AD-EYE

Implementation of an autonomous driving platform

VICTOR HANEFORS, CARL JENSEN, JONAS JUNGÅKER
FREDRIK LARSSON, HENRIK LIND, ANNA PHAN
JACOB RÖING, MAKSIMS SVJATOHA, ALMIDA WINQUIST DE VAL

MF2059, Mechatronics Advanced Course
Supervisor: Naveen Mohan
Examiner: Björn Möller

Abstract

Autonomous driving has seen explosive growth in the last decade. As part of this development researchers at the mechatronics department at KTH Royal Institute of Technology, has developed an autonomous driving platform called AD-EYE. The AD-EYE platform has been developed in simulation environment called Prescan. In order for the autonomous driving system to work in real life applications, there is a need for highly detailed 3D-maps, also known as point cloud maps. This report follows the development of creating such a point cloud map of KTH campus in Stockholm. A combination of different sensors were put together, including a Light Detection and Ranging (Lidar), Inertial Measurement Unit (IMU) and GPS. The data from the sensors was then recorded into a ROS-bag and finally converted into a point cloud map. This report also covers the integration of the AD-EYE platform onto a Research Concept Vehicle (RCV) developed by researchers at Integrated Transport Research Lab (ITRL), KTH. This integration was done using the point cloud map created in the previous section. Finally, the RCV was driven autonomously a short distance at KTH campus using the AD-EYE platform.

Acknowledgements

We would like to thank Naveen Mohan for his guidance, help and mentoring through out this project, Maxime Sainte Catherine for support with the AD-EYE platform and Björn Möller and Vicky Derbyshire for administrating the Mechatronics Advanced course. A further thank you goes out to the expert panel, including Daniel Frede, Lars Hässler, Lars Svensson, Tobias Vahlne, Lei Feng and Martin Törngren, for feedback and sharing of their professional experience during the design reviews through out the course. In addition, we would like to thank Gustav Sten for sharing experience relevant to the project and to Chirag Savant and ITRL for lending premises, tools and components for the project. We would also like to thank Robbin Hellström at 3D-interactive for sharing his expertise and providing us with highly detailed maps of KTH campus that were used in comparison with our own solution. Finally, we would like to thank the KTH Digitization platform for partly financing this project.

Contents

List of Figures

List of Tables

Acronyms

API Application programming interface.

AWS Amazon Web Services.

C2C Cloud to Cloud.

CAD Computer-aided Design..

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

GT Greedy Triangulation.

GUI Graphical User Interface.

ICP Iterative Closest Point.

IMU Inertial Measurement Unit.

ITRL Integrated Transport Research Lab.

Lidar Light Detection and Ranging.

M3C2 Multiscale Model to Model Cloud Comparison method.

MC Marching Cubes.

NDT Normal Distributions Transform.

PCD Point Cloud Data.

PCL Point Cloud Library.

RCV Research Concept Vehicle.

ROS Robot Operating System.

RPM Revolutions Per Minute.

UDP User Datagram Protocol.

WBS Work Breakdown Structure.

Nomenclature

ω	Angular velocity [rad/s]
φ	Steering angle [rad]
R	Turning radius [m]
v	Linear velocity [m/s]

Chapter 1

Introduction

1.1 Background

Today there are many autonomous vehicles being engineered and developed with the hope of driving on the streets in the future, for example Tesla, Waymo and Volvo [1, 2, 3]. AD-EYE is an autonomous driving platform currently being developed by researchers at KTH and funded by stakeholders such as Vinnova, Scania Commercial Vehicles A.B. and QRTECH A.B. The aim of this platform is to combine simulations with hardware in the loop to create realistic scenarios for autonomous driving [4, 5]. This allows researchers and engineers to test and validate the autonomous systems, such as sensor configuration, path planning and safety systems, in one platform [5].

The autonomous driving system is designed to be implemented on a real life vehicle, in this case the Research Concept Vehicle (RCV) at KTH, seen in Figure 1.1. For this to be possible, the platform has to have a real world reference, a 3D map, instead of the simulated world. The AD-EYE platform will then use the map for trajectory planning and, together with real time input from sensors on the vehicle, drive autonomously.

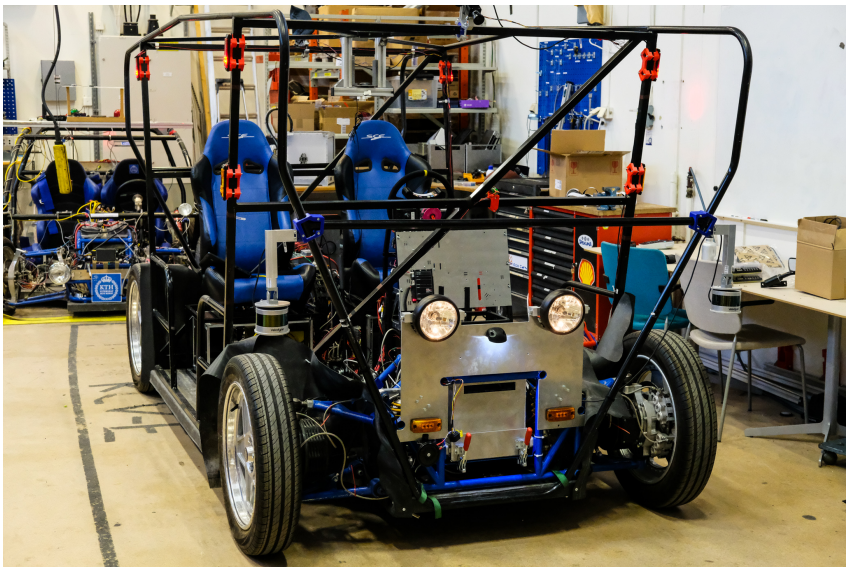

Figure 1.1. The RCV developed at Integrated Transport Research Lab (ITRL).

Different ways of obtaining 3D maps exist and use different sensors and programs to register and render a map of the real world. State of the art research and a prestudy on the subject were conducted prior to this implementation project, to decide on the appropriate course of action for creating such a 3D map [6].

The team behind the AD-EYE platform also has a collaboration with 3D Interactive, a company that creates 3D point cloud maps through self developed methods [7]. This method has millimeter precision and requires local scanning and georeferencing at every seven meters to obtain the point cloud maps. The maps are then stitched together into a coherent larger map. This implies that the method is too time consuming for regular updates of the map, however can be used for comparisons or as ground truth.

1.2 Delimitations

The scope of this project involves transferring the autonomous driving of the AD-EYE platform to the RCV, and not to develop the autonomous driving itself or further develop the safety features of the AD-EYE platform.

The mapping done within this project is limited to the area of the KTH campus. The goal includes one final solution for mapping, developed within this project and not a pre-existing solution or pre-fabricated map. The reason of this is that this project is supposed to not only develop a finished solution for mapping, but to be a teaching opportunity for participants within the advanced mechatronics course at

KTH. Moreover, a stakeholder requirement is that the map should be obtained as a ROS-bag[1], which limits the method of creating the 3D map. However, this project collaborates with 3D Interactive, a company that has their own tools and method for creating point cloud 3D maps and have mapped parts of the KTH campus. This collaboration means that 3D Interactive's map can be tried in the AD-EYE platform and compared with the project's created map, but not entirely substitute the team's task of creating a mapping solution.

The time limitation of this project is approximately 18 weeks, the length of the fall semester, and done within 15 credits, corresponding to a 50 % workload during this period of time. Furthermore, the budget given by KTH for this project is 50 000 SEK. Except of this amount, components can be borrowed from KTH.

When developing the sensor mount, it should be a solution that fits most vehicles and that can be reused regularly, without causing harm to the vehicle. Furthermore, this solution can not be a fully pre-fabricated construction, but should be designed, constructed and tested within the scope of the project.

1.3 Project description

The project aims to further develop the AD-EYE platform by enabling it with tools to generate point cloud maps from a real world environment. These point cloud maps should later be used in real life autonomous driving scenarios. The point cloud maps should also work as a base for creating solid structure meshes that will be used in simulated experiments for more realistic modeling of autonomous driving and the project should also enable a solution for creating these types of meshes.

1.3.1 Goals and impacts of the project

One of the goals of the project is to create a point cloud map of KTH campus and use this point cloud with the AD-EYE platform to control and drive a RCV autonomously. The other goal is to convert this point cloud in to a solid structure mesh.

The impact of successfully driving the RCV using the AD-EYE platform will show the potential of AD-EYE and work as a proof of concept that the platform works in a real life scenario. Also, having a tool for creating point cloud maps and evaluating this tool enables developers to create point cloud maps, if needed, with less effort and cost and further allows developer to focus thier time on platform related topics.

1.3.2 Deliverables and most important issues

At the end of the project the following should be deliverable:

[1]Please see section 2.3.1 for more information regarding ROS-bags

- A point cloud map of KTH campus in the format of .pcd
- A solid structure mesh based on the point cloud map
- Instructions and produced scripts and programs for reproducing and generate new point cloud maps and meshes
- A ROS-package for integrating the AD-EYE platform with the RCV

The most important part of the project is the creation of a point cloud map. The motivation behind this is that without a point cloud map, the AD-EYE platform can not utilize its localisation algorithms and further a vital condition for autonomously driving the RCV would not be met. Even if the project would not meet the goal of driving the RCV autonomously, it would still be beneficial for future development to have access to a created point cloud map as well as a method or tool to generate these point clouds maps since having these prerequisites for driving autonomously would greatly minimize the time to get the RCV out on the road.

1.3.3 The teams strengths

The team that have worked on this project consist of 9 mechatronics students from KTH. Every student has knowledge in electronics, mechanical design, control theory and computer science. Beyond this, the collected knowledge in the team expands to devops, deep learning, machine learning, computer security, economics and management. With the collected competence and knowledge regarding these fields and their limitations, solutions for achieving the goals mentioned in 1.3.1 have been possible.

