous vehicle sensor data online, as well as offline. This presentation took place at Volvo Cars with some of the developers physically present in the room.

4.3 Architecture Requirements

The first research question is "What are the functional and non-functional requirements for an architecture model to visualize data from automated vehicles testing?". To answer this question multiple research methods were used. A literature review was conducted, two focus groups were held, a survey was sent out and two similar systems were analyzed. These methods are described in section 4.1. The results of these methods were turned into requirements, that then were prioritized. The method for this process, as well as for evaluating the result is described in this section.

4.3.1 Creation and Prioritization of Requirements

The methods described in section 4.3 were used to generate lists of requirements. Each focus group, the literature review, the analysis of similar systems and the survey generated five different requirement lists. Those lists were merged and the duplicates were removed and some higher level requirements were split up in multiple requirements. Each requirement was then assigned a number in the format FRXX for functional requirements and NFRXX for non-functional requirements. Furthermore, every requirement has a name that described the requirement in one sentence and a longer rationale. After the creation of the requirements, they were prioritized in an ordinal scale with high, medium and low priorities. The priority of each requirement is assigned in three different ways, which resulted in three priorities per requirement.

To make it possible to see the different priorities by different means, they were not merged into one single priority.

The first priority was based on the main stakeholders preference at Volvo Cars. A meeting was organized with the main stakeholders at Volvo Cars and the requirements were prioritized in the specified scale. The second priority was based on the number of occurrences of the requirement in different elicitation methods. The requirements were elicited using five different methods: Focus group at Volvo Cars, Focus group at Chalmers, literature review, analysis of similar systems and survey. If a requirement was repeated in three or more methods, it was assigned high priority. If it was mentioned twice it got medium priority and if it was mentioned once it was assigned low priority. The third priority is based on the survey results. The survey answers are based on the Likert scale, with a range from one to five. Persons who answered with a four or a five were considered promoters of the feature and the percentage of promoters determined the priority. 75% or higher resulted in high priority, 50% or higher in medium priority and below 50% in low priority.

Table 2 and Table 3 give two examples how the prioritization process works. The main stakeholder priority is set by the main stakeholder directly and is high in these examples. In the first example, the occurrence priority is medium because the requirement occurred in two sources (elicitation methods). The survey priority is medium because between 50% and 75% of the participants of the survey rated the feature four or a five on the five point scale. In the second example, the occurrence is low because only one source mentioned the requirement. The survey priority is high because more than 75% of the participants of the survey rated the feature four or five.

FRXX	The Requirement.		
Priority	Main Stakeholder High	Occurrence Medium	Survey Medium
Source	Source 1, Source 2		
Rationale	Longer Description.		

Table 2: Prioritization process example 1.

FRXY	The Requirement.		
Priority	Main Stakeholder High	Occurrence Low	Survey High
Source	Source 1		
Rationale	Longer Description.		

Table 3: Prioritization process example 2.

4.3.2 Validation

Based on the elicited requirements, an architecture was created. This architecture was evaluated against the requirements together with the main stakeholders and the academic supervisors of this thesis. This was done by presenting the created architecture in the review meetings and orally explaining the architectural pattern and design decisions. The feedback from the reviews were used to improve the architecture.

4.4 Parameters and Features

The second research question is “How can the necessary parameters and features be identified and automatically extracted from the sensor data without loss of necessary information in order to generate a visual representation?” This research question shared most research methods with RQ1. The focus groups and the survey also included questions about the parameters and features that need to be included visuals. In the literature review and the analysis of similar systems, the visualization details were also taken into account. In addition, the available data was studied to learn what information was included in the sensor data and what was possible to visualize. The data study is described in this section.

A data study was performed to identify the parameters and features. This was done to learn how the input is structured and to assess how it can be modelled for use in the visualization tool. It was also important to identify the characteristics the sensors provide (for instance distance to another object, direction to the object), to be able to create a data model.

The data was provided as a .pcap (packet capture) file [23]. This file type is associated with the Wireshark network protocol analyzer [24]. The file contained UDP packages that had been sent over a network from a car to a computer for logging. The packages could be filtered on the destination port to see only packages sent from a specific sensor. The packages sent from different sensors structured the data in a different way, so they had to be decoded individually. All data was contained in large chunks of binary data and had to be extracted using its bit locations from the sensor specification.

Currently Volvo Cars extracts the data to a Matlab file using a C program. This code and the resulting .mat file were analyzed together to get better understanding of how the data is extracted. Live data is not available on the network all the time, so for testing purposes the provided .pcap file was replayed using the tcpreplay application [25], which is available for Linux. Since the development was done using a computer with the Windows operating system, Ubuntu was installed to run tcpreplay, using a virtual machine. Some properties of the .pcap file had to

be rewritten using the `tcpdump` and `tcpwrite` commands of the `tcpdump` software in order to send the packages to the correct destination address, namely the local address of the machine where the socket for listening was running. When replaying data using `tcpdump`, packages are re-sent and the receiver sees the packages as if they were coming from the original source.

4.5 Technologies

The third research question is "What existing technologies can be used to develop a tool that implements an architecture to visualize data from AV testing?". To answer this question, a literature study and an analysis of similar systems were used. This was done to identify the technologies that were used to implement similar tools. Furthermore a feasibility study was conducted, which is described in this section.

There are a lot of technologies available that can be used to implement a visualization tool for sensor data of AVs. To decide on the technology to be used for development, a feasibility study was done. First a list of possible technologies was identified that fulfilled one of the following criteria:

- The technologies were known to the students before starting to work on this thesis.
- They were mentioned by people at Volvo Cars or at Chalmers.
- They were mentioned in scientific publications that were read.
- They were returned by Google searches.

Every considered technology had to fulfill a list of must criteria to be considered for further analysis. The must criteria are as follows:

- The technology or library is being actively developed.
- The technology or library can be included in another project.
- The technology or library has sufficient documentation support.

The technologies and libraries that were still included at this point were analyzed in more detail to determine whether they can be used to fulfill the architecture requirements. It was checked which of the most important requirements that were identified in RQ1 could be fulfilled using the technology or library. For this a table was created with the requirements on one side and the technologies on the other side. Each cell in the table specifies whether the technology can satisfy the requirement or not. To identify if it is possible, the documentation of the technology was read

and tests were conducted if needed. The result of this study is shown in Table 10 and Table 11 in the results section.

4.6 Deviation between Sensors

The fourth research question is "How can the generated visuals be used to identify the deviation in the data of the same object in different sensors?". To answer this question, RQ1, RQ2 and RQ3 had to be answered first. Based on the knowledge gained, a visualization of the deviations in the data between two sensors (if any) was created. These visuals were iteratively improved using the design science approach. This approach is described in this section.

The used design science approach was influenced by the design science methodology proposed by Hevner et al. [4] and is shown in Figure 8. The knowledge base, as shown in the right part of the figure, was the result from RQ 1-3 and a data study. The environment, as given in the left part of the figure, is Volvo Cars with its employees and computer systems. Based on the knowledge and business needs, an initial version of the visual of deviation between the data of two sensors was created, as shown in the centre part of the figure. To give clarity to the user, the visuals were created with two different colors. This was validated by the main stakeholders at Volvo Cars. Based on their feedback, the solution was further improved. This process underwent several iterations that led to the final version.

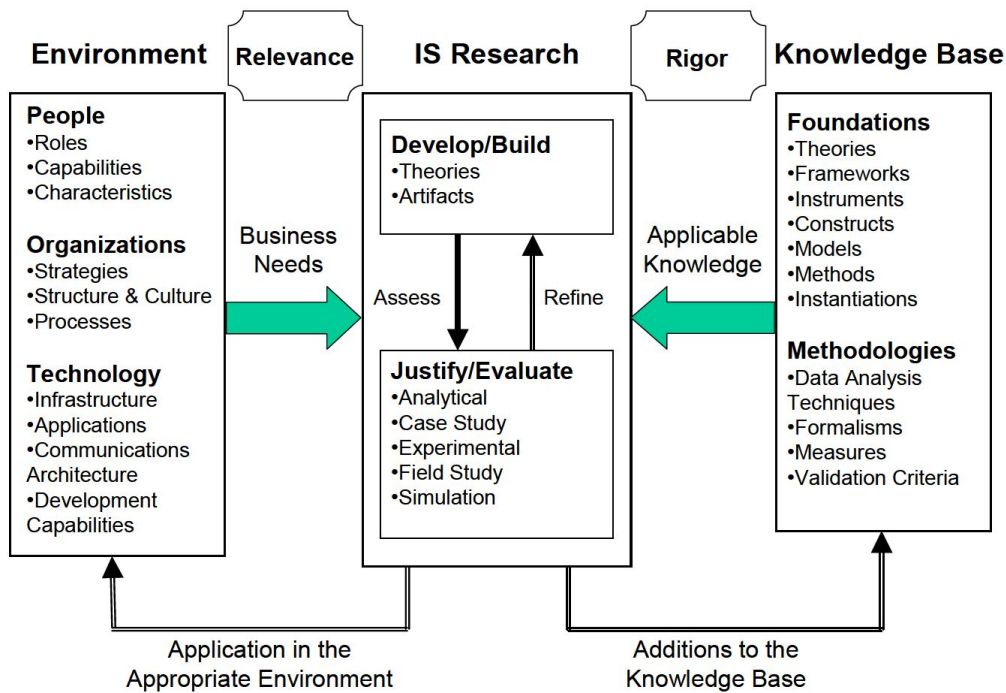


Figure 8: The design science approach as described by Hevner et al. [4]

Hevner et al. [4] proposed that the design science approach shall follow seven principles. The method used in this thesis follows all of them:

1. The research has to produce a viable artifact, in this case piece of code that visualizes deviations in sensor data.
2. The result has to solve a relevant business problem. Comparing the data of different sensors to find deviations is a real world problem at Volvo Cars.
3. The quality of the created artifact has to be evaluated. This was done by evaluating the result of each iteration with the main stakeholders.
4. The artifact has to provide a contribution to research. The approach was used to answer one research question of this thesis with a clear research goal.
5. A rigorous approach has to be used in both designing and evaluating the artifact. By thoroughly evaluating every iteration of the design with the main stakeholders, this principle was satisfied.
6. The approach is a search process for an effective artifact. The goal of both the approach and the research question behind it was to find a suitable way to represent the deviations.
7. The last principle focuses on how to communicate the result to all audiences. One of the advantages of visualization is the ability to communicate data more easily. This research followed all the seven principles.

5

Results

This chapter contains the results of the methods described in chapter 4. The results are split up per research question.

5.1 Architecture Requirements

The first research question aimed to find the functional and non-functional requirements of an architecture for a tool that visualizes the sensor data of self-driving cars. The used methods resulted in a prioritized list of requirements, multiple use case descriptions for important tasks and an architecture used for the implementation of the tool. The results are described in this section.

5.1.1 Functional Requirements

The requirement elicitation resulted in 57 functional requirements. A sample requirement is shown in Table 4. The full list of elicited functional requirements is confidential and therefore not included in this document.

FR05	The tool shall have a view that shows the car and its surroundings in bird's eye view.		
Priority	Main Stakeholder High	Occurrence High	Survey High
Source	Focus group Volvo, Focus group Chalmers, Analysis of similar systems, Literature		
Rationale	Top-down view on the car and all identified objects around the car.		

Table 4: An example of a functional requirement.

5.1.2 Non-Functional Requirements

Additional to the functional requirements, the elicitation process also resulted in 10 non-functional requirements. A sample non-functional requirement is shown in Table 5. The full list of elicited non-functional requirements is confidential and therefore not included in this document.

NFR10	The tool shall be able to run in a browser based environment.		
Priority	Main Stakeholder High	Occurrence -	Survey -
Source	Main Stakeholder		
Rationale	The tool will render the scenes using HTML5 based technology.		

Table 5: An example of a non-functional requirement.

5.1.3 Use Cases

For important requirements, use cases were generated. All the requirements were assigned priorities based on different criteria. One priority was given by the main stakeholders, one from the number of sources that mentioned the priority and one from the validation through the survey. For requirements that only got high priorities, or got high priority from the main stakeholders and medium priority otherwise, use case descriptions were created. A sample use case is shown in Table 6 and 12 use cases were generated in total. The full list of use-cases is confidential and not included in this document.

5.1.4 Architecture

The functional and non-functional requirements were used to create an architecture for a tool that visualizes sensor data of AV tests. The architecture is shown in Figure 9.

The architecture uses the Model-View-Controller pattern (MVC). In this pattern, the architecture is split into three parts. The model, that consists of both the Model component and the Input component, is responsible for the data in the tool and operations on the data. The controller is responsible for manipulating the model by sending requests to it. Finally the view is responsible for showing the data to the user. In the MVC pattern the user uses the controller to manipulate the model, which in turn updates the view, that is shown to the user. In other words, the user only interacts with the controller and only sees the view. The user never interacts with the model at all.

Task	UC2: Use offline mode
Purpose	The user wants to open a log file and view a test that happened at an earlier point in time.
Precondition	File is stored on a known location and in the right format. The user is allowed to read the file.
Subtasks	
<ol style="list-style-type: none"> 1. The user selects offline mode from the menu. 2. The tool opens a popup where the user selects if he or she wants to open a file or a folder. 3. The tool opens a file selector to pick the file to open. 4. The user selects a file having a supported filetype. 5. The tool reads the file from the hard drive. 6. The tool sends the data to the handler for the filetype of the file. 7. The tool visualizes the data returned from the handler. 8. The tool stops visualizing when it reaches the end of the data and waits for user input. 	
Variants	
<ol style="list-style-type: none"> 4a The user selects a folder. 4b The tool reads all the files having a supported file type from the folder. 4c Go to step 6. 6a Error while reading the file. Go to 3. 7a Error while handling the data. Go to 3. 	

Table 6: An example of a use case description.

The Controller component has two sub-components. The first one is the UserHandler component, that is responsible for accepting user requests. It has five interfaces, that provide different types of functionality. Every command a user (or another program) gives to the tool is sent through one of these interfaces. The UserHandler component then evaluates the input and sends it to the right interface of the model part of the tool. The second sub-component is the GUIHandler component. It provides one interface, that is used by the graphical user interface (GUI) of the tool. If, for instance, the user wants to apply a filter on the data before visualizing it, the tool has to be able to show the user a list of available filters. The GUI has to send a request to the model that keeps track of this information and get a list of all available filters.

The Model is split into two parts. The Input component part handles the input from the files and streams and the Model component part manages the data model. The InputHandler is the main component in the Input component part. It receives requests from the controller and delegates the tasks to other components through different interfaces. The IReader interface is used to read the input from files and streams. It has four components to read data from local files, UDP streams, Map APIs and Mat files. The IReader implementation sends the data to an IHandler implementation. This implementation decodes the data and extracts the necessary parameters and features. The handler sends the result to the ModelGenerator component, which creates an instance of the data model. This data model is returned to the InputHandler component, that sends it to the Model component using the IModel interface.

The ModelHandler component is the main component of the Model part and provides interfaces to the Input component and the controller. The ModelStorage component is used to keep instances of the data model at runtime. It provides interface to store, get and delete the data model instances. The FileWriter component is used to store information persistently in files. It can store configurations, settings, filters, logs, screenshots and videos. The FileReader component is used to load the stored configurations, settings and filters.

The view part is responsible for generating the visual representations of the data. The main component is the VisualHandler component, that provides the IView interface to the model for manipulating the views shown to the user. Based on the request, the handler forwards it to the respective IVisualGenerator implementation. The architecture has generators for birds eye view, text view, camera view and function view.

The logical flow in the application usually goes from the controller to the model and finally to the view. This flow is described using the examples below:

Example 1: The user starts the online mode.

The user invokes the `startOnlineMode()` function in the `IUserInput` interface of the `UserHandler` component. The component validates the input and sends the request to the `InputHandler` component using the `IInput` interface. To read the UDP stream, the `StreamReader` component is used and the read data is sent to either the `AdcanHandler` or the `SodHandler` using the `IHandler` interface, depending on the type of the data. The handlers use the `ModelGenerator` and its `IModelGenerator` interface to convert the data to the tools data model, that contains the necessary parameters and features extracted from the data.

The instance of the data model is then sent to the `ModelHandler` component using the `IModel` interface. This component stores the instance in the `ModelStorage` component using the `storeModel()` function in the `IModelStorage` interface. Based on the active views, requests are sent to the `VisualHandler` component using the `IView` interface. The `VisualHandler` forwards the information to the correct `IVisualGenerator` implementation which then uses the data to create visuals that are shown to the user.

The `StreamReader` component of the `Input` component continuously listens for new UDP packages and the received data is processed in the same way as described above until the user ends the task.

Example 2: The tool has to display all active sensors.