1.4 Requirements

Within the SOTA and pre study phase of the project, the stakeholder and technical requirements listed below where derived [6]. For this project, as well as the SOTA, the stakeholders are Naveen Mohan, ITRL - Integrated Transport Research Lab and MECS - Mechatronics Embedded Control Systems at KTH.

1.4.1 Stakeholder Requirements

1. Create a detailed point cloud map of the KTH campus.
2. Autonomous driving from point A to B on KTH campus.
3. Design a custom cluster of sensors needed to record maps of KTH, mounted on a commercial vehicle.
4. Design a custom mount for the NVIDIA Drive PX2 to be placed on the RCV.

5. Create a vectorized map of the KTH campus.
6. Keep the project within the budget.

1.4.2 Technical Requirements

1. 1.1. The point cloud map shall be recorded into a ROS-bag.
   1.2. The point cloud map shall be exported as .pcd format.
   1.3. The point cloud shall be generated by utilizing Lidar and IMU.
   1.4. The ROS-bag should include GPS-data.
   1.5. The sensor data should include camera pictures, to localize traffic signs etc.
   1.6. A method for filtering away noise and disturbances in the recorded ROS-bag should be implemented.
   1.7. The point cloud should have sufficient accuracy in order for the RCV to utilize ndt-matching.
   1.8. The Lidar should be calibrated with a rotational speed, angle and return type that provides the desired precision.
   1.9. The driving speed should be optimized to provide the desired precision.
   1.10. The point cloud shall be converted into a mesh sufficient enough for the simulation to run.
   1.11. The mesh shall be exported as a .dae-file.
2. 2.1. All subsystems of the AD-EYE platform should be tested in simulation before driving autonomously in real life.
   2.2. Performing final self driving tests shall be done at KTH in conjunction with the safety plan set at ITRL.
   2.3. The RCV shall provide the AD-EYE platform with its GPS position.
   2.4. The RCV shall provide the AD-EYE platform with its current angular velocity.
   2.5. The RCV shall provide the AD-EYE platform with its current linear velocity.
   2.6. The RCV shall provide the AD-EYE platform with IMU data.
   2.7. The AD-EYE platform shall provide the RCV with a reference angular velocity.
   2.8. The AD-EYE platform shall provide the RCV with a reference linear velocity.
   2.9. The communication between the RCV and the AD-EYE platform should follow the UDP protocol.

2.10. The computer that runs the AD-EYE platform shall be powered by the RCV.

2.11. The RCV shall provide the AD-EYE platform with data from at least one Lidar.

2.12. The RCV should provide the AD-EYE platform with camera imaging for its detection.

2.13. The RCV shall provide the AD-EYE platform with its odometry position.

2.14. The RCV shall handle linear velocity and angular velocity as control inputs.

2.15. AD-EYE should be able to use Lidar data for localisation.

2.16. AD-EYE should be able to use GPS data for localisation.

2.17. AD-EYE should be able to use odometry data for localisation.

3. 3.1. The sensor mount shall fit most types of car roofs or windshields.

3.2. The sensor mount shall not cause any damage to the car.

3.3. The sensor mount shall protect the sensors from any external driving forces that might harm them. Handling a 30 N load in all directions while driving for at least 30 minutes is required.

3.4. The Lidar shall be placed so that it can operate without hindering the field of view.

3.5. The sensor rig, including sensors, should be water resistant.

3.6. The water sensitive components shall be placed within the vehicle.

4. 4.1. The mount shall protect the Nvidia Drive PX2 from any external driving forces that might harm the card.

4.2. The mount shall protect the Nvidia Drive PX2 from any water damage in case of precipitation.

4.3. The mount shall make sure that the Nvidia Drive PX2 is not overheated during operation, maximum 60 °C.

4.4. The Nvidia Drive PX2 shall be placed so that it does not interfere with existing RCV components.

5. 5.1. The vector map shall be exported as .csv format.

5.2. The vector map shall be compatible to run on the AD-EYE platform and give global path planner the ability to create a trajectory from point A to point B.

6. 6.1. The budget shall not exceed 50 000 SEK, hence as many components as possible should be components that are already available for use at ITRL or MECS.

1.5 Report disposition

The report is built up by a number of chapters where the main goal of each chapter is to aid the reader in understanding the topics of this project and further be able to understand the results, discussion and conclusion presented in the last chapters. In the subsections 1.5.1 to 1.5.7 bellow, the reader will find a brief explanation of each section.

1.5.1 Theory

In this chapter the theory behind relevant fields and topics are presented in a more detailed manner. This section should work as reference for the reader if certain topics may be hard to grasp or if the reader would like to get a deeper understanding of these topics.

1.5.2 Design generation

This chapter presents the process of generating the original concepts that the final solution was based upon. The reader will here find what various concepts the team considered and also motivations for the final concepts.

1.5.3 Work Methodology and management

This chapter focus on the projects structures and how the team worked throughout the project in terms of management, planning and methods used throughout the project.

1.5.4 Implementation

In the Implementation chapter, the reader will find what the team did in order to achieve the goals presented in 1.3.1. Here the workflow flow integrating the AD-EYE platform with the RCV is presented as well as the procedure for creating a point cloud map, a vector map and solid structure mesh.

1.5.5 Verification and validation

In this chapter the reader will find the analysis of the point cloud maps as well as more detailed test protocols for testing the sensor rig that was used to map the KTH campus. Also in this chapter the reader can find additional explanation of how the RCV was tested to verify the integration of the AD-EYE platform with the RCV.

1.5.6 Results and discussion

In this chapter the results of the point cloud/vector maps, the mesh, the sensor rig and the integration of the AD-EYE platform with the RCV are presented and tied to the requirements as well as the discussion.

In the discussion section, the work on the RCV, point cloud map and vector map are discussed in greater detail. Changes to the original concept are discussed and motivated as well as issues that was encountered throughout the project.

1.5.7 Conclusion

In the conclusion chapter, the conclusion drawn from the results and discussion are presented as well as some final thoughts.

Chapter 2

Theory

This chapter is based on the state of the art research conducted as a part of the pre study for this project and explains the theory relevant for the implementation [6].

2.1 Sensors

2.1.1 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is a combination of an accelerometer, gyroscope and sometimes a magnetometer. Used in robotics, sensor stabilization and navigation, the IMU calculates linear and angular acceleration by measuring the forces applied to it. If a magnetometer is added, it can also measure orientation by measuring angular differences in its own and the earth's magnetic field. Unlike the early variants, modern IMUs measure all three axes of direction and rotation. Modern IMUs are also commonly integrated into larger sensor modules that feature additional functionality and include other sensors [8].

2.1.2 Global Positioning System

A GPS sensor is a receiver that utilizes the Global Positioning System (GPS), which itself is a type of Global Navigation Satellite System (GNSS). The receiver requires unobstructed line of sight to four or more GPS satellites. These satellites send geolocation and time data to the receivers using radio waves [9]. This means that relatively large objects, such as large buildings or mountains, can potentially block GPS signals.

2.1.3 Lidar

A Light Detection and Ranging (Lidar) measures distances using laser beams. It does so by measuring the time it takes to send a laser beam, have it bounce or reflect off a surface and return to a sensor attached to the Lidar. By measuring

hundreds or thousands of laser beams in this fashion, a mapping of various points at different distances from the Lidar is created - called a point cloud. For this map to be accurate, the Lidar requires calibration before use [10, 11].

Furthermore, a Lidar can support different return modes of the laser. The beams cover areas, due to the beam divergence, meaning that the laser becomes larger based on the distance it travels. This means that a part of the area might be reflected by an object close and the rest of the area will be reflected on an object far away. The return mode strongest will provide data about the returned section of laser beams that is strongest reflected. Whereas the dual mode will provide data of the strongest reflected laser beam as well as data of the second strongest return [12].

2.2 Sensor mounting

When it comes to mounting sensors on a car, it is of utmost importance that the mounting is rigid enough to withstand vibrations and all other driving forces that might harm the sensors and effect the measurements. Since the Lidar needs unobstructed field of view, it is also important to have an open design that can be mounted on top of the car. Furthermore, it is good to have a mobile mount that can be easily mounted and dismounted on the car. Lastly, the mount should not harm the car and should preferably not be specific to a certain type of car. Suction cups and roof racks are common ways of mounting sensors to a car when it comes to mapping. The pros and cons of each solution is discussed in the SOTA-report [6] and in Section 3.1.

2.3 Mapping

It is of high importance that the 3D map used for the autonomous driving of the RCV together with the AD-EYE platform is of sufficient quality. One solution of creating 3D maps is through recording a ROS-bag and with Autoware process it into a file of format Point Cloud Data (PCD). This can further be processed with filtering and meshing techniques as well as be used to create a vector map.

2.3.1 ROS-bag

ROS-bag is a format within Robot Operating System (ROS), that records data messages being sent to specified ROS-topics during recording and giving it a time stamp. The ROS-bag itself is of format .bag and can be filled with data of different formats, for example point data from a Lidar. This ROS-bag can later on be replayed and the data from every moment in time can be processed in different ways [13].

Examples of packages that can handle ROS-bags are ROS-bag and rqt_bag. The package ROS-bag is a command-line tool which has a Application programming interface (API) to read and write in C++ and Python. rqt_bag has a Graphical User Interface (GUI) to playback the .bag-file, show and export messages and display images [13, 14].

2.3.2 Point cloud map

Autoware's NDT mapping

Autoware is the program which is used in the AD-EYE platform and can be used for creating vector maps and point cloud maps from a recorded ROS-bag with point data. It is an open source software built on ROS, that includes tools for data processing and rendering 3D maps, as well as path planning algorithms [15, 16].