The tool has to show a list of active sensors to the user. It invokes the `getSensors()` function of the `IGUI` interface of the `GUIHandler` component. The handler validates the request and forwards it to the `ModelHandler` component. The `modelhandler` keeps track of the active sensors, since it receives data from the `Input` component. The `ModelHandler` component returns the list to the `GUIHandler` component, which returns it to the GUI that sent the request.

5.2 Parameters and Features

The second research question aimed to find the parameters and features to include in the visuals. This process consisted of two parts: identifying the visualization details that people expect to see in a visualization of sensor data in the context of AVs and studying the data that is sent from the sensors in order to identify the object information that is included in the data. Based on this knowledge the parameters and features to include in the visuals were decided. The results are described in this section.

5. Results

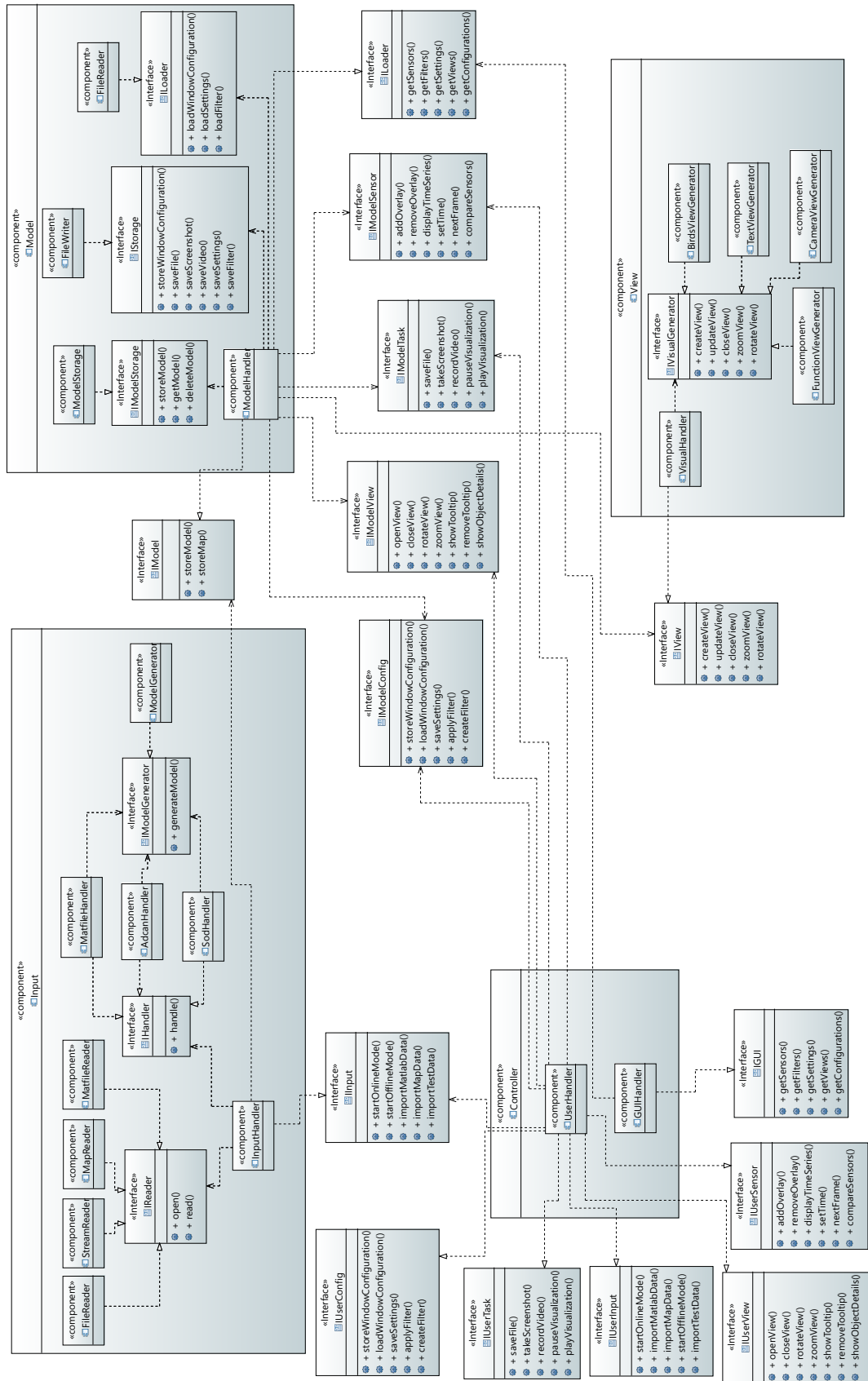


Figure 9: The created architecture.

5.2.1 Sensor Data

To be able to include a parameter or feature in the visuals, the sensors that gathers the data has to provide the necessary information. The data study provided information about what is available and technically possible to show with the considered sensors. The sensors didn't provide raw data, but already did fusion and object recognition and provided information about the sensed objects and certain properties of them.

Sensor 1: Radar and Camera with Integrated Fusion

This sensor is placed behind the rear view mirror and detects objects in its field of view in front of the vehicle, which is shaped like a cone. It consists of a radar and a camera and automatically detects objects and traffic signs by fusing information from its two parts. The general area that the sensor can detect objects in is shown in Figure 10. The exact angle and distances are confidential and are not shown in the image.

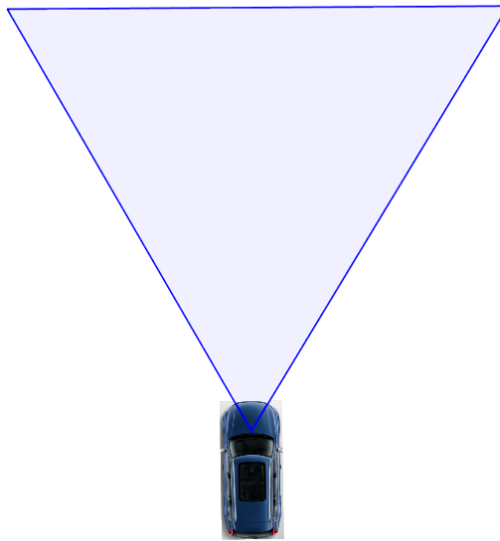


Figure 10: The area sensed by the first sensor.

The information that the sensor gathers about other objects is summarized in Table 7.

Furthermore, the sensor data includes information about identified traffic signs. This information is summarized in Table 8.

Position	The longitudinal and latitudinal position of the identified object.
Velocity	The longitudinal and latitudinal velocity of the object.
Acceleration	The longitudinal and latitudinal acceleration of the object.
Standard Deviations	Standard Deviations of the previously mentioned values.
Width	The width of the object.
Height	The height of the object.
Type	The type of the object. For instance car or pedestrian. Can be unknown.

Table 7: Object information in sensor 1.

Position	The longitudinal and latitudinal position of the traffic sign.
Type	The type of the traffic sign.
Value	The value of the traffic sign.
Location	Information about if the traffic sign is above the road or next to it.
Lane	The lane the traffic sign is above. Only valid if the traffic sign is above a lane.
Type1	The type of the first supplementary sign.
Type2	The type of the second supplementary sign.

Table 8: Traffic sign information in sensor 1.**Sensor 2: 360 Degree Radar**

This sensor consists of four radars, placed on each of the four corners of the car. By fusing information from the left two and the right two, object information is gathered in its field of view, which is shaped like a circle around the car. The area covered by the sensors is shown in Figure 11. The red boxes in the figure show the approximate sensor placement. The exact radius, in which detections are made, is confidential and not shown the figure.

The information that the sensor gathers about other objects is summarized in Table 9.

Position	The longitudinal and latitudinal position of the identified object.
Velocity	The longitudinal and latitudinal velocity of the object.
Acceleration	The longitudinal and latitudinal acceleration of the object.
Length	The length of the object.
Width	The width of the object.
Type	The type of the object.
Existence probability	The probability that the object really exists.

Table 9: Object information in sensor 2.

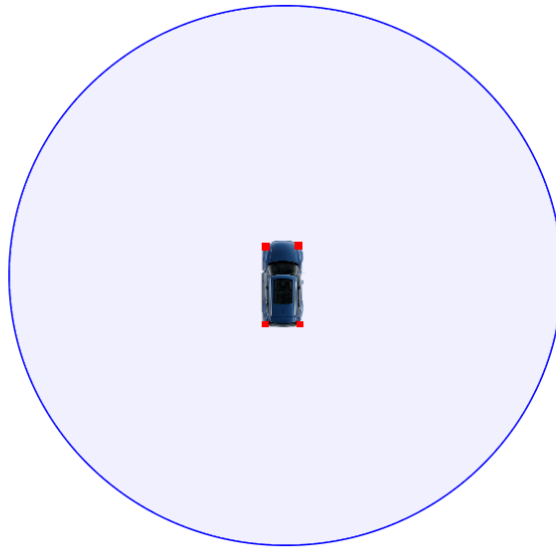


Figure 11: The area sensed by the second sensor.

5.2.2 Identified Parameters and Features

Based on the methods described in section 4.2 and 4.4, parameters and features to be included in the visuals were identified. The following were considered the most important by the study:

- Own car (including it's GPS position)
- Lane markings (including road edges)
- Other vehicles on the road
- Parked vehicles
- Pedestrians
- Speed of the vehicles
- Static objects not on the road
- Traffic signs

5.2.3 Included Parameters and Features

The information included in the data and the wishes of the engineers and researchers that participated in the survey and the focus groups were used to decide what parameters and features to include in the visuals generated by the tool. Most participants agreed that not all information should be shown in the visual itself, because it would make them very cluttered. Instead the user should have the possibility to click on an object in the birds eye view to show additional information about it. The following information is included in the visuals, either directly in the scene or in a separate text view:

- Own car (including it's GPS position)
- Other vehicles on the road
- Other objects on and near the road
- Position, speed and acceleration of objects
- Width, height and length of objects
- Type of the object
- Existence probability
- Lane markings

5.3 Technologies

The third research question aimed to find the suitable technologies to implement a tool using the created architecture. Multiple technologies were available to use, but it was important to find the most suitable one. The results are presented in this section.

5.3.1 Feasibility Study

The feasibility study was performed as discussed in section 4.5 A table was created using the technology dependent requirements in the rows and the technologies, that satisfy the selection criteria as given in section 4.5 in the columns. Each entry in the table represents whether the technology in that column can be used to implement

the requirement in its row. The result of this study is given in Table 10 and Table 11.

In the feasibility study, two of the requirements were merged to focus more on the technical background of the requirements. In order to input external data into the tool, or allow it to accept plugins, the tool has to be able to read local files. The study contains a requirement about reading files instead of the two mentioned requirements.

A total of 8 technologies and 18 requirements were included in the study. The following technologies, that fulfilled the conditions from section 4.5, were included in the study:

Three.js

Three.js is a lightweight Javascript library that can be used for creating WebGL graphics [26]. Using this library results in fewer lines of code than when using pure WebGL.

X3D

X3D is a file format and runtime environment used for representing 3D scenes. The objects and actions on the objects are stored as XML tags [27]. Through X3DOM the scenes can be integrated directly into a HTML page [28].

Node.js

Node.js is a Javascript runtime environment that is built for scalable network applications [29]. It provides additional functionality that is not available in "pure" Javascript in the areas of networking and file system access.

Unity

Unity is a game engine that can also be used for visualization purposes. The unity editor can be used to create applications for many platforms using Javascript or C# as programming languages [30].

Unreal Engine

Unreal Engine is another game engine that can be used for visualization [31]. As with Unity, the resulting application can be deployed on multiple platforms.

Java3D

The Java3D API allows a programmer to create 3D scenes in web-based Java applets [32]. The API provides a high level of abstraction for creating and manipulating 3D objects.

VTK

VTK (The Visualization ToolKit) is a library written in C++ that can be used for image processing and visualization. [33] It provides many built-in algorithms for visualization and modelling.

Qt

Qt is a framework of libraries that enables easy cross platform application development [34]. It aims to simplify writing code and developing for multiple screens.

	Three.js	X3D	Node.js	Unity	Unreal	Java3D	VTK	QT
Listen to UDP stream	No [35]	No [35]	Yes [36]	No [37, 38] N1	Yes [39] N1	Yes [40] N2	No [41, 42] N3	Yes [43]
Read files	Yes [44] N4	Yes [44] N4	Yes [45]	Yes [46]	Yes [47]	Yes [48]	Yes [49] N5	Yes [49] N5
Undock a view	Yes [50] N6	Yes [50] N6	Yes [50] N6	No N7	No N7	Yes [51]	Yes [52] N8	Yes [53]
Display multiple views	Yes [50] N6	Yes [50] N6	Yes [50] N6	Yes [54] N9	No N10	Yes [51]	Yes [55]	Yes [56]
Take screenshots	Yes [57, 58] N11	Yes [59]	Yes N12	Yes [60]	Yes [61]	Yes [62] N2	Yes [63]	Yes [64]
Record video	Yes [65] N13	Yes [65] N13	Yes N12	No [66, 67] N14	Yes [68]	Yes [69]	Yes [70] N15	Yes [71] N15
Arguments support	Yes [72] N16	Yes [72] N16	Yes [73]	Yes [74]	Yes [75]	Yes [76] N2	Yes [77] N5	Yes [78]
Rotate the view	Yes [79, 80]	Yes [81]	Yes N17	Yes [82]	Yes [83]	Yes [84]	Yes [85]	Yes [86]
Zoom the view	Yes [87]	Yes [88]	Yes N17	Yes [82]	Yes [83]	Yes [84]	Yes [85]	Yes [86]
Write file	No [89]	No [89]	Yes [90]	Yes [91]	Yes [92]	Yes [93] N2	Yes [49] N5	Yes [49] N5
RAM as a buffer	No [94] N18	No [94] N18	No [95, 96] N19	No [97]	Yes [98]	No [99]	Yes [100] N5	Yes [100] N5

Table 10: Results of the Feasibility Study on functional requirements.

Sometimes, it was not possible to answer if the requirement could be fulfilled using a technology by a simple yes or no. In those cases, additional notes were added. The notes are displayed as NXX in the table. The explanations can be found below:

	Three.js	X3D	Node.js	Unity	Unreal	Java3D	VTK	QT
Large Point cloud	Yes [101] N20	Yes [101] N21	Yes N22	No N23	No N24	No N25	Yes [102]	Yes [102]
Run on Windows PC	Yes [103, 104]	Yes [105, 104]	Yes [106]	Yes [107]	Yes [108]	Yes [109]	Yes [42]	Yes [110]
Run on Apple OSX	Yes [103, 104]	Yes [105, 104]	Yes [106]	Yes [107]	Yes [108]	Yes [109]	Yes [42]	Yes [110]
Run on Linux Ubuntu	Yes [103, 104]	Yes [105, 104]	Yes [106]	Yes [107]	Yes [108]	Yes [109]	Yes [42]	Yes [110]
Run on Android	Yes [103, 111]	Yes [105, 111]	No [106]	Yes [107]	Yes [108]	No [109]	No [42]	Yes [110]
Run on iOS	Yes [103, 112]	Yes [105, 112]	No [106]	Yes [107]	Yes [108]	No [109]	Yes [42]	Yes [110]
Run in browser	Yes [103]	Yes [105]	Yes [106]	Yes [107] N26	Yes [108] N26	No [109]	No [42]	No [110]

Table 11: Results of the Feasibility Study on non-functional requirements.

1. When developing for a web browser, UDP sockets are unavailable due to security reasons. When developing for a desktop operating system, it is possible to use UDP sockets.
2. Java3D is an API for Java, so all functionality provided in the Java language can be used.
3. VTK can be integrated in languages like C++ or Java, so UDP sockets can be created in the host language.
4. For security reasons, reading files automatically is not possible. The user has to manually select the file using a file picker.
5. VTK and Qt support implementation in C++, so reading and writing files, allocating memory and parsing command line arguments can be implemented using C++.
6. The rendering will be done in a browser window, and the window isn't docked by default. It is also possible to open multiple browser windows.
7. Undocking windows is not officially supported in either Unity or Unreal Engine. It might be possible by rewriting parts of the game engine.
8. Integration of VTK with Qt is possible. Undocking windows can be done using Qt.
9. Multiple cameras can be used and the "normalized viewport rectangle" can be set for each camera to cover only a part of the screen. Multiple views can be rendered by specifying different parts for the different cameras.
10. It is possible to create a split screen "game". By doing this every view has the same size and the number of views are fixed.
11. The three.js renderer has a property called `domElement` and it has a method called `toDataURL` to take screenshots.
12. Node.js is a server-side application and can't do the visualization itself. By using a Javascript library like three.js or X3D, it is possible to take screenshots and record videos.
13. Three.js and X3D are built on top of WebGL, and using WebGL it is possible to record a video.
14. There is a plugin called "AVPro Movie Capture" available for buying for Unity to record videos. Other external tools can be used as well.