To create the 3D point cloud maps, Autoware filters and processes the point cloud data with techniques such as voxel filtering. With this method, the raw points are converted into voxels and then used with Normal Distributions Transform (NDT) [17]. Autoware's node NDT mapping uses the NDT algorithm to take every point cloud recorded in time and compare it to the next one to compute the sensor's, often Lidar, position in the point cloud map. Hence, the point clouds can be stitched together to one solid point cloud map in the .pcd format [17]. Data from an IMU and GPS can preferably be registered together with the point cloud to give more location information in the 3D map [16].

Cartographer

Google has developed a tool for creating large 3D maps that is called Cartographer. It is similar to Autoware's NDT mapping in the manner that it is an open source tool based on ROS. It likewise requires point cloud data, preferably from Lidar, and filters it with voxel filters. It continuously calculates the Lidar's position with respect to the entire point cloud and generates a PCD-map [18].

Cartographer uses small segments of laser scans and imports these into sub-maps. Every sub-map has a probability grid, where each grid frame is evaluated to be a part of a solid object or not. The sub-maps are stitched together into a greater map. To avoid accumulated errors that might appear when stitching several maps together, a pose optimization is implemented [19]. Cartographer maps can be implemented with IMU data, which can help with reducing movements and noise during the data recording. Furthermore, the Cartographer tool is mainly recommended for indoor use [19].

CloudCompare

CloudCompare is an open source software for handling point clouds [20]. It does not generate any point clouds, but has many useful tools for processing them such

as segmentation, merging of multiple point clouds and distance calculation between points. The latter can also be used to measure an average distance between points in two separate clouds. For example if one cloud is known to be very precise and corresponds closely to reality, the accuracy of the second cloud can be evaluated using the Cloud to Cloud (C2C) [21]. Furthermore, the registration error can be controlled using the Iterative Closest Point (ICP) [22]. It also has tools for surface reconstruction using Delauney Triangulation, further mentioned in section 2.3.4.

2.3.3 Vector map

OpenStreetMap

OpenStreetMap is an online tool that provides editable maps. Contributors have gathered information about terrain with GPS, cameras and similar devices and added into the OpenStreetMap data base together with satellite images. OpenStreetMap is a non profit organisation that processed information into downloadable maps [23].

The differences between a vector map based on OpenStreetMap and the recorded PCD map based on Lidar can contribute with disturbances while running localization and path planning in AD-EYE. The OpenStreetMap is furthermore two dimensional, hence the real life distances might be longer than what the map shows. However, this error is something that can be edited. Even though these factors effect, OpenStreetMap's maps have been used successfully by the AD-EYE researchers in the simulation environment, hence having the possibility to work even with the RCV.

Autoware

Autoware's Vector Map Builder is a tool within Autoware that uses a point cloud map and thereafter lets the user manually mark vectors. The principle is that the point cloud map can be uploaded and vectors can be drawn to define different objects in the map, for example lanes, stop lines, signs and traffic lights. The program converts the marked point cloud into a vector map of format .csv, which is then used by the AD-EYE platform [24]. This solution has a advantage in that it is based on the actual point cloud to be used in the AD-EYE platform, even though it is less suitable in large scale due to the manual work.

2.3.4 Surface reconstruction algorithms for meshes

There are many different ways to convert a point cloud map into a solid structure mesh. The basic theory is to apply surface reconstruction on a point cloud using an algorithm.

Today there are existing libraries in a variety of languages that provide easy implementation of state of the art algorithms, where Point Cloud Library (PCL) in

C++ is one of them. The library has been developed together with robotics and perception researchers and provides most of the more common algorithms for surface reconstruction, as well as filtering of point clouds [25, 26]. PCL is furthermore compatible with ROS, where a ROS-bag can be processed with nodes from the ROS PCL-package [25, 27].

Greedy Triangulation

Greedy Triangulation (GT) is based on keeping a record of possible points that can be connected at a later stage. Short compatible edges are then added between points of the cloud with the constraint that the edges do not cross previously formed edges between other points. One GT approach is computing all distances, sorting them and examining them in regard to their length and the compatibility constraint. The shortest distance where a compatible edge can be formed is chosen. An example can be shown in Figure 2.1. Although it is simple at its core, it suffers from being unreliable. This is due to over-representation of planar points [28].

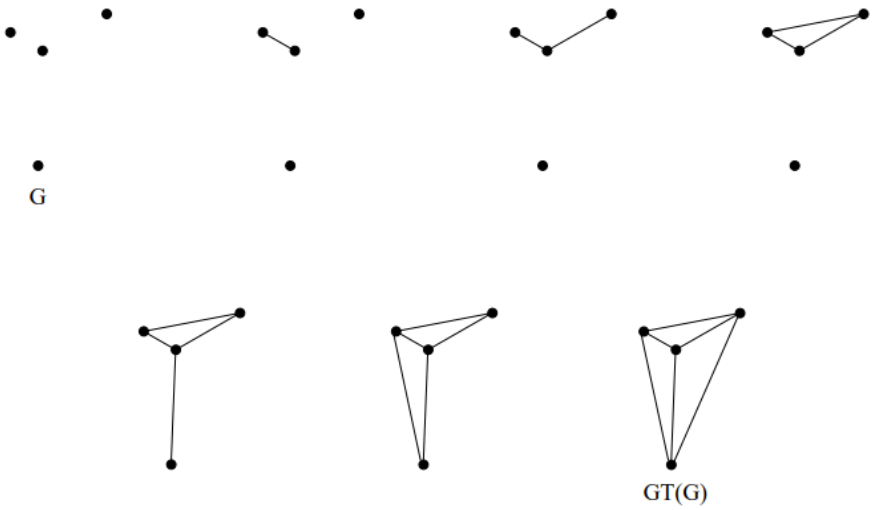

Figure 2.1. Greedy Triangulations illustrated. G is the point cloud and GT(G) is the final triangulation [29].

In the Point Cloud Library, GT allows for specifying the features of focus. This can be done with parameters such as neighbourhood size of the search, the search radius and the angle between surfaces [28].

Marching Cubes Algorithm

Marching Cubes (MC) is an algorithm that uses voxel data and creates an isosurface [30]. Assume a trivariate scalar field f(x,y,z) and the variable isovalue, denoted v.

An isosurface is a surface representing all locations in a three-dimensional (3D) space where f = v [31]. Marching cubes iterates, or marches, through a scalar field, each time taking eight neighbor locations in the form of a cube. The vertices of these cubes with a value less than the isovalue are classified as being "negative" and vertices with a value larger than the isovalue are classfied as being "positive". The algorithm determines the polygon(s) needed to represent the part of the isosurface that passes through the cube. Positive vertices are classified as being part of the isosurface while negative ones are not [30].

Mathematically, there are $2^8 = 256$ possible polygon configurations within the cube. However many of the configurations are mere images or rotations of each other. This means when MC looks at the boundaries at each voxel, it does a lookup into one of 15 different cases. The look up table is seen in Figure 2.2 below. The mesh that corresponds with the looked up case is added in place of the voxel. The output from one iteration is an 8-bit integer, one bit for each of the polygon indices. The desired surface is fused together by individual polygons and results in a set of mesh triangles that approximate the mesh that the point cloud was created from [30].

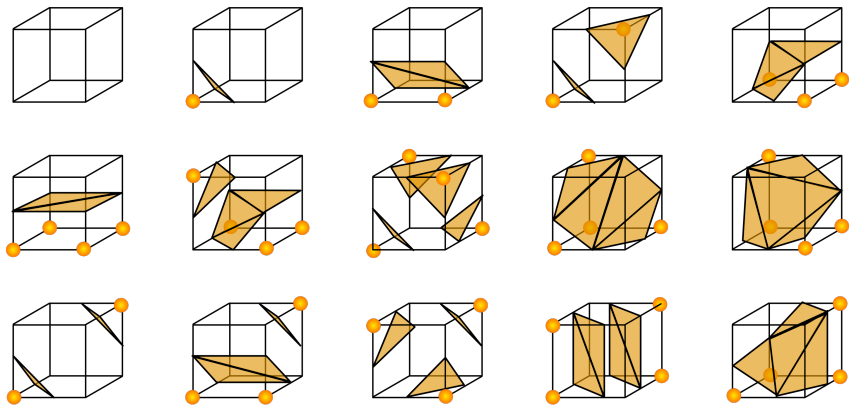

Figure 2.2. The originally published 15 cube configurations by Jmtrivial.

Poisson Surface Reconstruction

Poisson Surface Reconstruction is obtained by computing the indicator function, χ, and extracting the appropriate isosurface. The indicator function is defined as one at points inside the model and zero at points outside. Since it is constant for most points, this results in the gradient being zero for that point. These points are classified as being outside the isosurface. Points near the surface is equal to the inward surface normal. The oriented point samples, denoted χ, is samples of the indicator gradient. The Poisson function is described as

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}$$

where $\vec{V}$ is a vector field and χ is a scalar function whose Laplacian (divergence of gradient) equals the divergence of the vector field. Poisson reconstruction in 2D is illustrated in Figure 2.3 [32].

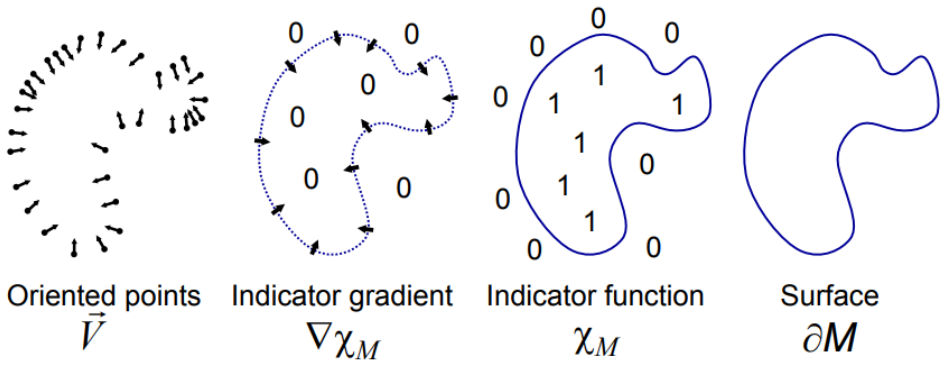

Figure 2.3. Poisson Surface Reconstruction [32].