15. Video capturing is not possible. Multiple screenshots can be made and then put together to make a video.
16. The arguments can be added at the end of the url as parameters, and will be processed similar as http GET request.
17. Node.js uses WebGL to render the data. Rotating and zooming is possible using both Three.js and X3D.
18. Javascript based solutions dynamically allocate memory when a new value is initialized. There is no built-in function to allocate memory, like in C or C++.
19. Node.js uses the V8 Javascript engine. V8 handles its memory itself and it is not possible to manually allocate or free memory.
20. Potree is a library based on Three.js that can be used to render point clouds.
21. Potree is a library based on Three.js, which can be integrated with X3D to render point clouds.
22. Node.js uses WebGL to render the data. Since both considered Javascript WebGL libraries support point cloud rendering, node.js supports it as well.
23. There is no official point cloud support for Unity. Some free and paid third party plugins are available.
24. There is no official point cloud support for Unreal Engine, but it is possible to write a point cloud renderer.
25. Java3D does not support point clouds.
26. When creating WebGL code that runs in a browser based environment, some limitations are introduced. It is no longer possible to listen to a UDP stream or reading arbitrary local files, for security reasons.

5.3.2 Chosen Technology

Based on the result of the feasibility study and the prioritized requirements, a choice of technology for the implementation was made. One of the important stakeholder requirements was that the tool would run in a browser based environment. This requirement limits the number of possible technologies.

The tool was implemented using nw.js, which is a framework that bundles Node.js with a Chromium browser. Three.js was used for the 3D visualization and Javascript

was used as programming language. Furthermore, HTML5 and CSS3 was used for generating the frontend. A longer discussion of why this is the best choice for the needs of this project can be found in chapter 6.

5.4 Implementation

After the requirements had been elicited and the technologies were chosen, the implementation started. This section describes the layout of the tool and the implemented requirements.

Main Window

The main window is shown when the user starts the tool and it consists of three segments. The first segment is used to display the tool's status information: "Playing Online" or "Playing Offline", the "Data Source", which shows the path of the source and the name of the sensor. The user can turn the visualization of the data of a sensor "ON" or "OFF" at any point in time.

The second segment consists of a series of 14 buttons, through which a user can use the different features of the tool. The "Online Mode" button is used to turn on the online mode and visualize the data received as UDP streams. The "offline Mode" is used to turn on the offline mode and visualize the data in the file in .pcap format. The "Close Data Source" button can be used to close the current data source of the visualization and select a new one. The other buttons in this segment are explained in the subsequent paragraphs.

The third segment consists of a scroll bar which shows the status of the visualization in terms of time. The slider can also be used to jump to a specific point in time of the visualization. It also has buttons to pause, play, stop, replay and go forward or backward in steps of time (50ms). The user can use the screenshot and record buttons to take screenshot and record video of the visualization. A play live button is available, which will be active only in the online mode. In the online mode, if the user pauses the visualization, the tool will stop visualizing but still stores any new data arriving. When the user clicks the play button, the tool will start playing from the time it was paused. If the play live button is clicked, it will visualize the live data. The forward seek button will be inactive in the online mode as long as it visualizes live data. If the user pauses the visualization or seek backwards in the data, it will become active. The main window is shown in Figure 12.

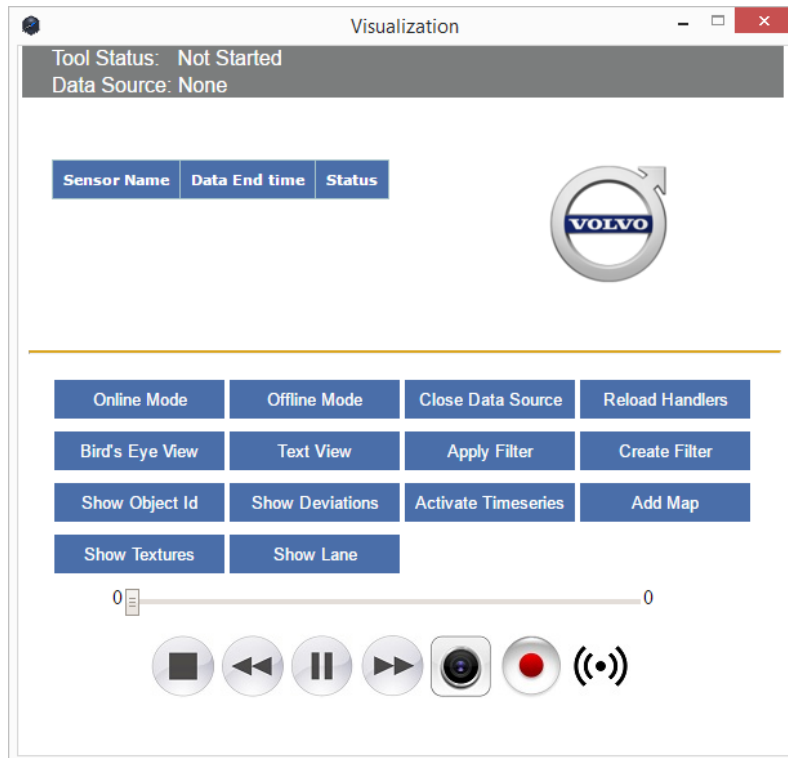


Figure 12: The main window of the application.

Bird's Eye View

This window can be opened by clicking the “Bird Eye View” button in the main screen. In this window the tool visualizes the data as 3D objects. The bird’s eye view don’t only visualize the objects from a top-down angle, but it is also possible to move and rotate the camera freely. In this tool, the bird’s eye view and the 3D view are merged. Initially, when the window is opened it will show the own car, in which the sensors are mounted, on a plane with a grid, as well as a sky box. Once the user starts the online or offline mode by clicking the respective buttons in the main window, the tool visualizes the data as 3D objects. The user can zoom in or out and also rotate the camera to any desired angle. The user can also see the current time of the visualization at the top left corner of the window. The tool uses different colors for different objects, based on their types specified in the data. A screenshot of the bird’s eye view is shown in Figure 13.

Text View

This window can be opened by clicking the “Text View” button in the main window. Here the user can view the parameters, like position, acceleration, velocity, timestamp etc., of the objects that are in the current frame. This view will be updated frame by frame as the objects are updated in the bird’s eye view. To see

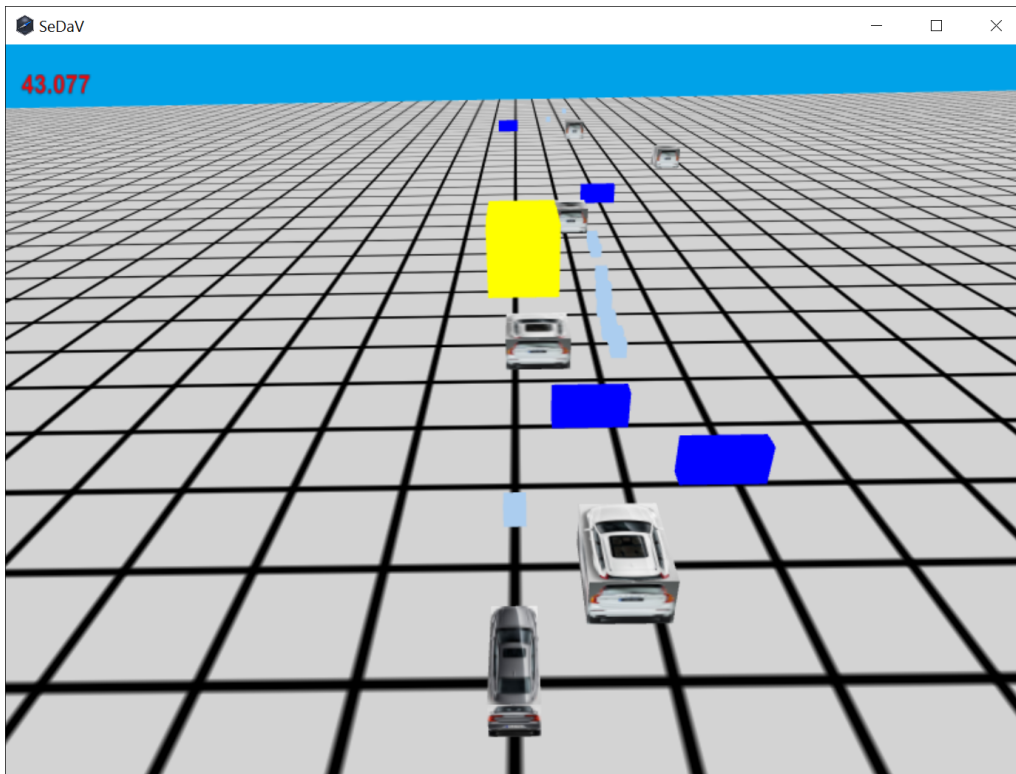


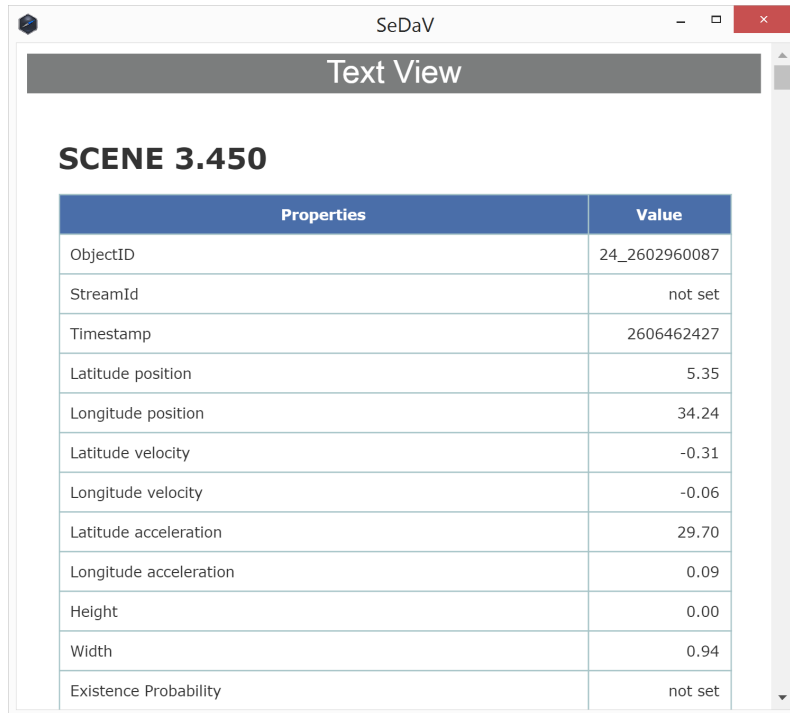
Figure 13: The bird’s eye view of the visualization.

the text data of a specific object, the user can click on that object in the bird’s eye view. A screenshot of the text view is shown in the Figure 14. Each frame will be associated with a scene id and the details of the object instances present in that frame. The tool visualizes the data from two sensors and each sensor records a list of parameters. The list mostly contains the same parameters, as well as some parameters unique to the sensor. The “StreamId” and “Existence Probability” are not available in one of the sensors data, so the value is set to "not set" when data from that sensor is shown. Furthermore, the “Height” of the objects is not available in the other sensors data, and is set to “0.00” in the text view for that sensor.

Time Series

The user can activate the time series by clicking the “Activate Time Series” button in the main window. After activating, the user can click on an object in the bird’s eye view for which the time series should be shown. The user can deactivate the time series by clicking on the “Deactivate Time Series” button in the main window. The time series of a car is shown as a blue line in the bird’s eye view of the visualization, as shown in Figure 15.

5. Results



The screenshot shows a window titled "SeDaV" with a "Text View" header. Below the header, the text "SCENE 3.450" is displayed. A table with two columns, "Properties" and "Value", lists various attributes of the scene. The table data is as follows:

Properties	Value
ObjectID	24_2602960087
StreamId	not set
Timestamp	2606462427
Latitude position	5.35
Longitude position	34.24
Latitude velocity	-0.31
Longitude velocity	-0.06
Latitude acceleration	29.70
Longitude acceleration	0.09
Height	0.00
Width	0.94
Existence Probability	not set

Figure 14: The text view of the visualization.

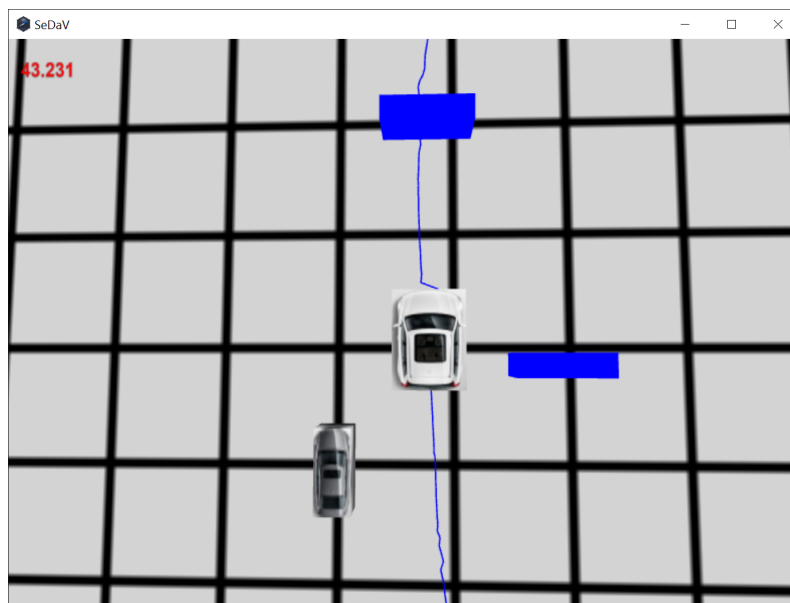


Figure 15: The time series view of an object.

Show Object Ids

Each object in the bird's eye view will be associated with a unique id. If the user clicks on the "Show IDs" button in the main window, the objects will be visualized in the bird's eye view along their id's. The id of one sensors data starts with "AD" and another starts with "SOD". A screenshot of the objects along with their id's is shown in Figure 16.

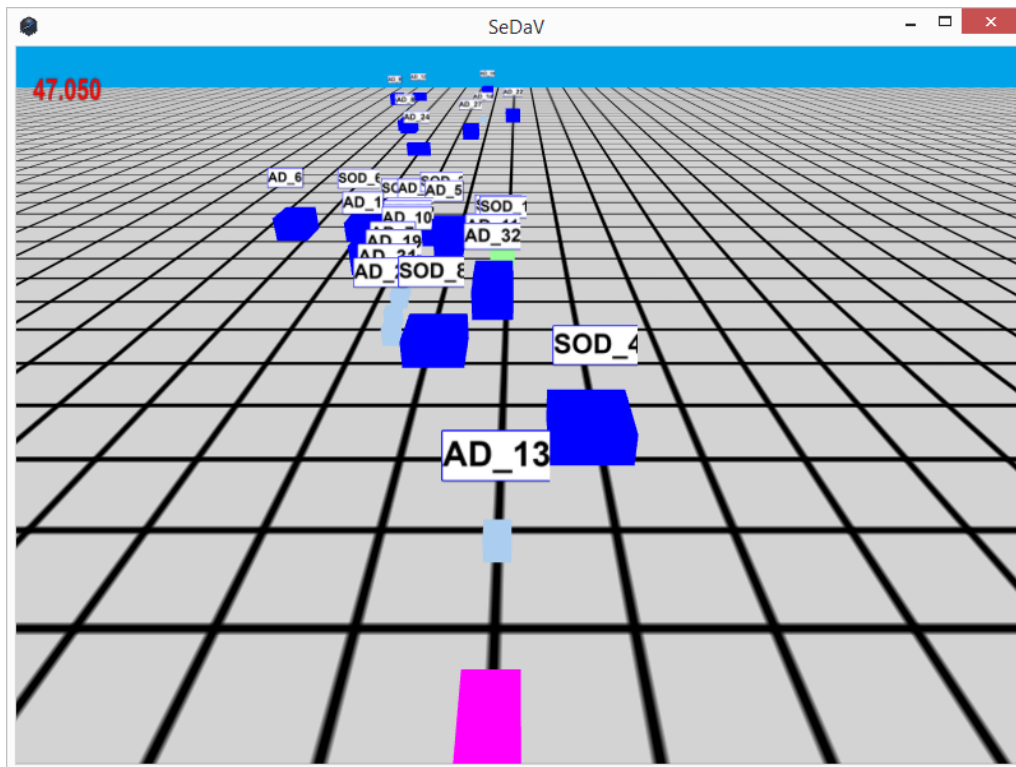


Figure 16: The objects along with its ID in the visual.

Add Map, Textures and Lane

The "Add Map" button is a toggle button, which can be used to turn on or off the Google map, over which the tool visualizes the data. The "Show Textures" button is also a toggle button, which can be used to show and hide textures. The own car is shown with the texture of the Volvo S90 car and other cars with the texture of Volvo XC90. The "Show Lane" button is also a toggle button to show and hide the lane in which the own car is driving. The lane is added as red colored curves on both sides of the own car. A screenshot of the bird's eye view showing map, texture and lane is shown in Figure 17.