The Poisson equation $\Delta u = f$ where f is periodic, can be solved using the Fourier transform. The Fourier series expansion is thus $-\mid \zeta \mid^2 \hat{u}(\zeta) = \hat{f}(\zeta)$, or equivalently $\hat{u}(\zeta) = \frac{-1}{|\zeta|^2}\hat{f}(\zeta)$. In our case we use $\hat{\chi} = \frac{-1}{|\zeta|^2}\nabla \cdot \vec{V}$.

Surface reconstruction with the Poisson equation offers a number of advantages. Unlike previously mentioned surface fitting methods that segments the data and incorporates partitioning of data, local fitting and blending, Poisson reconstruction is a global solution that considers all data at once. Thus it creates a smooth surfaces that robustly approximate noisy data all while, the Poisson problem is locally supported and have a hierarchy of locally supported functions, that reduces its solution to a well-conditioned sparse linear system [32].

2.3.5 Algorithms for filtering

The purpose of filtering the point cloud map is to remove outlying points and reduce noise. PCL can be used for filtering of point clouds and offer several different C++ functions, based on different algorithms, to filter with [25].

Statistical outlier removal

An example of the statistical outlier removal feature is shown in Figure 2.4, where the rightmost picture has eliminated the outliers of a PCD-file. With this algorithm, the mean distance to every data point's neighbours is calculated. The points with a mean distance outside of a specified interval, that is defined with respect to a statistical Gaussian distribution, are excluded [33].

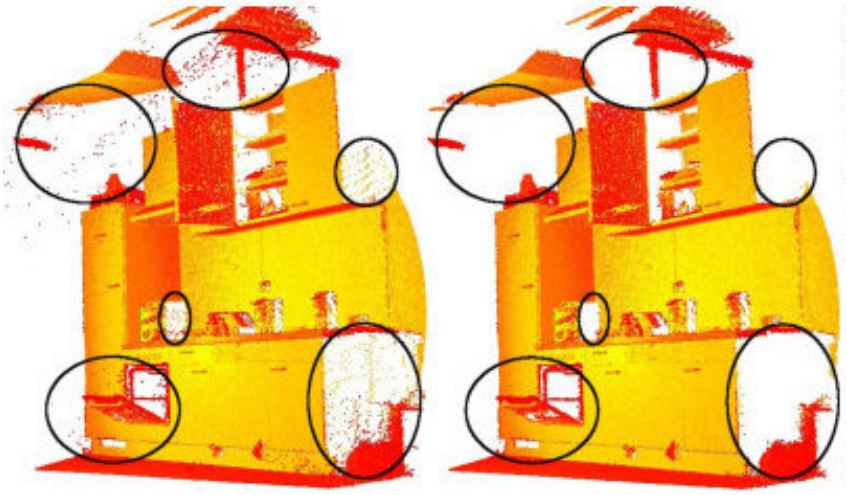

Figure 2.4. Point cloud filtered with statistical removal [33].

Pass through filter

The pass through filter found in PCL is used to remove certain areas of a point cloud. The user defines an interval on every axis x, y and z, where points should be saved, and all point outside of this area are removed [34]. In Figure 2.5, an example is shown of the implementation of PCL's pass through filter, where certain areas of points are removed.

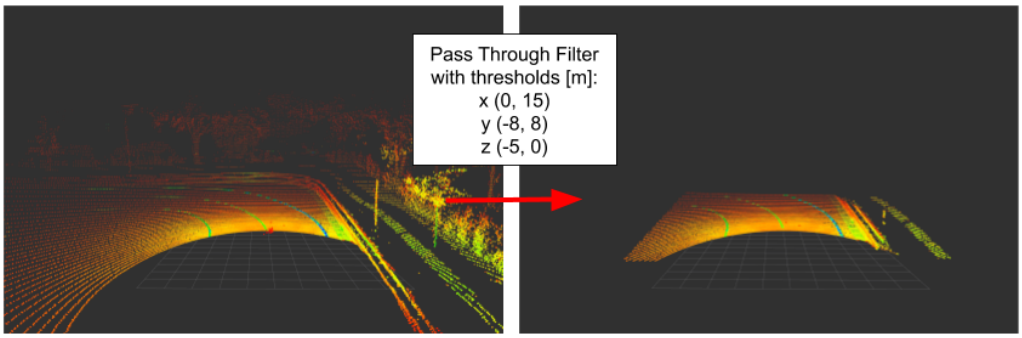

Figure 2.5. PCL's pass through filter [35].

Voxel grid filter

A voxel grid filter is a down-sampling filter that functions by generating small 3D boxes (known as voxels) that form a grid over the point cloud [36]. The data points within each voxel can then be approximated into a center point. Regulating the down-sampling amount is done by controlling the size of the voxels, as a larger voxel

will approximate more points at once thus also removing more points. The filter technique is visualized in figure 2.6. This filtering technique is useful for removing unwanted clusters of points at the cost of some accuracy.

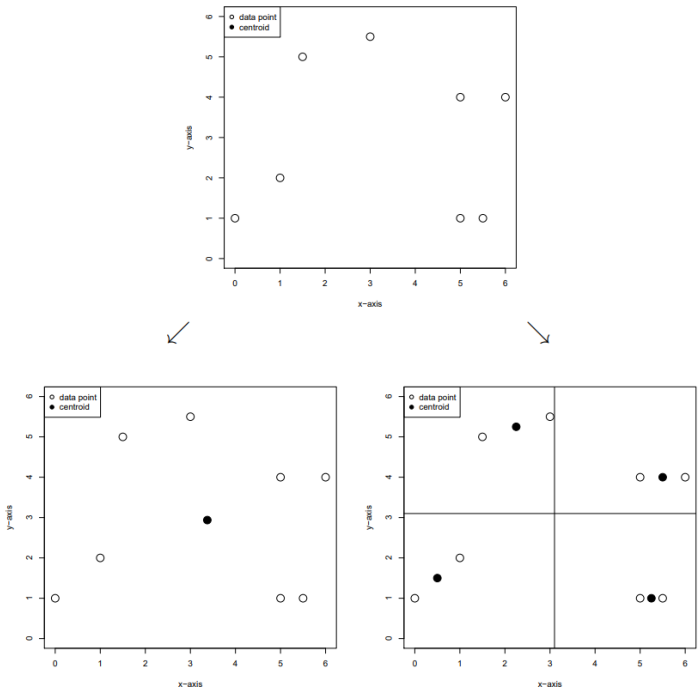

Figure 2.6. Visualization of a voxel grid filter [36]. Left with a larger voxel and right with smaller voxels. Black dots represent the new data points.

2.4 AD-EYE

In Figure 2.7 a schematic overview of the AD-EYE platform is shown. The system is based on Autoware's autonomous driving intelligence, with perception and planning, that together with KTH's developed safety channel leads to decision making for the vehicle.

The nominal channel found in figure 2.7 handles perceptions for localisation, object detection and visual recognition. The channel is also in charge of global planning.

The safety channel on the other hand handles decisions related to safety planning, such as braking and evade if encountered with an anomaly like a pedestrian walking in front of the vehicle.

The AD-EYE platform needs to communicate with either a simulated environment or a vehicle driving in a real environment. The simulated environment that is used

here is created in Prescan (see section 2.4.1). The vehicle that is available to use for real life implementation of the platform is a Research Concept Vehicle.

On the RCV, a computer runs compiled Simulink code to control the vehicle and process the sensor inputs and autonomous driving inputs [37]. In Figure 2.7, the simulation world will be substituted with the real world when the AD-EYE platform is used on the RCV.

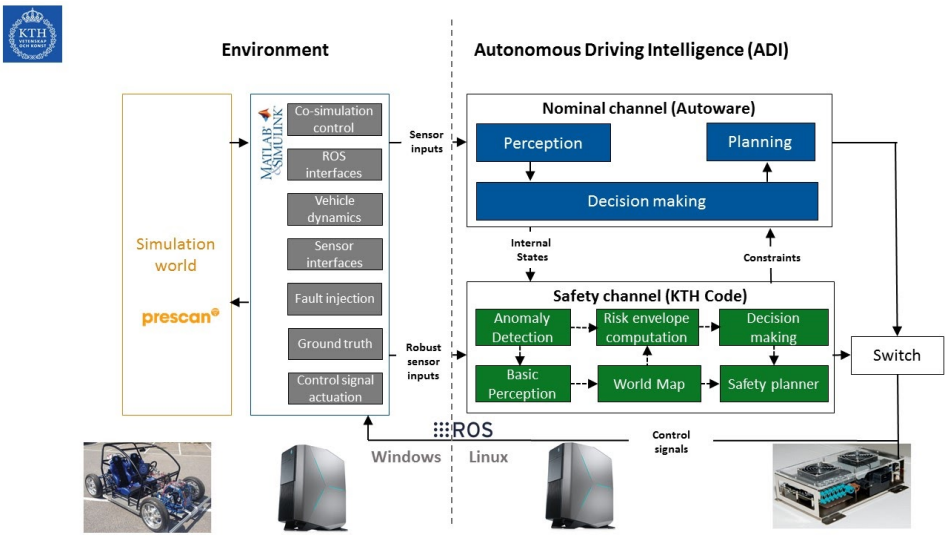

Figure 2.7. Overview of the AD-EYE architecture [37].