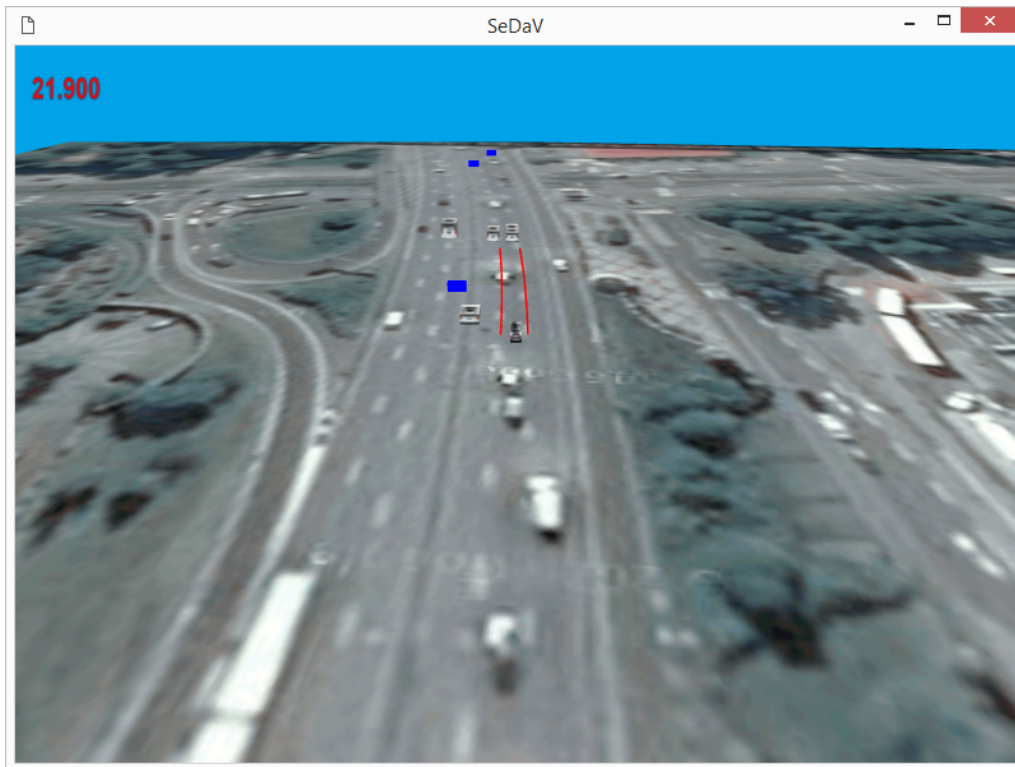


Figure 17: The bird's eye view showing map, lane and textures.

Apply Filter

The user can select this feature by clicking the “Apply Filter” button in the main window. When clicked, a window with available filters will be displayed. The user can apply and remove the filters at any point of time during the visualization. Currently the user can filter “cars only” to see only the cars in the visualization, “Positive Latitude” to see the objects on the right side of the own car and "Negative Longitude" to only see the objects behind the own car. A screenshot of the window is shown in Figure 18.

Create Filter

The user can use this feature by clicking the “Create Filter” button in the main window. A filter can be created by providing the filter name and other properties specified in the window. It is not necessary to provide all the properties listed. Instead the user can create a filter for whichever of the properties that are desired. The created filter will be shown in the “Apply Filter” window, from where the user can use it. A screenshot of the window is shown in Figure 19.

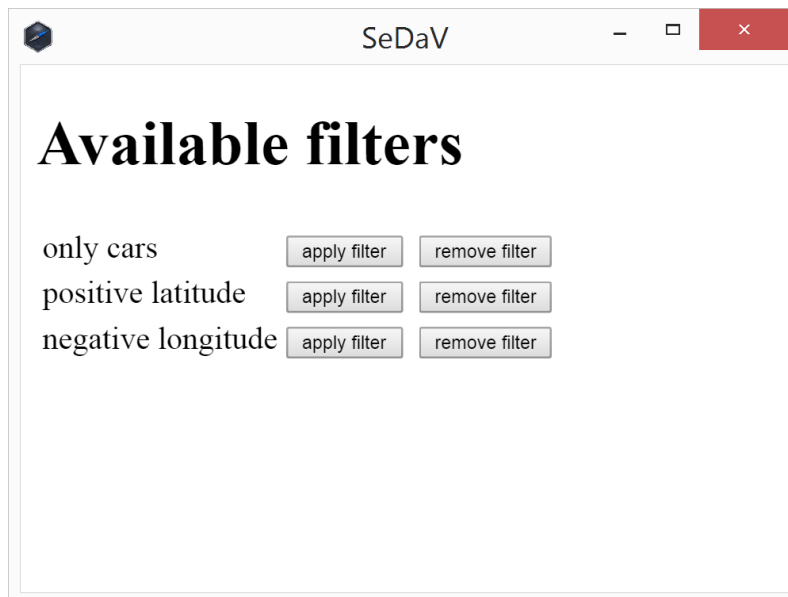


Figure 18: Apply filter window.

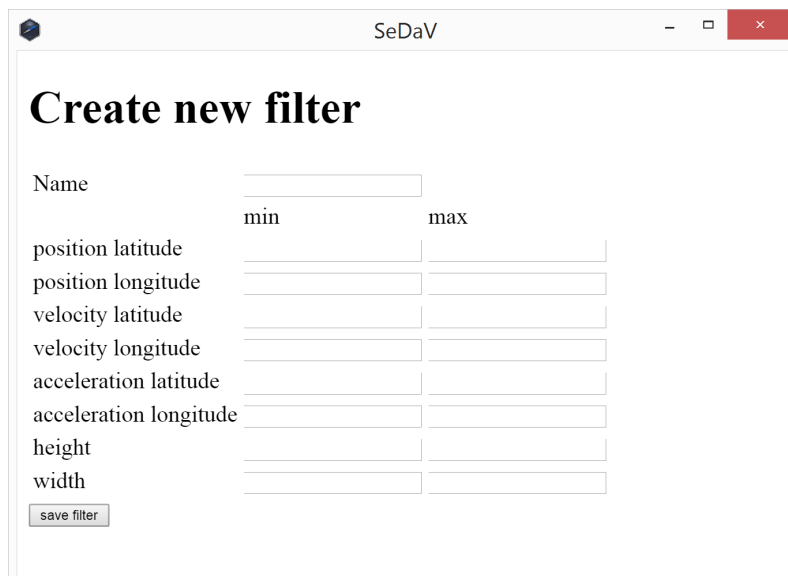


Figure 19: Create filter window.

Reload Handlers

Each type of sensor has its own handler, which is responsible for extracting information from the sensor's data and converting it to the tools data model. The sensor specific information is stored in XML files and each sensor type has its own generator to generate its handler from the XML file. If there are any changes in the sensor specific information, the user can update the corresponding XML file and click the "Reload Handlers" button, which will regenerate the handlers. This makes it easy for the user to update the handlers without having to change the code itself.

Show Deviations

This feature helps the user to compare the data from multiple sensors. The tool handles the data from two different sensor types, whose field of view overlaps. This means that the two sensors can capture data of same object. This feature automatically identifies such objects and highlights them with red color for one sensor and black for the other. Once activated, the user can use the same button to hide the deviations again. This feature is described in more detail in section 5.4.1.

Implemented Requirements

This list summarizes the requirements implemented in this tool.

- The tool has an online mode, which listens to UDP streams on a specified port number.
- The tool has an offline mode, which reads the UDP stream data from a file in .pcap format.
- The tool can visualize the data of multiple sensors at the same time in the coordinate system of the own car.
- The tool can show combination of multiple views at the same time.
- The tool can show the car and its surroundings in bird's eye view.
- The tool can show the properties of the objects in the sensor data in text form, for instance, type, position etc.
- The user can change the size of the views.
- The user can undock any view in the tool.

- The tool can display the time series of an object.
- The tool can visualize the data at a particular timestamp.
- The user can scroll through time in the offline mode.
- The tool can visualize the difference by comparing the data between a sensor and the reference sensor.
- The tool can show the ID of the objects in the view.
- The tool can show the visuals over a map.
- The tool can show the lane in which the own car is driving.
- The user can click on an object in the visual and view the details of the object in the text form.
- The tool can listen to UDP stream for the input in the online mode.
- The tool can replay the data captured in the online mode.
- When replaying data in online mode, the tool can also jump to visualize the live data.
- The user can take screenshots of the bird's eye view in different camera angles.
- The user can record a video of the bird's eye view.
- The user can call the viewer with arguments to open a file and select a start and end time.
- The tool can play back the data in the memory while still recording real time data.
- The tool visualizes objects in different color in the compare sensor mode.
- The user can rotate the bird's eye view.
- The user can zoom the bird's eye view.
- The user can apply filters to see objects with certain properties and focus on what they want to analyse.
- The user can create their own filters based on the available parameters in the data.

- The user can pause the visualization in the offline and online mode.
- The tool can jump to a specific point in time of the visualization.
- The user can to go forward and backward in time frame by frame(in steps of 50ms).
- The tool can show the list of active sensors including the time when the last package was received.
- The tool can visualize the data from new sensors, if the handlers to read the data are provided.
- The online and offline mode have the same look.
- Scrolling forward and backward in time is equally fast.
- The tool runs on a Windows PC.
- The tool runs on chromium web browser bundled along with the nw.js tool.

5.5 Deviation between Sensors

The fourth research question aimed to find a good way to visualize the deviations in the data of different sensors measuring the same object. As described in chapter 4, the design science approach was used to create a solution that then was iteratively improved. The results of the different iterations are described in this section.

5.5.1 Iteration 1

To visualize the deviations in the data of two sensors of the same object, a decision has to be made what conditions have to be fulfilled for two objects to be counted as the same object. In the first iteration, all objects whose center point where no more than two meters apart in latitude and longitude direction were treated as the same object. The two meters were based on the average size of objects identified by the sensors. To visualize the deviations, one object was rendered as a red box, while the other was rendered as a black box. The boxes overlapped in the visual, making the deviation between them clear. An example is shown in Figure 20.

The approach was validated with the main stakeholders at Volvo Cars and they generally liked the idea. They didn't want to depend on an algorithm to decide

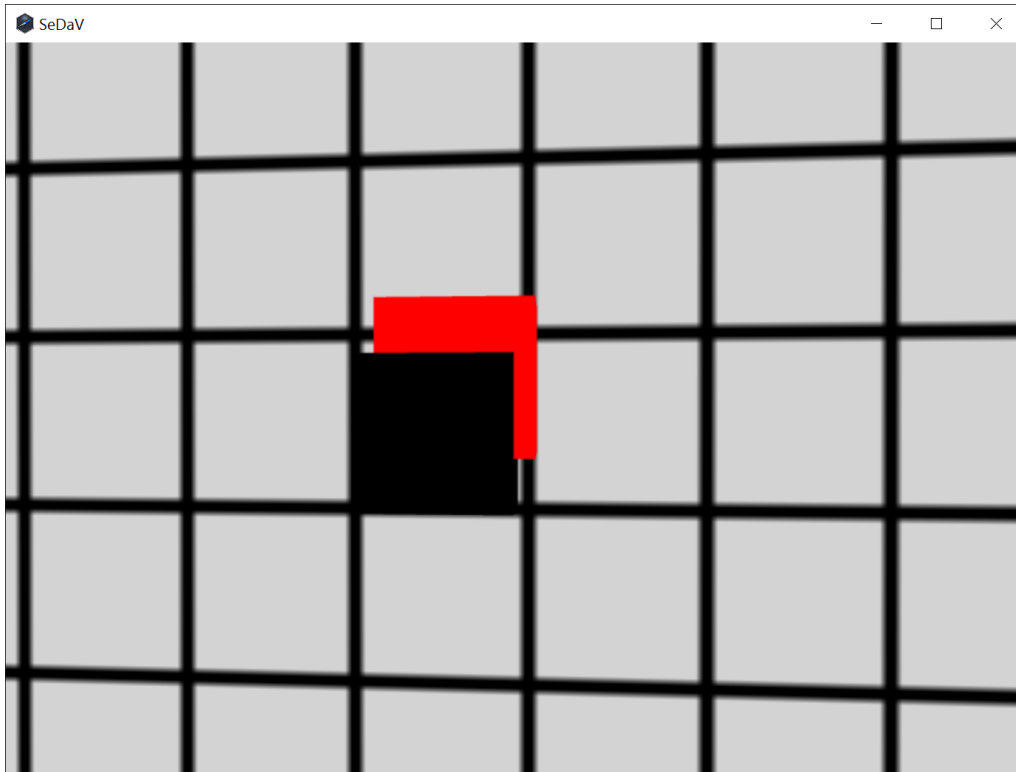


Figure 20: The same object recognized by two different sensors. Iteration 1.

which objects to compare and wanted a way for the user to select the objects. This idea was used as an input for iteration 2.

5.5.2 Iteration 2

In iteration 2 the main goal was to enable the user to select two or more objects and show their data in a separate view to compare them. The tool already had the possibility to select an object in the view to show its details and this was used as basis for implementing the second iteration. In addition to left-clicking on an object to select it, the user was made able to right-click on one or more objects to select them. As soon as one object was selected with left-click and one or more objects were selected with right-click, the text view changes to a table with all the objects on one side, and all data about the objects (position, speed, etc.) on the other side. The data in this table is then updated every time a new scene is rendered. The user can exit the compare sensor mode and go back to the normal text view by clicking the "Unselect Comparison Objects" button.

To make it clearer what objects are being compared, the colors of the objects in the visuals change when this comparing mode is active. The "main" object, that is selected with a left-click, becomes red while all objects selected with a right-click

5. Results

becomes black. This way it is easy to follow the selected objects as they move through the scene.

The visualization in the bird's eye view is shown in Figure 21 and the text representation of this comparison is shown in Figure 22.

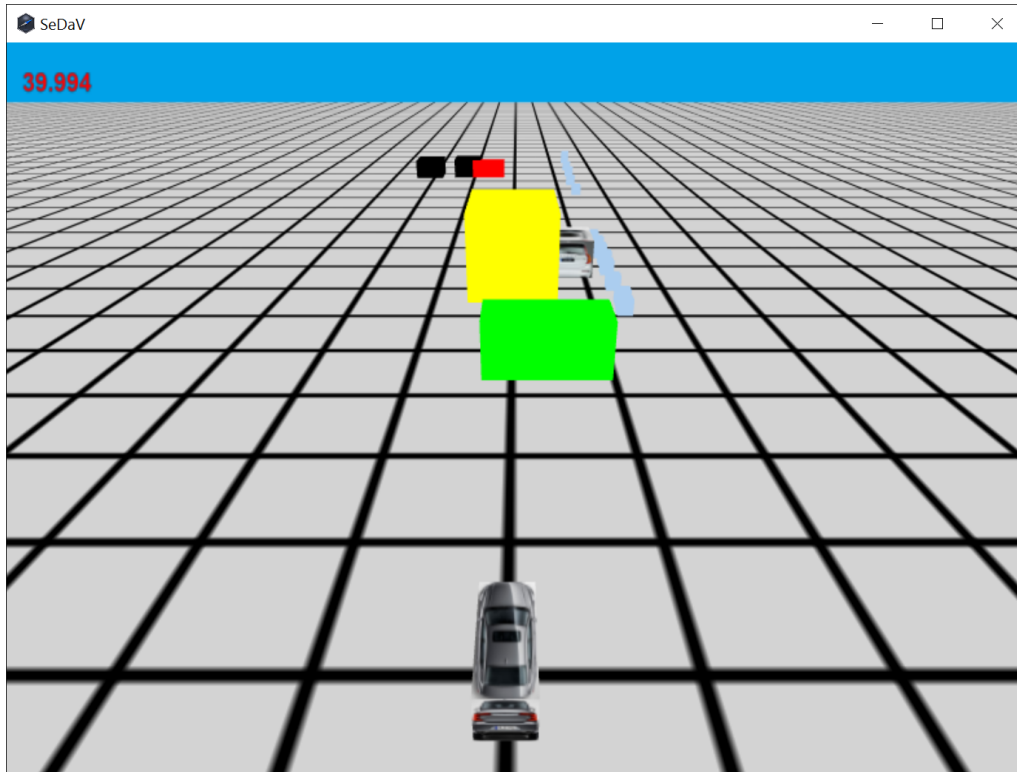
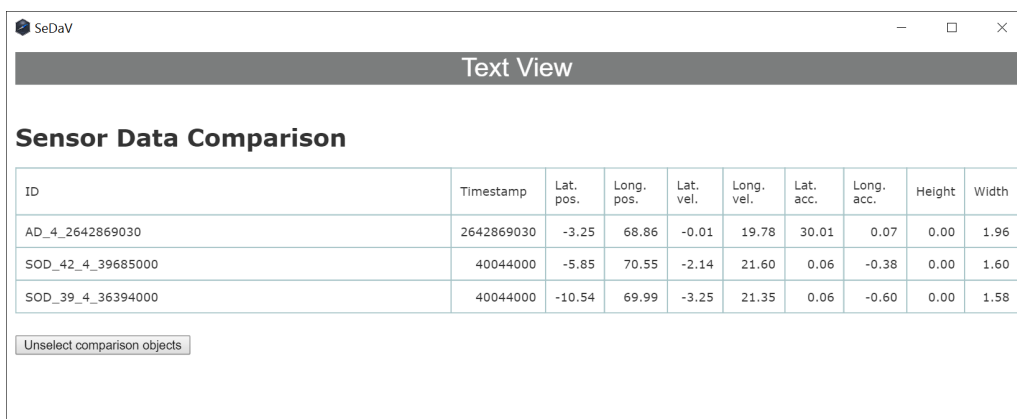


Figure 21: Comparison of different objects in the scene. Iteration 2.



ID	Timestamp	Lat. pos.	Long. pos.	Lat. vel.	Long. vel.	Lat. acc.	Long. acc.	Height	Width
AD_4_2642869030	2642869030	-3.25	68.86	-0.01	19.78	30.01	0.07	0.00	1.96
SOD_42_4_39685000	40044000	-5.85	70.55	-2.14	21.60	0.06	-0.38	0.00	1.60
SOD_39_4_36394000	40044000	-10.54	69.99	-3.25	21.35	0.06	-0.60	0.00	1.58

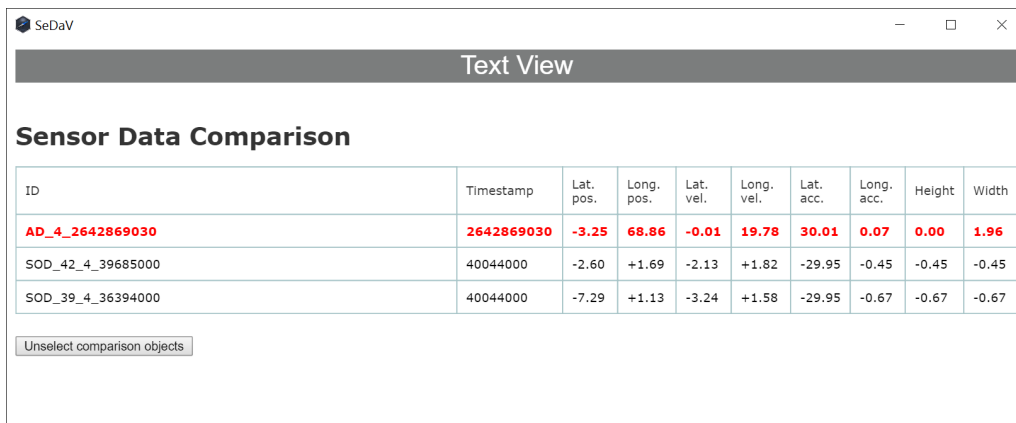
Figure 22: Comparison of different objects in the text view. Iteration 2.

The second approach was again validated with the main stakeholders at Volvo Cars. The tool showed the absolute values of all selected objects, but the stakeholders thought that it would be better to only show the absolute values of the main selected object (the one that had been chosen by a left click) and values relative to the main selected object for all other objects (the ones selected by right click). Furthermore

it should be clearly visible in the text representation which one of the objects is the main selected object. This feedback was used as input for iteration 3.

5.5.3 Iteration 3

As stated before, the main idea of this iteration was to make the information in the textual representation relative to the main selected object. The updated text view displays the difference of its own values to the absolute values of the main selected object. Furthermore, the row describing the main object is now red, while the text about the other objects is black. These are the same colors that are used in the visual itself. The visual representation, that has not changed since iteration 2, can be seen in Figure 21. The updated textual representation can be seen in Figure 23.



ID	Timestamp	Lat. pos.	Long. pos.	Lat. vel.	Long. vel.	Lat. acc.	Long. acc.	Height	Width
AD_4_2642869030	2642869030	-3.25	68.86	-0.01	19.78	30.01	0.07	0.00	1.96
SOD_42_4_39685000	40044000	-2.60	+1.69	-2.13	+1.82	-29.95	-0.45	-0.45	-0.45
SOD_39_4_36394000	40044000	-7.29	+1.13	-3.24	+1.58	-29.95	-0.67	-0.67	-0.67

Unselect comparison objects

Figure 23: Comparison of different objects in the text view. Iteration 3.

The implementation was once more validated with the stakeholders at Volvo Cars. They were satisfied with the solution, however, the comparison table was lacking the type of the object, which was added as a small fix. In a later review meeting, one of the main stakeholder commented that the objects should be semi-transparent, instead of having a solid color. This was also implemented as a small fix and a screenshot of this is shown in Figure 24.

Since the stakeholders were satisfied with the solution and didn't propose any additional functionality, it was decided to end the process after three iterations. Additional ideas how the comparison of sensor data can be improved in future is included in chapter 7 of this thesis.

5.6 Validation

The developed tool was validated with 7 different users in the Sensor Verification and Analysis group of Volvo Cars. Initially, a demonstration of the various functionalities

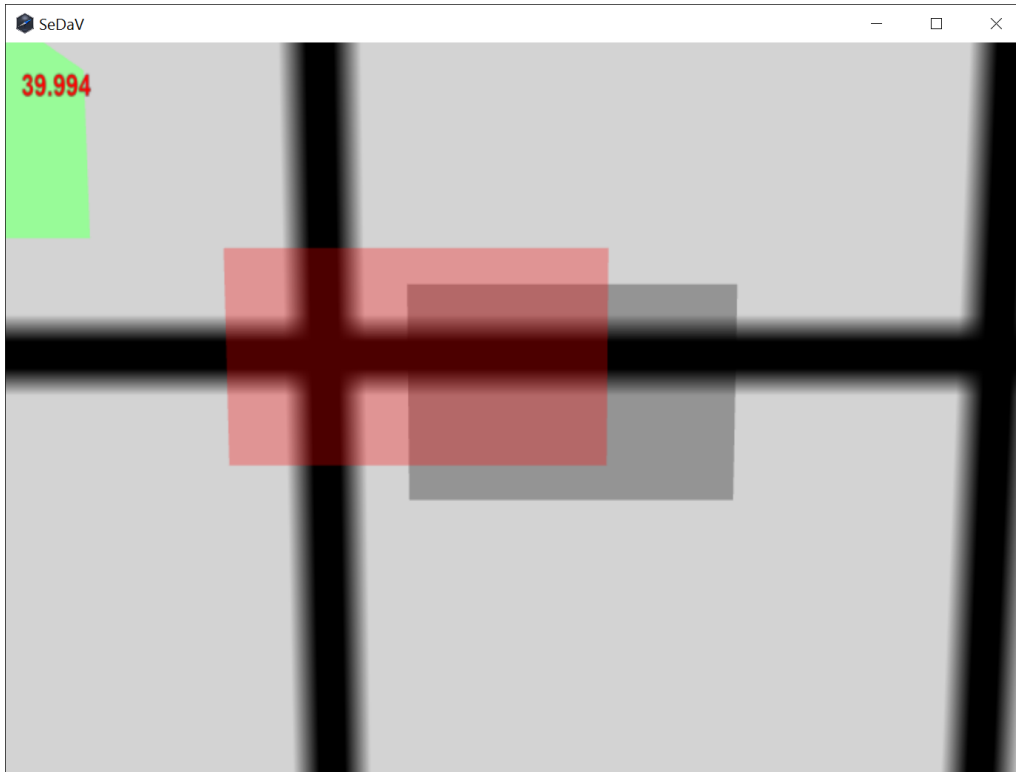


Figure 24: Comparison of different objects with transparent color.

of the tool was given to individual users. Some of the users also tried some hands-on themselves with the tool. Once the demo was finished, they were asked to answer 6 questions. The first four questions used a 1-10 likert scale, where 1 was the worst score and 10 the best. The last two questions were used to get additional feedback.

The first question was "How well does the tool fit your analyzing needs?" The average number of points was 7.71. Some users noted that some features were missing, which reduced the score. All the responses can be seen in Figure 25.

How well does the tool fit your analyzing needs? (7 responses)

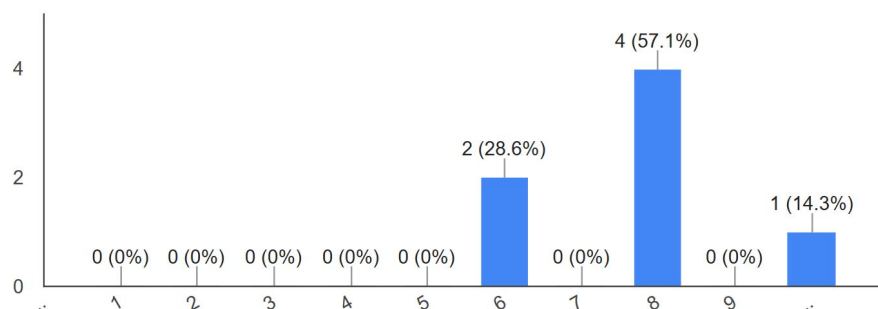


Figure 25: Validation question 1.

The second question was "How clear is it what the objects in the visual represent". This question aimed to learn if visualizing the objects as cubes with different colors and sizes was a good way of representing the data. The average number of points was 7.43. The users mainly complained about the missing legend, which resulted them in not knowing what a cube of a specific color represents. All the responses can be seen in Figure 26.

How clear is it what the objects in the visual represent? (7 responses)

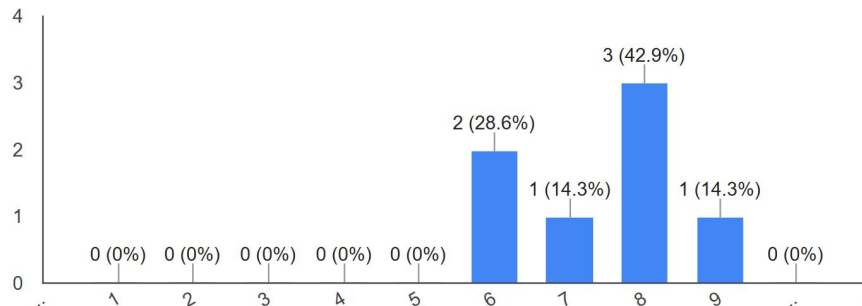


Figure 26: Validation question 2.

The third question was "How well are deviations in sensor data visualized", this question aims to find whether the users were able to see the deviations in the data of an object sensed by different sensors. The average number of points was 8.14. Generally most users appreciated this feature and one user asked to have a comparison with the own car data, instead of having to select another object. All the responses can be seen in Figure 27.

How well are deviations in sensor data visualized? (7 responses)

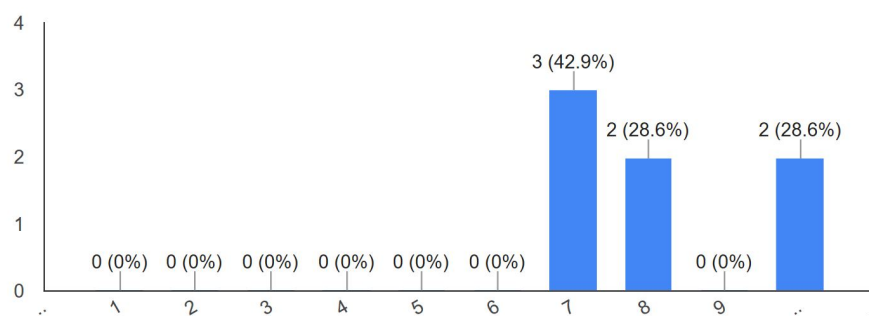


Figure 27: Validation question 3.

The fourth question was "How happy are you with how to find information about objects in the tool (position and distances in the bird's eye view, rest in text view)?". This question aims to find whether the information that the users expect is included in either the visuals or the text view, and that it is included at the right place. Most of the participants were satisfied with the information representation and the average number of points was 9.

How happy are you with how to find information about objects in the tool
(position and distances in the bird's eye view, rest in text view)?

(7 responses)

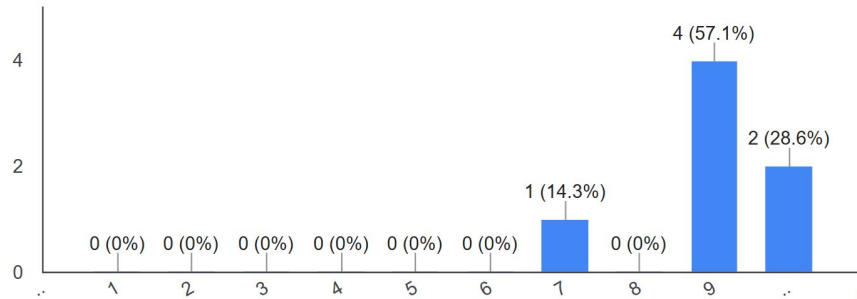


Figure 28: Validation question 4.

The fifth question was "What feature(s) are you missing in the tool?". This was an open question that resulted in the following list:

- Camera overlay.
- Legend for objects in the visuals.
- Instructions how to use the tool.
- Point cloud overlay.
- Description of the buttons in the main view when the user hovers over them.
- Reading databases that specify how the information is stored in the sensor data and extract the information accordingly.
- Save created filters and load them when the application is started.
- Showing raw data in the visuals.
- A deviation view relative to the host car.

Most of those requests were also mentioned in the elicitation phase, but they were not part of the tool for priority and scope reasons.

The last question was "Do you have any final comments?". This question aimed to get an overall view of the tool and the participants' impression about the tool. Most of the participants liked the tool very much. All the comments were "Very good" or something similar. There was no additional feedback about the tool.

6

Discussion

The goal of this thesis was to find out how sensor data of AVs can be visualized in order to make it easier for humans to analyze the data and understand the behaviour of the vehicles on the test track. In this chapter the results of the different phases of this thesis and how they relate to this goal are analyzed and discussed.

Kobayashi et al. [14] says that a visualization tool serves as a good communication tool and also that any such tool needs to answer four questions. The first question is to find out the necessary information to be visualized. This is achieved through different methods: two focus groups, a stakeholder survey, an analysis of similar systems, a literature review and a data study. The second question is about the identification of the right technologies to implement the tool. This is achieved through the use of a literature review, an analysis of similar systems and a feasibility study. The third question is about the validation of the used technique to visualize the data. The stakeholders were also involved in the technology selection and after the selection a prototype of the tool was developed and validated by the stakeholders in a review meeting. The fourth question is about the results obtained from the visualization, which are presented in chapter 5.

6.1 Elicitation phase

The first step in this thesis was to elicit the requirements of the architecture and the parameters and features to include in the visuals. A lot of ground work was done to decide on the methods that were used to do this.

To avoid bias from any of the main stakeholders at Volvo Cars or the students, it was decided to do a focus group. This was done, because focus group interviews are unstructured and are more open for discussions and that can help to bring out more ideas. In order to make the tool usable not only for the engineers at Volvo Cars, but also to other people working in this domain, it was decided to organize a second focus group at Chalmers. After the focus groups, a survey was done to validate the elicited requirements from the focus groups and also to elicit more requirements.

The decision to do a survey was taken because people may get more ideas after seeing some real requirements and as planned, the decision helped to elicit some additional requirements.

During the requirement elicitation phase many different requirements were gathered. Some of them were mentioned repeatedly, like the bird's eye view, while others were the wishes of only one single person, like acoustic feedback for important events. All of the wishes were converted into requirements, where the ones that were only mentioned rarely got a lower priority. While all requested features can help some people analyze the data being visualized, the implemented tool focuses on the ones with the highest priority. However, the created architecture aims to fulfill all of the requirements. Furthermore, most of the participants mentioned the basic features (overview, zoom, filter, details-on-demand, relate, history and extract) as stated by Fagant et al. and Zankl [12, 15]. All those features got high priority through the prioritization process and are part of the implementation as well.

In the focus groups and the survey, the participants focused more on the features they expected from the tool, instead of non-functional properties of a tool. The elicited non-functional requirements are either platform requirements, i.e. what operating systems shall be supported, or very abstract (for instance, the online and offline mode shall have the same look). People expect the tool to start and load files "fast", but it is hard to specify what amount of time is allowed to still call it "fast". The participants of the focus groups talked on a higher level of abstraction, making it hard to deduct non-functional requirements.

Eliciting what parameters and features to include in the visuals was very challenging. When asked what people want to see, most people gave an answer in the sense of "all the normal things", with some people giving a few examples, for instance other cars or road edges. People appeared to not have given much thought to what specific things they want to see, or thought it is obvious. This problem also appeared in the survey where most people chose almost all the provided options, making a selection hard. Zankl [15] says that in an environment where lots of continuous data is generated, it is impossible to visualize all the data. As said, only the position and the size of the objects are shown in the visual itself, while other details are shown in a text view.