2.4.1 Prescan simulation

Prescan is a program by Siemens with purpose to offer a simulation environment. This simulated environment could be used by an autonomous system like AD-EYE. The simulation is physics-based and makes it possible to test safety and reliability, by simulating sensors, vehicle models and driving scenarios. Prescan makes it possible to create own scenarios and worlds, to optimize and develop autonomous vehicle functionalities [38].

Prescan is compatible with Matlab's Simulink, and communicates, in the setup shown in figure 2.7, with Autoware and AD-EYE to drive a simulated vehicle around in the simulated world. The world can be constructed with road segments, trees and buildings as well as filled with moving pedestrians, bicyclists and cars. Solid worlds that are not generated in Prescan can also be imported and used. Data from the simulation, both sensor data and ground truth data, can be sent to the autonomous driving function from Prescan. Simulink acts as the middle hand between Prescan and AD-EYE/Autoware to handle communication and synchronization of time so simulation can be run at a desired time, and not real time [5].

2.5 Research Concept Vehicle

The Research Concept Vehicle developed at ITRL has a fully functional driving system. It is based on a Simulink model that is compiled to run in real time on the RCV's computer, a MicroAutoBox 1401/1513 developed by dSpace [39], that has a GUI program named dSpace Control Desk. The driving mode can be switched from manual mode to automatic mode. When automatic mode is activated the MicroAutoBox on the RCV listens to incoming User Datagram Protocol (UDP) messages. User Datagram Protocol (UDP) is suitable for time-sensitive applications since it has no retransmission delays [40]. Since UDP avoids the overhead associated with connections, error checks and the retransmission of missing data, it is highly suitable for real-time applications like the RCV. The UDP message needs a specified source port and ip-adress, destination port and ip-adress, as well as number of bytes to be sent [40].

In order to drive the RCV in automatic mode the RCV needs values for throttle, break and steering angle. Throttle and break should be given values in the range 0-1 and steering angle in the range $-\pi - \pi$ radians.

Chapter 3

Design generation

3.1 Concepts sensor mount

3.1.1 Concept 1: Roof rack

The use of a roof rack for mounting sensors on vehicles with the pre-existing roof rack attachment points is not new. This concept is based on securing the sensors to existing racks on a car, either by screwing or clamping. These connections can easily be modelled, for example using a simple framework in metal, 3D-printed connectors, screws and threads. For vehicles with a naked roof, Thule has modular load carrier feet with a patented clamping technology. Some examples are Thule Edge Clamp, Thule Evo Clamp and Thule Rapid System 754 [41]. These are easily installed to the side of the vehicle in a set of pairs and later accompanied by a load bar. It is a robust solution that does not require much maintenance once installed. Humidity is not a concern, and benefits include a rigid connection between mount and car. One downside using this concept is that if the car does not have roof rack attachments, the solution relies on manufacturers' naked roof rack products, and these will not fit all cars. This limits the option of what car that can be used for mapping.

3.1.2 Concept 2: Suction cups

Using negative fluid pressure of air, a suction cup creates a partial vacuum to adhere to nonporous surfaces. The material for suction cups usually consists of rubber or soft plastic which has some dampening properties. One of the main benefits using suction cups is that it can be mounted on any nonporous surface, and as many modern car roofs consists of various materials this will in most cases work well. If it is not possible to mount it on the roof, then the glass on the windshield is another good mounting option.

Suction cups that could fulfill the requirements are tripods for equipment like GoPro cameras, for example Montana GoPro tripod that can handle a load of 700 g at a

speed of 250 km/h [42].

3.1.3 Concept 3: Magnetic mount

One other way of mounting components on to a vehicle is by using magnets. Magnets are predictable, but the strength is strongly dependant on the right magnet, and the fit. Benefits include an easy mounting, and constant behaviour when mounted on a flat surface. However, there are several issues that might make this type of mount not feasible for this project. One issue is that many modern cars use plastic on the external parts of the car, making magnetic mounts unusable. Another issue is that the magnetic field created by the magnets might interfere with the sensors. Since different cars might be used, the concave shape of the mounting surface will be unknown, and could lead to unpredictable behaviour.

3.1.4 Choice of mounting concept

A possible vehicle for disposal of this project, to use while mapping, is ITRL's Renault Twizy. It is a beneficial car, since it is modest in size and is built on an electric motor for propulsion, causing little sound and vibrations.

The concept that is most feasible with this car is the second concept that is based on suction cups. The Twizy does not come with roof racks and requires modification to add such, and it does have a plastic non-magnetic roof, hence eliminating the other two concepts. Meanwhile, the suction cups could possibly be attached to both the roof and the glass window shield. The suction cup based concept is furthermore a preference due to the fact that it is the most versatile solution, that fits the largest amount of car models, an advantage even for future mapping with other vehicles. A conceptual rendered CAD model, with sensors included, can be viewed in Figure 3.1 and attached to the Twizy in Figure 3.2.

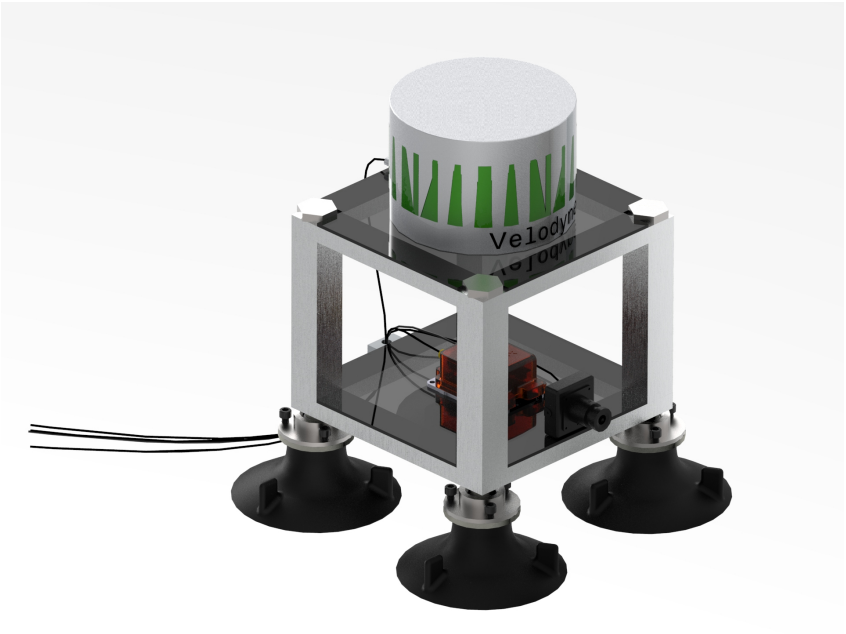

Figure 3.1. Suction cup and sensor rig. Generated with Solid Edge and rendered in Keyshot.

Figure 3.2. Sensor rig mounted on a Renault Twizy. Generated with Solid Edge and rendered in Keyshot.

3.2 Concepts mapping

Two concepts where developed for the mapping solution. One concept is cloud based and the other one is recorded locally. Furthermore, the concepts differ in how the vector map should be created.

Both concepts use Autoware's Normal Distributions Transform (NDT) mapping node as mapping tool. Other possible solutions that were compared with NDT mapping was Cartographer. Autoware's tool is more documented, and since the AD-EYE platform is based on Autoware, it was evaluated as the more appropriate choice.

The sensors chosen for both concepts are the ones recommended to give appropriate inputs to Autoware's NDT mapping tool. These sensors consist of a Lidar, for capturing points to be compared for localisation, a camera to picture the surroundings, an IMU to specify the position of the lidar and lastly a GNSS to obtain coordinate location in the map.

One of the Lidar's with the best range of motion on the market is the Velodyne Lidar VLP-16 and has therefore been chosen. Benefits with this Lidar is that it especially has a good vertical field of view and long range, compared to competitors on the market, and it can work in temperatures below zero which is needed due to the climate of Sweden there this project takes place. Moreover, it is also a component that is for lending at ITRL, which is preferable since buying one would exceed the budget.

The MTi-G-710 was chosen as the GNSS and IMU for the project. It has wide functionality, since it combines both an IMU and a GNSS, and it can be borrowed from ITRL. Compared to similar components, the MTi-G is the best with respect to included features, power consumption and general performance. The only drawback found in the research of picking an IMU is its expensive price, which however is not an issue since it can be borrowed.

Both concepts include filtering and meshing of PCD. The mesh is for use in the Prescan simulation.

3.2.1 Concept 1: Cloud mapping

Figure 3.3 depicts a graphical representation of the cloud mapping concept. It is based on the idea of recording the ROS-bag data directly into an Amazon Web Services (AWS) virtual machine to simultaneously be able to generate the PCD map. The sensors are recorded in ROS with topics that can be subscribed to and the ROS-bag is created in the online accessed ECU. Through a second ECU the NDT mapping creates the map. These calculation would be too power consuming to run on the mapping laptop and are therefore moved to the cloud. A stationary

computer would later create the vector map with Autoware's vector map builder, which would be used in the simulation and on the RCV together with the PCD map.