Another source for parameters and features was the literature study. This study gave some valuable ideas what to visualize, but as Keim et al. [113] and Zankl [15] stated, it is important to involve humans in the process of deciding what parameters and features to extract from a large data set in every specific context. Work on similar projects, even in the same area, had different goals in mind and for them different details were important than what is the case in this thesis. Furthermore, as stated in the related work chapter (chapter 3), very little work has been done on the topic of visualizing tests of autonomous vehicles. In the end the literature survey gave less input on what parameters and features to include in the visuals as expected.

The expectations on both the tool itself, as well as how the visuals should be created were different depending on the asked person. People who have different roles have a different way of looking at things. What is very important to one person is not interesting to everyone else. This makes it challenging to create visuals that are helpful for everyone. Details that provide valuable insight for someone could be unimportant clutter to others. Creating different visuals for all groups of people is a contradiction to the idea of developing one tool for everybody to use. All the wishes mentioned in the elicitation process were turned into requirements. These requirements were then prioritized in three different ways. To get an overall view of the requirements and to avoid bias from any person or group, the decision was taken to prioritize the requirements in three different ways. The requirements which falls into the selection criteria mentioned in chapter 4 are considered for implementation.

6.2 Architecture creation

The created architecture follows the MVC pattern. The user interacts with the controller, that manipulates the model, which in turn updates the view. This approach was chosen to get a clear separation of concerns and a visible information flow through the architecture.

An early version of the architecture consisted of two tiers: One that was responsible for reading the input and one that created the visuals. The two parts could then be implemented using different technologies. In that case, one part could be written in a lower level language like C or C++, which can be used to create sockets, read the input and extract information from the input, while the visualization part could be written in Javascript and a WebGL based technology for creating the visuals. However, this approach would require communication between the two parts over either the localhost or by reading and writing to common files, or establishing some form of shared memory. This overhead would interfere with the “live” requirement on the visualization. The created architecture was evaluated in the review meetings with the main stakeholders and the supervisors to get their feedback and this idea was eventually dropped.

Based on the knowledge gather while creating the early version and the feedback from the main stakeholders and the supervisors, it was decided to create an architecture, which bundles backend and frontend of the tool as a single entity. This way, communication between the backend and frontend was avoided. The current architecture has three main components, the model, the view and the controller. Each component in the architecture was assigned with a single and clearly defined purpose. The controller receives all inputs from the user and passes it to the model. The controller has two sub-components, one to handle the GUI related tasks and another to handle the user input tasks. The model component is divided into two main components, the Input component and the Model component. The Input component is responsible for handling the received inputs and the Model component is

responsible for handling the created models and further they are subdivided into additional components based on their roles. The view component has different components for each view. For instance, the `BirdsViewGenerator` component is used to generate visuals in the bird's eye view and the `TextViewGenerator` component is used to generate the text view of the data model. Furthermore, the architecture has well defined interfaces and as a result, information hiding is enabled. All information regarding a component is private and if any external component needs access, it can be done through the interface of each component. This also makes it easier to replace any component if necessary and thereby reduces the coupling. Low coupling is also achieved by avoiding circular dependencies. None of the component in the architecture depends on each other. The relevant modules are packed in same component and they interact well and thereby increase the cohesion between different components. The architecture is simple to read, as the flow is in one direction. The controller modifies the model and the model updates the view and there is no cross reference at any level.

The architecture also fulfills the elicited non-functional requirements. The models are generated based on the timestamp and they are stored in the RAM until the user closes the application. The user can jump forward or backward in time or replay from any point in time to view the data in the visual at that time, and this increases the availability of the data to the user. Furthermore, each component in the architecture is allocated with single responsibility, for instance the `ModelGenerator` component generates models for the features it receives, `BirdsViewGenerator` generates only the bird's view of the visual. Thereby the responsibilities are allocated clearly and this improves adaptability of the tool.

6.3 Technology selection

One of the main stakeholder requirements of the visualization tool was that it would work in a browser based environment. This would ensure that the tool runs on many different platforms with little to no porting issues due to the fact that HTML5, CSS3, Javascript and WebGL are available in most modern web browsers. However, there are several security considerations that made a pure browser based implementation impossible by design. An unrestricted access to websocket communication and the local filesystem could be used by hackers to cause a lot of damage. Every time a person visits a malicious homepage, multiple requests could be sent from the visitors computer and by that, making him or her participate in a distributed denial of service attack. Furthermore, the website would be able to read arbitrary file on the user's hard drive and send the contents to the hacker. Because of this potential for misuse, it is unlikely that the security restrictions will be lowered in future and thus making a pure browser based implementation possible.

This thesis also considered direct memory management to use a fixed amount of RAM as a buffer for the gathered data as one of the requirements. When the buffer

is full, the oldest data should be deleted to make room for new measurements. The investigation showed that modern programming languages use automatic garbage collection to delete objects from the memory that are no longer referenced. It is not possible to allocate a fixed amount of memory and manage it in the program in Java or Javascript. C and C++ provide a method to allocate a fixed amount of memory that is reserved until it is actively freed using another method. The C++ based solutions weren't optimal for a browser-based implementation. This led to a conflict between the requirement that the tool shall run in a browser and the requirement that the tool should be able to manage the memory itself. A trade-off had to be made and based on the prioritization process, the browser support was considered more important.

The nw.js framework, formerly known as node webkit, was chosen to implement the tool. This is a mostly web based solution with a node.js part for mitigating the limitations of a pure online solution. An application written only with modern web technologies like HTML5, Javascript, CSS3 and WebGL should run in all browsers on all platforms, including mobile, with only minor layout differences. The usage of nw.js limits the supported platforms to those supported by the webkit as well, namely Windows, Linux and Mac. This trade-off was necessary to overcome the limitations from a pure web-based solution mentioned above. Furthermore nw.js comes with a bundled chromium web browser, which is the only one that can be used for the user interface.

An alternative to using nw.js is to write a server-side application in node.js and use any browser as frontend to render the user interface and the visuals. This solution is similar to nw.js, that bundles both parts into one application. Both solutions were evaluated with the main stakeholders and based on the discussions it was decided to use the nw.js based solution to implement the tool. If the node.js and any web browser solution had been used, there might have been compatibility issues when either a new web browser version is installed or node.js is updated. Nw.js allows bundling itself with code written for it and shipping an executable containing node.js, the chromium browser and the application together. All users using the tool have the same version of node.js and the web browser. This approach mitigates compatibility issues.

Three.js, a Javascript and WebGL based library, was chosen for the front-end 3D rendering along with nw.js. Initially X3D, a XML based technology for 3D rendering, was considered. In X3D it is needed to regenerate all XML tags over and over again in every frame when objects are dynamically generated. In three.js it's enough to create objects once and they can be cloned when necessary. On basis of the hands-on experiments done with three.js and X3D, it was decided to choose three.js for the generation of the 3D scenes.

Instead of the node webkit, a game engine could be used to generate the visuals. For this thesis, Unity and Unreal Engine were considered. While both could be used to generate the visuals, they are optimized for their main purpose, which is

game development. None of them is able to open multiple views and they are also not optimal for creating the non-visualization parts of the application i.e. the user interface and the background logic.

Java with its 3D library Java3D could have been used to implement the tool. It fulfills almost all requirements and Java is a widespread language. However, there is an issue with the browser support of Java applets. Most browsers have stopped or plan to stop supporting browser plugins, a feature that is needed for Java applets to work. Oracle, the owner of Java, has notices this and plan to stop supporting applets themselves with the release of next major version of Java, Java 9 [114]. It will be replaced by a new technology called Java Web Start, but this transition phase makes it hard to develop code that reliably will work for a longer period of time. However, it would still be possible to develop a non-browser-based application that can run on any platform that has a Java runtime environment.

Two desktop frameworks were also considered to develop the tool: The visualization toolkit and Qt. Even though these frameworks don't support web browsers for the front-end, they were considered to make a comparison study and to find one or a combination of technologies which can make the development of the tool simple and efficient. Both VTK and Qt fulfilled most of the requirements. Qt was also used in one of the similar systems studied in the requirement elicitation phase described in section 4.2.3. The reason it wasn't used for the tool developed in this thesis is that it does not use browser-based technologies, which is a important requirement for this thesis. If less emphasis is put on this aspect, these frameworks are good solutions for writing a visualization tool for live sensor data.

Most investigated technologies were able to fulfill most of the requirements, but all of them had some disadvantages. Using a framework for C or C++ gives more control over the memory and low level operations and enables the developer to create an efficient program that fulfills all feature requirements. A Web-based solution however will run on many platforms but have the limitations mentioned earlier, that were introduced for security reasons. In the end the choice of technology depends on which requirements are considered as most important for the person or organization doing the implementation.

6.4 Development

The development was done using nw.js, a framework using node.js in the background and Javascript as programming language. As identified in the feasibility study, it is possible to implement most of the requirements using those technologies and during the implementation no new technology dependent problems were discovered. This validates the results from the feasibility study. The tool fulfills all the requirements selected for implementation. The exact list of implemented requirements is specified in chapter 5.

The visuals should be created "live" when the vehicle is on the road. When the work on this thesis started the word "real-time" was used to describe the time constraint. However, real-time can't be guaranteed, since the data is generated at a remote location, sent via UDP and then processed in Javascript. Every step takes some time and there is no way to show that all "real life" events end up in the visual within a certain, very small, timeframe. The "live" implementation, that the tool uses to create visuals in online mode, stores all newly read packages to a buffer. An interval is used to periodically render the contents of the buffer and then empty it. This way a maximum timeframe can be specified from the receiving of the data to when it is rendered. This is also known as "soft" or "near" real-time.

One aspect that was hard to estimate through the feasibility study alone was the performance of the implementation. To really be sure that a solution is able to handle and visualize data in a fast manner, it has to be developed first. Some simple tests are not enough to ensure feasibility. This is even more important in online mode, where too slow data processing will lead to new measurements building up a queue and the visualization to fall further and further behind. This led to a small problem in the implemented solution. In offline mode, the file is read and the data model is created. At every timestamp, all data instances whose timestamp falls within certain limit of time difference are collected. Those instances are then visualized and after a short wait the process is repeated for the next timestamp. This searching process takes too much time if the data has to be processed and the necessary parameters and features have to be extracted at the same time. For that reason a different approach was used in online mode. All new measurements are stored in a buffer in addition to the data model. Instead of searching through the data model each time a new frame is to be rendered, the content of the buffer is rendered instead. If the user wishes to go back in time while using the online mode, the offline mode algorithm is used to search the data model. This ensures good performance in online mode, even if the data model has a large amount of data, gathered through a long timespan of recording.

In online mode the tool keeps recording data until the user ends either the program or the data receiving. Since no data is deleted while the online mode is running, the tool will run into a memory problem eventually. This problem is related to the memory allocation issue in section 6.3. One possible solution would be to periodically check if there are measurements older than a specific amount of time and actively delete them. This could however delete unnecessary many measurements since the amount of time not necessarily correlate with the number of received measurements, which depends on the number of sensors and how often they send data. If the test crew goes on lunch break and no new measurements arrive during that period of time, there is no need to delete "old" measurements, since no new ones arrive. This issue is not solved in this thesis, but is added as a suggestion for future work.

The created 3D environment with a movable camera has a lot of advantages over just watching a video of an AV test. In the tool it is possible to pause the visualization at an interesting event and move the camera to a angle where it is easiest to see

what is happening in that situation. It is then possible to click on the different objects involved in the situation to get more details about how the sensors sees the objects. In combination with the ability to go forward or backward frame by frame (jumping forward or backwards a small amount of time), it is possible to analyze a situation quickly by accessing the detailed information in an easy way. This directly improves the analyzability of the data.

One requirement of the visualization tool was to be able to hover over an object in the visualization to show its details, instead of having to click on it. From an implementation perspective this is easy to do, as the code that currently happens on a mouse click event has to be executed on a mouse move event instead. However, there is a performance problem with that solution. To find out what object is behind the mouse cursor a technique called ray tracing has to be used. This technique involves sending a ray through the scene and calculating what objects it hits. Even without reflection or refraction, this takes some time to calculate. If these calculations are done every time a frame is rendered, the performance of the visualization in total decreases, as a simple test while implementing the requirement for clicking on objects showed. While showing some information in a tooltip while hovering over an object would improve the analyzability of the data, this solution is not feasible.

The time series helps the user to analyze the path history of the selected object, in the view of the sensor. As mentioned in chapter 1, the collected data is numeric and is very hard for humans to understand. The time series will draw a line representing the path of the object from its entry into the field of view of a sensor till its exit. This also helps the user to validate the sensor readings. For instance, if the object jumps from one position to another, or travels back in comparison to its previous position, then the user can easily see that in the visual through the time series lines.

The decision to use a framework for creating the visuals instead of creating them in pure WebGL made the implementation easier. Three.js (and the alternative, X3D) allows creating 3D scenes without having to learn or spend time on "low-level" computer graphic concepts like fragment- and vertex shaders, or matrix transformations. Adding anisotropic filtering, to make distant objects appear sharper in the visual, was implemented using one line of code. The same feature in pure WebGL would have been harder to implement. The decision to use Three.js therefore increased the development speed.

One of the challenges described in the introduction of this thesis was that the online and offline mode should have the same look and feel. As of now, different tools are used for analyzing online and offline data at Volvo Cars. There is a wish to be able to use the same tool with the same functionalities in both online and offline mode. The tool that was developed in this thesis had this in mind during the design phase and all functionalities of the tool is available both online and offline. This makes analysis much more convenient, as there is only one user interface and the same way to control the tool, no matter whether a logfile or live data is being analyzed.

6.5 Deviation between Sensor Data

The comparison of data from multiple sensors is a good way to validate the data collected by the sensors. In an ideal world, sensors would provide exact measurements and the computer making autonomous decisions about how the car should drive could rely on the collected data. However, in reality there will always be measurement errors and to validate how well a sensor works in certain conditions, it is important to be able to compare the data to some other value. The comparison can be either to another sensor, or in a testing scenario, where the objects location are known, to the ground truth. Knowing how large the deviation in the measured data is from the truth helps to analyze the data of AV tests.

In this thesis, two different approaches to compare different objects were used. The first approach compared the position of every object sensed by one sensor with the position of every object sensed by another sensor. If the center points of the objects were within two meters of each other in both latitude and longitude direction, they were considered to be the same object, which was shown in the visual. The second approach was to let the user choose what objects to compare. In the first approach, the user can't miss any overlaps of objects, but calculating all the distances every frame takes time and requires a strong processor. The second approach gives the user more control and is faster, but the user has to know what to look at and can't depend on the tool to find the overlaps.

The identification of deviations in the sensor data was done as an iterative approach and in each iteration constructive feedback was given by the main stakeholders. It shows their interest in this feature and the results in each iteration. In each iteration the results are further improved, based on the discussions and feedback. Even after finishing the iterative approach, new ideas to improve the visualization can be generated. In this thesis, the color of the objects that are compared was changed from a solid black and red to semi-transparent versions of those colors. This made the overlaps even more visible. Instead of eliciting the requirements of this feature in the beginning and then doing the implementation, the iterative approach led to a better solution, because people get more ideas when they see a solution. It is hard to think of the "best" way to create an artifact at once, without undergoing several iterations.

6.6 Validation

As stated by Logre et al. and Broring et al. [13, 16], the visuals help the users to analyze the test data and acquire information and knowledge about the data. This fact was validated in the review meetings with the main stakeholders. In the meetings, the stakeholders were able to see the visuals and immediately knew what

was happening on the screen. This would not have been possible with only the text data.

To understand the general view of other users about the implemented tool, a user evaluation was done after the development. The feedback of the user evaluation was mostly positive. In the evaluation a question was asked about what features was missing in the tool and the most common answers were camera and point cloud overlay. These were part of the requirements identified in the elicitation phase, but that were not implemented because of their priority or the scope of this thesis. The same is true for most other requirements mentioned in the validation. The fact that the requirements that were elicited and not implemented are mentioned again in the validation phase shows that the elicitation phase resulted in the correct requirements.

In total, the persons that participated in the validation seemed very satisfied with the tool. The lowest rating received on any of the questions was a six, and the lowest average was 7.43, on the question "How clear is it what the objects in the visual represent". Two of the seven participants requested for a legend to explain the different colors used for different objects in the visual. By adding a legend, the information becomes more clearer and the rating would probably rise.

6.7 Threats to Validity

In this section the threats to validity of the research performed in this thesis is outlined.

6.7.1 Construct Validity

In the literature survey, the assessment stopped when ten papers in a row were considered not relevant for the study. This was done because the searches returned many papers and not all of them were relevant to this thesis. The papers were ordered according to how relevant the search algorithm of the database thought it matched the search query. However, this priority might not be the same as how relevant the paper is for the topic of this thesis. By ordering the papers wrong and stopping the assessment too early, important papers could have been missed.

The analysis of similar systems was done by attending two presentations at Volvo Cars. The companies that held the presentation had been invited by Volvo Cars and the tools that were presented aimed to achieve similar goals as the tool that was developed as a part of this thesis. However, there is no guarantee that the two presentations actually presented the requirements that are most important for a

vehicle sensor data visualization tool. The similar systems could have implemented requirements that are not needed, or missing other requirements, or both.

This thesis only considered parameters and features that could be found in the data of the two sensors that were included in the data study. Other sensors, that either are used currently or are added later, might include additional visualization details that are not considered in this thesis. This may result in the elicited list of parameters and features to include in the visuals becoming incomplete or out of date.

In this work, three different approaches were used to prioritize each requirement. For some of the requirements this worked very well, and the same or at least a similar priority was elicited using every different technique. However, for other requirements the priorities differed based on the used technique. Additionally, some requirements seemed to be very important to some people, and less important to others. This makes it unclear if the prioritization techniques measured the importance of the requirements correctly.

The feasibility study was conducted mainly using the official documentation of the considered technologies. The resulting feasibility table therefore shows if the documentation claims it is possible, and not that an example implementation was created that definitely shows that it is possible. While it is likely that the information is true, this thesis gives no information on how well the technology is able to fulfill a specific requirement. Furthermore, only some suitable technologies were chosen for the feasibility study. There could be others that are just as, or even more, suitable for implementing a visualization tool.

The offline mode of the tool was tested with a logfile containing test data with a duration of one minute, so there may arise performance or scalability issues when data of longer time is used. The online mode of the tool was not tested with real live data. Instead, it was tested by replaying the one minute log file from the Ubuntu operating system installed in a virtual machine using the `tcpreplay` tool [25]. The replayed data was captured by the visualization tool in Windows. Therefore, when live data is sent directly from the test, performance issues may arise.

6.7.2 External Validity

Two focus groups were conducted, one at Volvo Cars and one at Chalmers University of Technology. In both cases the supervisors of this thesis took the first contact to the participants and asked who was willing to participate. This is a very convenient sampling approach, but there is no guarantee that the people who are willing to participate in the focus groups are representative for the entire population. Furthermore only four persons participated in the first focus group and two in the second. Thus, the small sampling size makes it hard to generalize the results.

A similar threat is present for the results of the survey. The survey was sent to three groups in one department at Volvo Cars and was left online for approximately two weeks. During this time, 11 out of 54 persons responded. This is again a convenient sampling method, as only people who wanted to participate, did so. The small sample makes it hard to generalize the results to the entire population. Additionally, because only people from one department at Volvo Cars was invited to participate in the survey, the results can't be generalized to the automobile industry as a whole or even to other areas without collecting further data.

6.7.3 Internal Validity

The questions for the focus groups and the survey were created by two students without much experience in qualitative data elicitation. This might have led to unintended effects on the outcome of the data elicitation, even though the questions merely were used as guidelines for the otherwise open discussion. The same is true for the analysis of the collected data, since the students weren't experienced with interpreting tones, body language and similar aspects of the participants.

Between the initial requirements elicitation phase and the final validation of the tool, a new visualization software that had been developed by a consultancy company was released at Volvo Cars. Some people who were either involved in the development of this software, or had used it, were part of the validation of the tool developed in this thesis. There is a risk that the expectations of the people who had worked with this software before the validation of the tool changed. This might have influenced the result of the validation.

Some of the participants of the requirements elicitation process were also part of the evaluation of the tool. Therefore, the feedback of those participants might be influenced, based on what they expected or what was discussed during the elicitation process.

7

Conclusion

This thesis showed that visualizing sensor data makes it easier for humans to analyze the data. To do this, the requirements of a software architecture of a tool to visualize sensor data were elicited. Furthermore, the thesis investigated which parameters and features humans like to see in the visuals to be able to draw conclusions about what is happening in the scene. A feasibility study was conducted to decide on what technologies are suitable for developing a tool using the created software architecture. Based on the elicited requirements, the created architecture and the technology study, a visualization tool was developed to demonstrate the feasibility of the solution. The tool was then used to generate visuals showing the differences in the data of multiple sensors measuring the same object to visualize the deviations in the sensor measurements.

This thesis elicited requirements through multiple sources for a tool that is used to visualize the sensor data in the context of AVs. Every method used in the elicitation process resulted in many unique requirements. This shows the importance of eliciting requirements through multiple sources instead of relying on single source. When people are involved in the process, their expectations and views may change and it is important to create a solution that satisfies the needs of all the stakeholders.

The created architecture follows the MVC pattern and standard design principles. It also satisfies all the elicited requirements and therefore serves as a good reference for developing similar tools in the context of visualization of sensor data from vehicles.

When creating visuals from sensor data, it is important to consider the needs of the users. If the visuals should be useful for people with different roles, it is worth to consider to make it possible to filter out some information, based on the needs of the viewer. Furthermore, not all details have to be included in the visual itself. Instead, an easy way has to be available for the user to access additional information about the objects in the visual. For any visualization tool it is important to elicit the needs of the humans who will work with the visuals and the tool itself.

The choice of technologies to use for implementing a visualization tool depends on how the stakeholders want to use the tool. Web based technologies can be used to create a tool that runs on many devices and operating systems, but they have

many limitations for security reasons. Desktop based implementations provide more functionality but are often tied to one or a few operating systems. This issue becomes even more serious when developing a tool for both desktop and mobile operating systems. In the end, the stakeholders have to make a choice what is most important for them. The feasibility study table gives insight about the possibilities of using different technologies to implement the requirements considered in this thesis.

Comparing the data of different sensors, or the data of one sensor to the ground truth provides an excellent way of validating said data. There are different ways these comparisons can be visualized, based either on automatic overlap detection, or by allowing the user to select different objects and compare them. A combination of using colors in the visualization and a text representation in a separate window gives the user a good way to analyze the data.

Visualization makes analysis of sensor data much easier for humans. By using a tool with the right architecture and suitable technologies to create visuals that include the necessary information for researchers and engineers, it becomes possible to draw better conclusions from AV tests. This will in turn aid the pursuit of creating fully autonomous vehicles in the near future.

7.1 Future Work

This section outlines possible future work that hasn't been included in this thesis for scope or time reasons, but still give valuable contributions to the thesis topic or improves the developed tool.

There are more ways to visualize deviations in the sensor data. For instance, it is possible to draw all the objects recognized by one sensor in the same color, and have a different color for each sensor. This approach can be combined with the automatic overlap detection and showing non-overlapping objects in a very light version of a color and as soon as two objects recognized by different sensors overlap, their color can be changed to a darker version of the same color to make the overlap clearly visible in the visualization. It is also possible to define one of the sensors as a reference sensor and visualize it in a stronger color, while visualizing data from all other sensors in a lighter color. Both solutions make deviations clearly visible and help humans to analyze the data.

The visuals generated by this tool only show data from sensors on one car. As a next step, information from different vehicles or even static points could be added to a common picture. In that case, not only deviations in the data from different sensors on one car about an object can be analyzed, but also where different cars think the same object is placed. To make this possible, all vehicles could make their sensor data available on a network interface and the tool could read measurements from all the senders by collecting everything that is online.

The tool stores all data it receives in order to make it possible for the user to go back in time and look at older data. When using online mode over a longer period of time the gathered data takes up a lot of space. If old data is never deleted, the entire RAM will get filled, making it impossible to store new data. A concept has to be developed how the data management is to be handled. A solution created in C or C++ has the ability to allocate a fixed amount of memory for this data buffer and delete the oldest entry when this space is used, but other solutions have to be found for Java or Javascript based implementations.

The requirement elicitation phase resulted in a requirement to overlay object data over a camera image, and another requirement to overlay the data over a point cloud (LIDAR output). The prioritization process gave a high priority for the camera overlay. However, the test data that was used while developing the tool did not contain any camera data or point clouds, so this thesis focused on other requirements. In future this would be a valuable addition and improve the analyzability of the data. Both of the requirements were mentioned in the validation of the developed tool as well.

Textures and well-designed models make the visuals look more natural and makes it easier for the human who uses the tool to imagine what is happening in the scene. For the visuals created in this thesis, boxes of different sizes and colors were used to represent different types of object. For some object types textures were used, but the textures only consisted of six images applied to the different faces of a box. For an improved version of the tool, 3D models of cars, trucks, pedestrians and similar objects can be created and higher quality textures can be applied to them.

As of now, only .pcap files are supported as input for the tool. Since a lot of people work with Matlab files, a future version of the application should be able to open and read this type of files as well. Furthermore, the application could add support for a file type that is associated with the application. Once the needed information is extracted from a data source and converted to the applications data model, the data that is kept in the tool can be stored in a file and loaded again at a later point in time. Since this file type would store the data in the tools data model, no conversions are necessary when loading the file, which makes starting faster.

The requirements of the architecture and the parameters and features to include in the visuals were elicited at Chalmers University of Technology and Volvo Cars. Even though that gave valuable input to this work, as a next step, the requirements could be elicited at other companies or institutions as well. This would broaden the scope and validate the results of this thesis.

The data used in this thesis was object data that had already been processed and fused. This process consists of multiple steps and an error could happen in any of them. To be able to analyze where in this process the fault is and therefore know why the AV don't behave as expected, the result of every step of this process should be visualized. This was a requirement that was elicited in the requirement phase of

7. Conclusion

this thesis. As that requirement was assigned low priority, it was not considered for the implementation. This could be addressed in the future. The same is true for visualization of raw data, that has not been processed or fused.

Bibliography

- [1] “Volvo car group global media newsroom.” <https://www.media.volvocars.com/global/en-gb>. [Accessed: 2016-05-16].
- [2] “About | astrazero.” <http://www.astazero.com/about-astazero/about/>, 2016. [Accessed 2016-03-15].
- [3] B. Kitchenham, “Procedures for performing systematic reviews,” tech. rep., Keele University, 2004.
- [4] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information system research,” *MIS Quarterly*, 2004.
- [5] J. M. Anderson, K. Nidhi, and K. D. Stanley, *Autonomous Vehicle Technology*. RAND Corporation, 2014.
- [6] L. Figueiredo, I. Jesus, J. Machado, J. Ferreira, and J. de Carvalho, “Towards the development of intelligent transportation systems,” *Proceedings. 2001 IEEE Intelligent Transportation Systems*, 2001.
- [7] A. Jarasunienea and G. Jakubauskasb, “Improvement of road safety using passive and active intelligent vehicle safety systems,” *Transport*, 2007.
- [8] “itransit - intelligent traffic management system based on its - vinnova.” <http://www.vinnova.se/sv/Resultat/Projekt/Effekta/2009-02186/iTRANSIT---intelligent-TRAffic-maNagement-System-based-on-ITS/>, 2016. [Accessed 2016-03-15].
- [9] M. Sedlmair, P. Isenberg, D. Baur, M. Mauerer, C. Pigorsch, and A. Butz, “Cardiogram: Visual analytics for automotive engineers,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011.
- [10] L. Gerard, A. Javi, E. Alun, and B. Josep, “Web-based data visualization of an mmo virtual regatta using a virtual globe,” *Proceedings of the 20th International Conference on 3D Web Technology - Web3D '15*, 2015.

- [11] K. Bimbraw, “Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology,” *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2015.
- [12] D. J. Fagnant and K. Kockelman, “Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations,” *Transportation Research Part A: Policy and Practice*, 2015.
- [13] I. Logre, S. Mosser, and M. Riveill, “Composition challenges for sensor data visualization,” *Companion Proceedings of the 14th International Conference on Modularity*, 2015.
- [14] T. Kobayashi, M. Tsubokura, and M. Oishi, “Technology of automobile and visualization studies,” *Journal of Visualization*, vol. 11, no. 1, pp. 15–22, 2008.
- [15] S. Zankl, “Visualizing sensor data,” tech. rep., Ludwig Maximilian University Munich, 2008.
- [16] A. Bröring, D. Vial, and T. Reitz, “Processing real-time sensor data streams for 3d web visualization,” *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming*, 2014.
- [17] M. Hammoudeh, R. Newman, and S. Mount, “An approach to data extraction and visualisation for wireless sensor networks,” *Eighth International Conference on Networks, 2009. ICN '09*, 2009.
- [18] M. Tonnis, R. Lindl, W. L., and G. Klinker, “Visualization of spatial sensor data in the context of automotive environment perception systems,” *6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007. ISMAR 2007.*, 2007.
- [19] R. Stampach, P. Kubicek, and L. Herman, “Dynamic visualization of sensor measurements: Context based approach,” *Quaestiones Geographicae*, 2015.
- [20] C. Roessing, A. Reker, M. Gabb, K. Dietmayer, and H. P. A. Lensch, “Intuitive visualization of vehicle distance, velocity and risk potential in rear-view camera applications,” *IEEE Intelligent Vehicles Symposium (IV)*, 2013.
- [21] “Databases - chalmers library.” <http://www.lib.chalmers.se/en/search/databases>, 2016. [Accessed: 2016-03-29].
- [22] K. Baxter, C. Courage, and K. Caine, *Understanding your Users (Second Edition)*. Elsevier, 2015.

- [23] “Development/libpcapfileformat - the wireshark wiki.” <https://wiki.wireshark.org/Development/LibpcapFileFormat>, 2016. [Accessed: 2016-03-15].
- [24] “Wireshark · go deep.” <https://www.wireshark.org/>, 2016. [Accessed: 2016-03-15].
- [25] “Tcpreplay overview.” <http://tcpreplay.appneta.com/wiki/overview.html>, 2016. [Accessed: 2016-03-15].
- [26] “mrdoob/three.js.” <https://github.com/mrdoob/three.js>. [Accessed: 2016-05-09].
- [27] “What is x3d | web3d consortium.” <http://www.web3d.org/x3d/what-x3d>. [Accessed: 2016-05-09].
- [28] “x3dom.org.” <http://www.x3dom.org>. [Accessed: 2016-05-09].
- [29] “About | node.js.” <https://nodejs.org/en/about/>. [Accessed: 2016-05-09].
- [30] “Unity - game engine, tools and multiplatform.” <https://unity3d.com/unity>. [Accessed: 2016-05-09].
- [31] “What is unreal engine 4.” <https://www.unrealengine.com/what-is-unreal-engine-4>. [Accessed: 2016-05-09].
- [32] “Java se desktop technologies - java 3d api.” <http://www.oracle.com/technetwork/articles/javase/index-jsp-138252.html>. [Accessed: 2016-05-09].
- [33] “Vtk - the visualization toolkit.” <http://www.vtk.org>. [Accessed: 2016-05-09].
- [34] “Qt - product | qt for application development.” <https://www.qt.io/qt-for-application-development/>. [Accessed: 2016-05-09].
- [35] “Rfc 6455 - the websocket protocol.” <https://tools.ietf.org/html/rfc6455>, 2016. [Accessed: 2016-03-29].
- [36] “Udp/datagram sockets node.js v5.9.1 manual & documentation.” <https://nodejs.org/api/dgram.html>, 2016. [Accessed: 2016-03-29].
- [37] “Technologies, u.unity - manual: Security sandbox of the webplayer.” <http://docs.unity3d.com/Manual/SecuritySandbox.html>, 2016. [Accessed: 2016-03-29].

- [38] “Unity - manual: Using the transport layer api.” <http://docs.unity3d.com/Manual/UNetUsingTransport.html>., 2016. [Accessed: 2016-03-29].
- [39] “Unreal engine | fudpsocketbuilder.” <https://docs.unrealengine.com/latest/INT/API/Runtime/Networking/Common/FUdpSocketBuilder/index.html>, 2016. [Accessed: 2016-03-29].
- [40] “Datagramsocket (java platform se 8).” <https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html>, 2016. [Accessed: 2016-03-29].
- [41] “Vtk: Vtksocketcommunicator class reference.” <http://www.vtk.org/doc/nightly/html/classvtkSocketCommunicator.html#details>, 2016. [Accessed: 2016-03-29].
- [42] “Platform agnostic | vtk.” <http://www.vtk.org/features-platform-agnostic/>, 2016. [Accessed: 2016-03-29].
- [43] “Qudpsocket class | qt network 5.6.” <http://doc.qt.io/qt-5/qudpsocket.html>, 2016. [Accessed: 2016-03-29].
- [44] “File api.” <https://www.w3.org/TR/FileAPI/>, 2016. [Accessed: 2016-03-29].
- [45] “File system node.js v5.9.1 manual & documentation.” https://nodejs.org/api/fs.html#fs_class_fs_readstream, 2016. [Accessed: 2016-03-29].
- [46] “Technologies, u.unity - manual: Text asset.” <http://docs.unity3d.com/Manual/class-TextAsset.html>, 2016. [Accessed: 2016-03-29].
- [47] “Unreal engine | ffilehelper.” <https://docs.unrealengine.com/latest/INT/API/Runtime/Core/Misc/FFileHelper/index.html>, 2016. [Accessed: 2016-03-29].
- [48] “BufferedReader (java platform se 7).” <https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html>, 2016. [Accessed: 2016-03-29].
- [49] “Input/output with files - c++ tutorials.” <http://www.cplusplus.com/doc/tutorial/files/>, 2016. [Accessed: 2016-03-29].
- [50] “Window.open().” <https://developer.mozilla.org/en-US/docs/Web/API/Window/open>, 2016. [Accessed: 2016-03-29].
- [51] “How to make frames (main windows) (the java™ tutorials > creating a gui with jfc/swing > using swing components).” <https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html>, 2016. [Accessed: 2016-03-29].