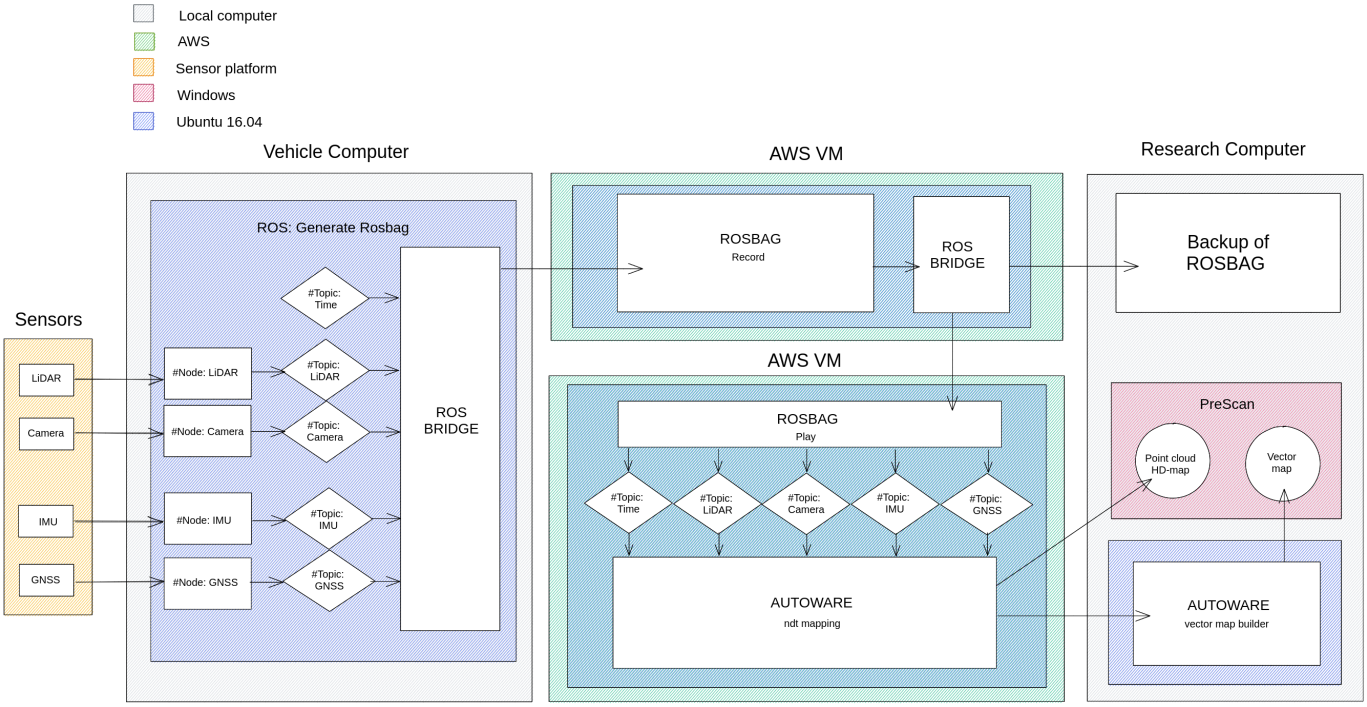

Figure 3.3. A graphical representation of the concept Cloud mapping created using Excalidraw.

3.2.2 Concept 2: Local mapping

The second concept is shown in Figure 3.4. It includes the vehicle computer, that similarly to the first concept reads sensor data, but instead creates the ROS-bag directly on it. The ROS-bag is saved to a SSD memory, which can be transferred to a powerful stationary computer that can process the ROS-bag into a PCD with NDT mapping. The same computer is also used to generate a vector map from OpenStreetMap. The PCD and vector map are then possible to use in Prescan simulation and on the RCV.

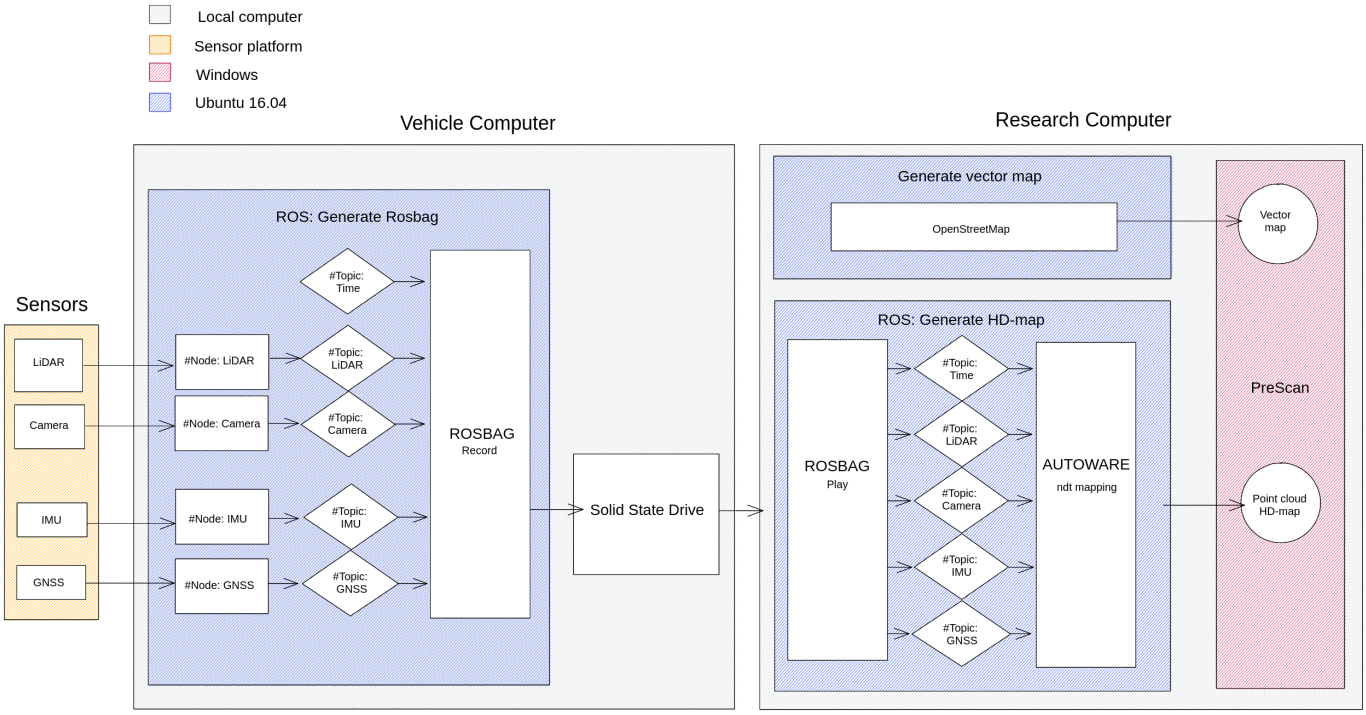

Figure 3.4. A graphical representation of the concept Local mapping created using Excalidraw.

3.2.3 Choice of concept

Weighing pros and cons of the two solutions leads to the choice that the concept local mapping is the one to be further developed and implemented. This solution is simpler, once factor being that no internet connection is needed during mapping. Being able to generate the PCD continuously is a plus, but not a requirement. On the contrary, it requires more technical features to work and communicate together. Another factor is that, even though an SSD adds an additional cost to the budget, the price for the AWS is not pre-determined. Moreover, OpenStreetMap is a tool that has been used with the AD-EYE platform and Prescan earlier, confirming that it could work, and has a simpler user interface than Autoware's vector map builder. Vector map builder also requires vectors to be placed manually, which for a large map can be time consuming.

3.3 Concept PX2 mounting

The RCV is equipped with a box, containing free space and a metal plate that can be pulled in and out of the box, pictured in Figure 3.5. The metal plate is not being used and is possible to modify as desired. The Nvidia Drive PX2 has build in M6 fastening holes and the metal plate can be drilled in to match these.

The box has the size of height 25 cm, length 38 cm and width 29 cm, providing sufficient space for the Drive PX2 to be mounted horizontally. The space is not fully sealed and therefore provides airflow and possibility for connectors and cables to be attached. If necessary, dust filter will be added, along with cooling fans. The RCV is furthermore equipped with 230 V power supply, which the PX2 can be powered by. Since the RCV itself is sensitive to water, the RCV will never be driven outside during poor weather conditions. However, if there unavoidably will be occurrence of rain, the Drive PX2 is protected by the box.

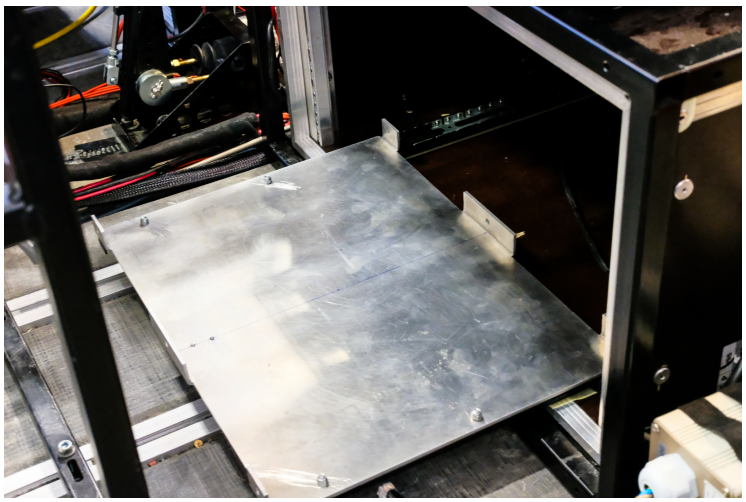

Figure 3.5. Planned space for housing the Nvidia Drive PX2 on the RCV.

Chapter 4

Work methodology and management

4.1 Methodology

The pre study, including state of the art research, planning, literature review, budgeting and concept designing, was done as a project itself and documented in a report [6]. Within this project, stakeholder and technical requirements where derived. Three main tasks were identified for meeting the goals and requirements of this project, and where further divided through a Work Breakdown Structure (WBS).

- Map KTH campus
  - Book a vehicle that can be used for mapping
  - Perform software updates for mapping
  - Gather/buy components for mapping
  - Design and construct final design in CAD
  - Manufacture and assemble rig for sensors to be mounted on car for mapping
  - Map campus
  - Data modifications of map data
  - Deliver a ROS-bag of processed map
- Mounting for NVIDIA Drive PX2 on the RCV
  - Buy components for mount
  - Design and construct final design in CAD that protects the unit from disturbance
  - Mount and test Nvidia Drive PX2 on RCV

- Drive autonomously from point A to point B within KTH campus with the RCV
  - Keep the code within AD-EYE simulation as close as possible to the code that is running on the Drive PX2 at the end of the project
  - Mount and test all sensors on RCV
  - Software checks on RCV
  - Verify that all systems are running as planned
  - Test the RCV on KTH campus
  - Adaptation of software to integrate obtained 3D map

The planned methodology to be used through out the project was agile inspired, iteratively improving the project solution through sprints - implementing, testing, verifying and documenting. The reason of this was that the base solution for solving the task was low, according to the stakeholder guidelines, but the team wanted to develop as good of a solution as possible and improve it within the time frame of the project. However, the project hit more road bumps than planned, which resulted in the planned development steps taking more time than anticipated. The solution of this was that the methodology was changes to a more waterfall based strategy, since there was only time for one iteration of the full product.

Even though the overall methodology was waterfall implemented, iterations were done separately within different areas of the project. The mechanical design went through different stages of CAD models and prototypes, that were tested and evaluated, to finally land in the solution further described in Chapter 5.1. How the tests where conducted and evaluated can also be read about in Chapter 6.1. Regarding mapping, the map was updated by developing the quality of the PCD, mesh and vector map, documented in Chapter 5.2.

Furthermore, the AD-EYE implementation on the RCV was conducted in small steps, implementing one node and feature at a time, described in Chapter 5.4. The work followed a clear V-model structure [43], with outlining necessary operations and then defining a detailed design before implementation. Finally, integrating, testing and verifying the functionality of the RCV was done. Integration of the AD-EYE platform onto the RCV was given increasingly more focus towards the end of the project.

4.2 Time plan

The three main tasks and subtasks were planned for in a GANTT schdule, that can be viewed in Appendix A. An estimation was made to approximate the amount of time that was needed to finish each task. Parallel with finishing tasks, the

implementation and results were document in this report. Every week the GANTT schedule was revised to confirm of the project was preceding on time.

4.3 Management

During the first half of the project, the group was divided into subgroups in the following structure:

1. Mechanical design: Main task of finalizing construction of the sensor rig and testing its strength and robustness. Including responsibility for CAD design, prototype making, power supply for the sensors as well as testing and verification.
2. Sensors and map: Record ROS-bag, calibrate sensors, post processing of map, create point cloud map in Autoware, responsibility for mapping laptop and testing the result of the recorded data and map.
3. AD-EYE simulation: Responsibility and set up of software on Nvidia PX2 and stationary computer, run simulation with ROS, Matlab, Simulink, Autoware and Prescan.

Half way through the project, and once the mechanical construction of the sensor rig was done and tested, the subgroups were changed to the following:

1. Sensors and map: Create point cloud maps in Autoware, testing the result of the recorded data and map with different sensor adjustments.
2. RCV and AD-EYE integration: Run simulation with ROS, Matlab, Simulink, Autoware and Prescan and finally modify software to run with real world instead of simulation world. Communicate control signals to and from the RCV.
3. Post-processing: Filtering and creating meshes of point cloud, as well as constructing script for generating these features automatically.

The group used an online kanban board and had morning meetings once a week to update it. All assignments where given a responsible group and where no longer than approximately 15 hours of work. With these meetings, all group members were updated on the progress of the project and they were seen as a learning opportunity to spread knowledge between the subgroups. The members in the subgroups were rotated, giving everyone the chance to participate on several aspects of the project. Furthermore, weekly meetings with the project's supervisor were held.

Two group members, rotating though out the project, where given the project management responsibility. One responsible of kanban, schedule, task delegating and economy, and the other for email, communication, task delegating and meeting notes.

The project work documents have been collaboratively worked on through Google Drive, the documentation and code from the stakeholders have been given through Box and GitHub. Hardware has been manufactured at KTH Machine Design's Prototype Center and all purchased components have been ordered through KTH.

4.3.1 Third party software and libraries

Through out the project, various third party software and libraries have been used to solve different tasks and is presented in the lists bellow.

1. ROS: Robot Operating System, a flexible framework for writing robot software [44]
2. Autoware: A research and development platform for autonomous driving technology [16]
3. PCL: open source tools for 2D and 3D image and point cloud processing [45]
4. CloudCompare: a 3D point cloud and triangular mesh processing software [20]
5. SolidEdge: 3D modelling software [46]
6. Matlab: software tools for discrete data analysis and simulation [47]
7. Prescan: Prescan is a simulation tool for the development and validation of Advanced Driver Assistance Systems [38]

In section 5, the implementation the software mentioned above is further described.

Chapter 5

Implementation

The implementation phase was conducted based on the theory, planning and concept generation during the spring semester. All components used in this chapter are listed in Table 5.1.

ID	Component	Model	Qty.	Unit price	Total price	Notes
1	Lidar	Velodyne VLP-16	1	$4000	$4000	Borrowed from ITRL
2	Computer platform	Nvidia Px2 AutoChaffeur	1	-	-	Borrowed from stakeholder
3	Stationary computer	i9-9900K, RTX 2080, 32GB RAM	1	-	-	Borrowed from ITM
4	Lenovo Thinkpad	20QV001FMX, i7-9750H, GTX 1650, 32GB RAM	1	37000 SEK	37000 SEK	Borrowed from stakeholder
5	Suction cups	Mantona Action Camera Tripod w suction cups	3	179 SEK	537 SEK	Bought from Conrad
6	IMU	Xsens MTi-700	1	€3979	€3979	Borrowed from ITRL
7	M.2 SSD	WD Blue SN550 NVMe SSD 1TB M.2	1	1190 SEK	1190 SEK	Bought from Webhallen
8	M.2 SSD External Case	Asus ROG Strix Arion M.2 NVMe SSD Enclosure	1	649 SEK	649 SEK	Bought from Webhallen

9	Sensor rig cable clamp	Own design, PLA plastic.	1	-	-	3D-printed with material from KTH
10	Sensor rig base plate & Power case	Own design, 6 mm acrylic.	1	-	-	Laser cut with material from KTH
11	Sensor rig screws	5xM3 & 1x1/4-20 UNC	6	-	-	Obtained at ITRL
12	Batteries	12 V, 7.2 Ah	2	$\sim$ 250 SEK	500 SEK	Borrowed from KTH Prototype Lab
13	DC/DC converter	SD-50A-12, 50.4 W, 12 V	1	363 SEK	363 SEK	Bought from Elfa Distrelec
14	Molex minifit jr	4P 2x2	6	2,25 SEK	13,5 SEK	Bought from Elfa Distrelec
15	Molex minifit jr	20P 2x10	2	19,88 SEK	39,75 SEK	Bought from Elfa Distrelec
16	Zener diode	017AA 27V 5W	10	5,26 SEK	52,63 SEK	Bought from Elfa Distrelec
17	Fuse (10-pack)	10A fast FSF, 5x20mm	1	25.60 SEK	25.60 SEK	Bought from Elfa Distrelec
18	Fuse (10-pack)	16A fast SP, 5x20mm	1	37.86 SEK	37.86 SEK	Bought from Elfa Distrelec
19	Fuse (10-pack)	12,5A slow FS, 5x20mm	1	42.13 SEK	42.13 SEK	Bought from Elfa Distrelec
20	Fuse (10-pack)	16A slow FST, 5x20mm	1	28.65 SEK	28.65 SEK	Bought from Elfa Distrelec
21	Fuse holder	5x20mm, FPG3	2	25,48 SEK	50,95 SEK	Bought from Elfa Distrelec
22	Fuse holder	5x20mm, FX	2	51,28 SEK	102,55 SEK	Bought from Elfa Distrelec
23	Molex minifit jr	10P 2x5	2	5,40 SEK	10,80 SEK	Obtained at ITRL
24	Electrical wires	Multi-stranded	$\sim$2m	-	-	Obtained at KTH

Table 5.1. BOM list.

5.1 Sensor mounting

During implementation the solution for sensor mounting proposed in the planned concept and the SOTA report [6] was simplified and a single acrylic plate was used for all sensors. This plate was cut using a laser cutter, including featured holes for

mounting the various sensors as well as cable management. With the Lidar mounted on the top side of the plate, the IMU was placed directly under as seen in Figure 5.1. Since the vertical distance between them is known it can be accounted for when merging the retrieved positional data from the IMU with the points from the Lidar. Several iterations were constructed, featuring improvements to cable management and mounting with each new version.

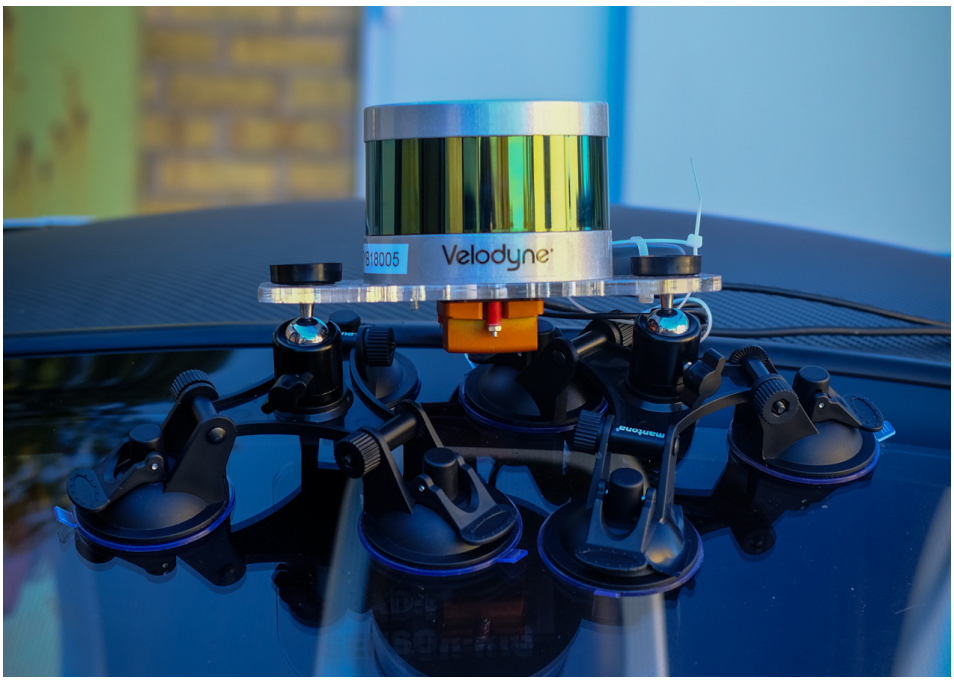

Figure 5.1. Close up of the sensor rig with Lidar on top and IMU underneath.