- [52] “Vtk/tutorials/qtsetup - kitwarepublic.” <http://www.vtk.org/Wiki/VTK/Tutorials/QtSetup>, 2016. [Accessed: 2016-03-29].
- [53] “Q3dockwindow class | qt 4.8.” <http://doc.qt.io/qt-4.8/q3dockwindow.html#Q3DockWindow>, 2016. [Accessed: 2016-03-29].
- [54] “Technologies, u.unity - manual: Camera.” <http://docs.unity3d.com/Manual/class-Camera.html>, 2016. [Accessed: 2016-03-29].
- [55] “Vtk/examples/cxx/visualization/sidebysideviewports - kitwarepublic.” <http://www.vtk.org/Wiki/VTK/Examples/Cxx/Visualization/SideBySideViewports>, 2016. [Accessed: 2016-03-29].
- [56] “Model/view programming | qt 4.8.” <http://doc.qt.io/qt-4.8/model-view-programming.html#view-classes>, 2016. [Accessed: 2016-03-29].
- [57] “Screenshot in javascript - learning three.js.” <http://learningthreejs.com/blog/2011/09/03/screenshot-in-javascript/>, 2016. [Accessed: 2016-03-29].
- [58] “Three.js / documentation.” <http://threejs.org/docs/#Reference/Renderers/WebGLRenderer>, 2016. [Accessed: 2016-03-29].
- [59] “X3dom documentation: Tutorials.” <http://doc.x3dom.org/tutorials/applicationTutorials/screenshot/>, 2016. [Accessed: 2016-03-29].
- [60] “Technologies, u.unity - scripting api: Application.capturescreenshot.” <http://docs.unity3d.com/ScriptReference/Application.CaptureScreenshot.html>, 2016. [Accessed: 2016-03-29].
- [61] “Taking screenshots | unreal engine.” <https://docs.unrealengine.com/latest/INT/Engine/Basics/Screenshots/index.html#standardscreenshot>, 2016. [Accessed: 2016-03-29].
- [62] “How to capture screenshot - java tips.” <http://www.java-tips.org/java-se-tips-100019/21-java-awt/1759-how-to-capture-screenshot.html>, 2016. [Accessed: 2016-03-29].
- [63] “Vtk/examples/cxx/utilities/screenshot - kitwarepublic.” <http://www.vtk.org/Wiki/VTK/Examples/Cxx/Utilities/Screenshot>, 2016. [Accessed: 2016-03-29].
- [64] “Screenshot example | qt widgets 5.6.” <http://doc.qt.io/qt-5/qtwidgets-desktop-screenshot-example.html>, 2016. [Accessed: 2016-03-29].

- [65] “Capture canvas and webgl output as video using websockets | smartjava.org.” <http://www.smartjava.org/content/capture-canvas-and-webgl-output-video-using-websockets>, 2016. [Accessed: 2016-03-29].
- [66] “Record a video - unity answers.” <http://answers.unity3d.com/questions/24146/can-i-tell-unity-to-record-a-video.html>, 2016. [Accessed: 2016-03-29].
- [67] “Creating a video from unity - unity answers.” <http://answers.unity3d.com/questions/14612/creating-a-video-from-unity.html>, 2016. [Accessed: 2016-03-29].
- [68] “Record a movie using matinee - ue4 answerhub.” <https://answers.unrealengine.com/questions/50724/can-i-use-matinee-to-download-my-clip.html>, 2016. [Accessed: 2016-03-29].
- [69] “Lecture#23 (11/30/00): Java3d.” <http://web.mit.edu/1.124/LectureNotes/Java3DLecture23.html>, 2016. [Accessed: 2016-03-29].
- [70] “Vtk: vtkgenericmoviewriter class reference.” <http://www.vtk.org/doc/nightly/html/classvtkGenericMovieWriter.html>, 2016. [Accessed: 2016-03-29].
- [71] “Camera overview | qt multimedia 5.6.” <http://doc.qt.io/qt-5/cameraoverview.html>, 2016. [Accessed: 2016-03-29].
- [72] “Location.” <https://developer.mozilla.org/en-US/docs/Web/API/Location>, 2016. [Accessed: 2016-03-29].
- [73] “Process node.js v5.9.1 manual & documentation.” https://nodejs.org/docs/latest/api/process.html#process_process_argv, 2016. [Accessed: 2016-03-29].
- [74] “Technologies, u.unity - manual: Command line arguments.” <http://docs.unity3d.com/Manual/CommandLineArguments.html>, 2016. [Accessed: 2016-03-29].
- [75] “Command-line arguments | unreal engine.” <https://docs.unrealengine.com/latest/INT/Programming/Basics/CommandLineArguments/index.html>, 2016. [Accessed: 2016-03-29].
- [76] “Command-line arguments (the platform environment).” <https://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html>, 2016. [Accessed: 2016-03-29].

- [77] “How to parse command line parameters. - c++ articles.” <http://www.cplusplus.com/articles/DEN36Up4/>, 2016. [Accessed: 2016-03-29].
- [78] “Qcommandlineparser class | qt core 5.6.” <http://doc.qt.io/qt-5/qcommandlineparser.html>, 2016. [Accessed: 2016-03-29].
- [79] “three.js / documentation.” <http://threejs.org/docs/index.html#Reference/Core/Object3D>, 2016. [Accessed: 2016-03-29].
- [80] “three.js / documentation.” <http://threejs.org/docs/index.html#Reference/Cameras/Camera>, 2016. [Accessed: 2016-03-29].
- [81] “X3dom developer api documentation: Class: Transform.” <http://doc.x3dom.org/developer/x3dom/nodeTypes/Transform.html>, 2016. [Accessed: 2016-03-29].
- [82] “Unity - scripting api: Camera.” <http://docs.unity3d.com/ScriptReference/Camera.html>, 2016. [Accessed: 2016-03-29].
- [83] “Viewport controls | unreal engine.” <https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/Viewports/ViewportControls/index.html>, 2016. [Accessed: 2016-03-29].
- [84] “Orbitbehavior (java 3d 1.3.2).” <http://download.java.net/media/java3d/javadoc/1.3.2/com/sun/j3d/utils/behaviors/vp/OrbitBehavior.html>, 2016. [Accessed: 2016-03-29].
- [85] “Vtk: vtkcamera class reference.” <http://www.vtk.org/doc/nightly/html/classvtkCamera.html>, 2016. [Accessed: 2016-03-29].
- [86] “Qgraphicsview class | qt 4.8.” <http://doc.qt.io/qt-4.8/qgraphicsview.html#details>, 2016. [Accessed: 2016-03-29].
- [87] “three.js / documentation.” <http://threejs.org/docs/index.html#Reference/Cameras/PerspectiveCamera>, 2016. [Accessed: 2016-03-29].
- [88] “X3dom developer api documentation: Class: Viewpoint.” <http://doc.x3dom.org/developer/x3dom/nodeTypes/Viewpoint.html>, 2016. [Accessed: 2016-03-29].
- [89] “File api: Writer.” <https://www.w3.org/TR/file-writer-api/>, 2016. [Accessed: 2016-03-29].
- [90] “File system node.js v5.9.1 manual & documentation.” <https://nodejs.org/api/fs.html>, 2016. [Accessed: 2016-03-29].

- [91] “Unity - scripting api: File.” <http://docs.unity3d.com/ScriptReference/Windows.File.html>, 2016. [Accessed: 2016-03-29].
- [92] “Unreal engine | ifilehandle.” <https://docs.unrealengine.com/latest/INT/API/Runtime/Core/GenericPlatform/IFileHandle/index.html>, 2016. [Accessed: 2016-03-29].
- [93] “Reading, writing, and creating files (the java™ tutorials > essential classes > basic i/o).” <https://docs.oracle.com/javase/tutorial/essential/io/file.html>, 2016. [Accessed: 2016-03-29].
- [94] “Memory management.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management., 2016. [Accessed: 2016-03-29].
- [95] “Embedder’s guide.” <https://developers.google.com/v8/embed#handles-and-garbage-collection>, 2016. [Accessed: 2016-03-29].
- [96] “V8 node.js v5.9.1 manual & documentation.” <https://nodejs.org/api/v8.html>, 2016. [Accessed: 2016-03-29].
- [97] “Unity - manual: Understanding automatic memory management.” <http://docs.unity3d.com/Manual/UnderstandingAutomaticMemoryManagement.html>, 2016. [Accessed: 2016-03-29].
- [98] “Garbage collection & dynamic memory allocation - epic wiki.” https://wiki.unrealengine.com/Garbage_Collection_%26_Dynamic_Memory_Allocation, 2016. [Accessed: 2016-03-29].
- [99] “Understanding memory management.” https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html, 2016. [Accessed: 2016-03-29].
- [100] “malloc - c++ reference.” <http://www.cplusplus.com/reference/cstdlib/malloc/>, 2016. [Accessed: 2016-03-29].
- [101] “Potree | webgl pointcloud renderer.” <http://potree.org/wp/>, 2016. [Accessed: 2016-03-29].
- [102] “Documentation - point cloud library (pcl).” http://pointclouds.org/documentation/tutorials/compiling_pcl_dependencies_windows.php, 2016. [Accessed: 2016-03-29].
- [103] “three.js / documentation.” http://threejs.org/docs/index.html#Manual/Introduction/Creating_a_scene, 2016. [Accessed: 2016-03-29].

- [104] “Firefox — 45.0 system requirements.” <https://www.mozilla.org/en-US/firefox/45.0/system-requirements/>, 2016. [Accessed: 2016-03-29].
- [105] “Browser support - x3dom.org.” <http://www.x3dom.org/contact/>, 2016. [Accessed: 2016-03-29].
- [106] “Nw.js.” <http://nwjs.io/>, 2016. [Accessed: 2016-03-29].
- [107] “Unity - system requirements.” <https://unity3d.com/unity/system-requirements>, 2016. [Accessed: 2016-03-29].
- [108] “Unreal engine faq.” <https://www.unrealengine.com/faq>, 2016. [Accessed: 2016-03-29].
- [109] “What are the system requirements for java?.” <http://java.com/en/download/help/sysreq.xml>, 2016. [Accessed: 2016-03-29].
- [110] “Supported platforms | qt 5.6.” <http://doc.qt.io/qt-5/supported-platforms.html>, 2016. [Accessed: 2016-03-29].
- [111] “Firefox for android — mobile web browser — more ways to customize and protect your privacy.” <https://www.mozilla.org/en-US/firefox/android/>, 2016. [Accessed: 2016-03-29].
- [112] “Firefox for ios — mobile web browser for your iphone, ipad and ipod touch.” <https://www.mozilla.org/en-US/firefox/ios/>, 2016. [Accessed: 2016-03-29].
- [113] D. Keim and H.-P. Kriegel, “Visualization techniques for mining large databases: a comparison,” *IEEE Transactions on Knowledge and Data Engineering*, 1996.
- [114] “Moving to a plugin-free web (java platform group, product management blog).” https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free. [Accessed: 2016-05-09].

A

Focus Group Questions

A.1 Opening Question

Please introduce yourself and tell us what you work with.

Motivation: This question starts the conversation and gives a short background of the people participating in the study.

A.2 Introductory Question

Do you have experience in working with or have knowledge about visualization tools? Please share your views.

Motivation: This question brings out the ideas and views of the participants on different visualization tools currently available in different contexts.

A.3 Transition Question

What are your general views and opinions on visualization tools in the context of sensor data of Autonomous Vehicles(AV)?

Motivation: This question helps the participants to focus more towards the context of the key questions.

A.4 Key Questions

How do you analyze sensor data today and draw conclusions from it?

Motivation: This question helps to understand the current way of working.

What are the main challenges you are facing when analyzing data?

Motivation: This question helps to understand the critical areas which need more attention and also the discussion will help to find solutions or a way to move forward.

What are your goals for visualizing the sensor data?

Motivation: This question helps to learn the advantages the participants expect from the visualization and the reasons behind requested features.

How would you like the data generated from AV tests to be visualized (e.g 2D, 3D, camera angle)?

Motivation: This question elicits if a simple birds-eye-view 2D model is preferred for analyzing data or if people prefer a 3D environment where they can freely move the camera and see what is happening from different angles.

What is the necessary information that has to be included in the visuals? (e.g. speed, distances, moving objects around the car etc.)

Motivation: It is not possible to include every single detail in a visual. This question gathers knowledge about what people think is important.

We are developing a tool that visualizes the sensor data. What features would you expect from a tool like this?

Motivation: The tool should satisfy the needs of the users. By including them from the beginning and asking them what they expect will help to elicit the requirements of the tool.

A.5 Ending Questions

Will visualization of sensor data be helpful to other people as well? Who do you think it will be helpful for?

Motivation: This question aims to identify further stakeholders that haven't been considered yet.

What did you think was most important? What did we not cover that we should have?

Motivation: This question aims to find open issues and new questions for the survey.

B

Survey Questions

How would you like to use the visualization tool?

- Online - See the visualization live while the test takes place
- Offline - Analyze a log file at a later point in time

Figure 29: First survey question.

On what devices would you like to use the tool?

- PC
- Phone
- Tablet
- Specify your own value:

Figure 30: Second survey question.

B. Survey Questions

What details should be included in the visualization of an autonomous vehicle test?

- Own car
- Lane markings
- Other cars on the road
- Pedestrians
- Speed of vehicles
- Distances between objects
- parked cars
- static objects not on the road (trees, walls, ...)
- Traffic signs
- Specify your own value:

Figure 31: Third survey question.

How important are the following views for your daily work?

	Not important		Average		Very important	N/A
	1	2	3	4	5	
1. Birds eye view of the car and its surroundings	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. View of sensor data in text form	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. View with the camera output with an information overlay	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. View with the output of different functions applied on the sensor data	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 32: Fourth survey question.

What other view(s) would help you in your daily work?

Figure 33: Fifth survey question.

B. Survey Questions

	Not important		Average		Very important	
	1	2	3	4	5	N/A
1. Import external data into the visualization tool	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. Take a screenshot of a view	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. Record a video of a view	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. Call the application with arguments to open a file and select start/end time of visualization	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5. Customize the settings of the tool (i.e. size of views, colors of objects)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. Interact with the application through an API	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. Link to Matlab (import/export data)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8. Show multiple views of the data at once	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9. Maximize one view of the data	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
10. Rotate a view using the mouse	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
11. Click on an object in a view to show more information about it	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 34: Sixth survey question.

What other features do you expect from a tool that visualizes sensor data?

Figure 35: Seventh survey question.

Do you have any other thoughts about sensor data visualization tools that you want to share?

Figure 36: Eight survey question.

C

Statement of Contributions

Finally, the contributions of the two students working on this thesis is presented. Most of the work was done by both students, and the final thesis document has been reviewed by both of them. The individual contributions is listed in the sections below.

The introduction, including the research goal and the research questions were created in collaboration by both students. The same is true for the methods used in this thesis. In the results and discussion section one of the students wrote a section that then was updated by both students together.

Matthias Pernerstorfer wrote about autonomous vehicles and active safety in general in the background section, and about Cardiogram and the visualization in the rear view mirror of a car created by C. Roessig et al. in the related work section. In the tool, he worked on handling the data from the first sensor, showing object id's in the visual, making objects clickable in the visual, showing the time series of an object, filters, and overlaying a map over the data. He also did the implementation of the visualization of deviations in the sensor data, though the ideas of how to visualize it was joint work. Furthermore, he worked on handling the different windows of the application and the communication between them. He also worked on the creation of the visuals and creating objects based on the data, as well as loading textures for the ground grid, vehicles and the sky.

Rajesh Thangaswamy wrote about the general background of Data Visualization in the background section and about the visualization system for spatial sensor data by M. Tönnis et al., visualization system for a Massive Multiplayer Online (MMO) virtual regatta using a virtual globe by Llorach et al. and context based dynamic visualization system for sensor data that varies in space and time by Stampach et al. In the development of the tool, he worked on handling the data from the second sensor and visualizing it. He also worked with extraction of GPS data, lane data and visualize it. Furthermore, he worked on the slider bar to show the visualization status, the player controls to control the visualization. He also worked with taking screenshot of the visualization and record video features of the tool. He also worked with the general text view and GUI of the tool and also with the creation of visuals in the bird's eye view together with other student.