The plate is mounted on the top of a car using suction cups, see Figure 5.1. Two Mantona GoPro 3-leg suction cup holders were used [42]. With their ball joint on top, a firm placement with possibility of varying the horizontal angle as suited was provided. The feet of the tripods are able to tilt and ensure a snug connection even on convex surfaces often found on the top of a car or on the windshield.

Cables from the sensors run along the roof and in through a window where they connect to the power supply and a laptop for saving data, shown in Figure 5.2. In the event of unintended stretching of the cables, a cable strain relief system was mounted on the rear end of the plate, seen in Figure 5.3. This will minimize any harm done to cable connections on the sensors or to the sensors themselves.

Figure 5.2. The sensor rig mounted on the Renault Twizy, with cables drawn into the car through the driver's window.

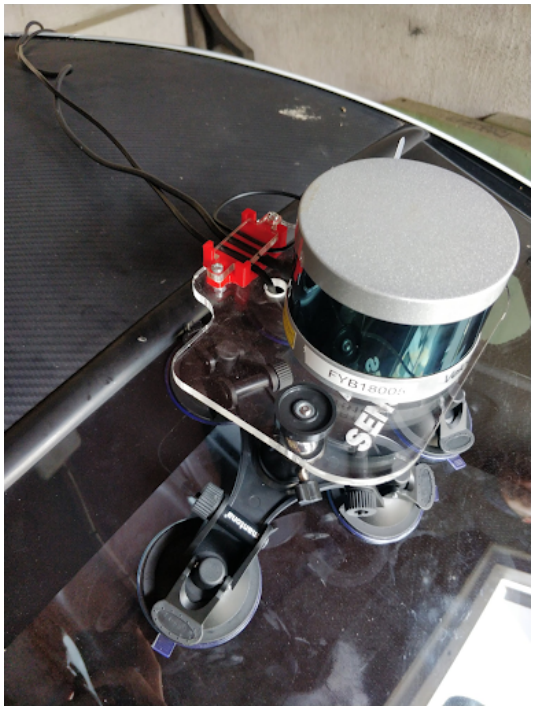

Figure 5.3. The sensor rig mounted on the Renault Twizy, with cable relief on the back of the platform.

5.1.1 Power case

In order to increase the ease of use, a case was made to house all the necessary components for powering the sensor rig, seen in Figure 5.4. A 12-volt battery was used in combination with a DC/DC-regulator of type SD-50A-12. The regulator was used in order to ensure a steady 12 volt output to the Lidar. The IMU and GPS were powered via USB by the laptop that recorded the ROS-bag. The case was made by laser cutting 6 mm thick acrylic and 3D-printing the parts that were needed for fitting the components. The case was made to fit the Lidar breakout board that could be easily inserted and removed from the case.

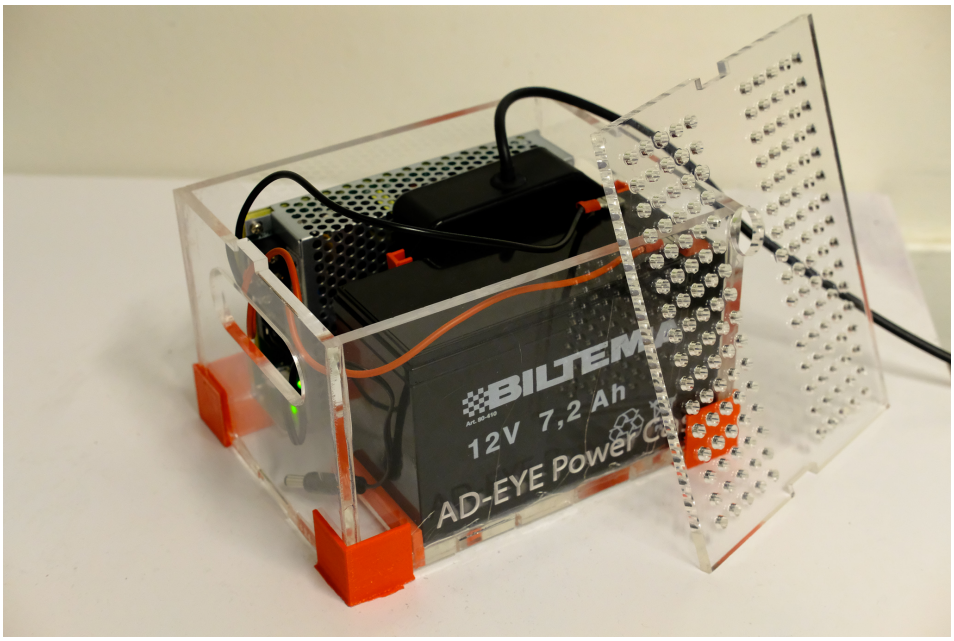

Figure 5.4. The case housing a 12-volt battery, a DC/DC-regulator and the Lidar breakout board.

5.2 Sensors and mapping

The IMU and Lidar were set up and started through ROS. Test recordings with the sensors were done to test the sensors functionality. A Python script was created to automatically start the sensors and the recording, to make the mapping faster and more user friendly. It includes options of what nodes want to be started with the script, for example the option to start Rviz and what sensors should be included in the recording.

5.2.1 Creating a map from ROS-bag

A Python script was created and implemented, using the techniques explained in section 2.3.2 and chosen concept in 3.2. The script also subscribes to the first sensor message from the IMU in order to compensate for orientational differences of the sensor rig when attaching it and sets a fix coordinate system from which the map is related to. It automatically saves the created map when it has finished computing, making the process of creating a point cloud from a ROS-bag with sensor recordings completely automated. Moreover, another script was developed, where it is possible to have a folder with multiple ROS-bags as input, where the script iterates through the folder, creating multiple point clouds.

The computed cloud can not be a sensor message larger than 1 GB due to internal ROS specifications limiting the message size over the network. Therefore, a large recording will not be able to be computed into a point cloud. In order to make a scalable solution, the final map was created by splitting the ROS-bag into smaller recordings, with overlapping sections. The submaps are later stitched together in CloudCompare, using the overlaps to get a transformation for the rest of the cloud. This solution would also provide a greater control of the mapping procedure and therefore resulting in a more stable solution that worked for all of our larger mappings. After stitching together the point clouds, a program written in C++ that utilized the pass through filter provided by PCL was used to cut the final point cloud into smaller chunks that could be loaded into the AD-EYE platform if needed.

An alternative method, approximate normal distribution transform, was tested and disregarded. The implementation in Autoware is based on the same procedure for generating a map, however the difference is that submaps are saved based on an approximate distance that the Lidar has traversed. The reason this can be computed is that the computation of normal distribution transform is only conducted on the active submap. The method was disregarded due to the error of overlaps between the submaps, as well as the built up errors in global precision. The method required a lot of testing and tuning of settings for every specific map, and could potentially not even be feasible for a given ROS-bag.

5.2.2 Comparing sensor settings

Tests were conducted to help with the development of the map and improving its quality. The tests and results of them are documented in section 6.

The speed of which the Lidar rotates will affect the resolution of the point cloud. Increasing the speed will lower the resolution meaning that there is a larger distance between every point in a frame. However, the benefit of increasing the speed is that there is a faster response of frame to frame variations. Typically it is better to have higher RPM when driving autonomously to have quicker response, and lower RPM

when mapping to get a better resolution[12].

Some of the tests for the RPM setting failed due to the normal distribution transform algorithm in Autoware. If a frame from the Lidar does not contain a 360 degree view of the environment, the algorithm will not be able to match it against the already created map.

To evaluate sensor performance, a factor contribution test was conducted. Using the map aligning process described in section 5.2.3, average distance deviation from ground truth from maps generated by the eight combinations of factors considered was recorded. This was done twice, by different people and covered two areas of KTH campus. The process could not be repeated more due to the time-consuming nature of the process of matching and recording data from eight different maps.

5.2.3 Comparing Point Clouds

For the comparison of the point clouds a software called CloudCompare was used. This software is developed to handle point clouds and meshes for both manipulation and evaluation. To be able to perform the point cloud comparison, there needs to be a reference cloud. For our purpose of evaluation this reference cloud needs to have a high precision as it will directly influence comparisons made with the map the team have created. For this task, the use of the 3D-Interactive map of KTH could be used as it has local millimeter precision. The main portion of the comparison was conducted after the evaluation of Lidar settings. However, as mentioned above, the comparison method was also used to compare the different settings.

The practical work in CloudCompare is pretty straight forward. First the user needs to make sure to select what part of the map to compare and cut out that part in the created point cloud. For the reference cloud, the user can do the same procedure but should make sure that the reference cloud is bigger than the created cloud. If the reference cloud is too large this could lead to performance issues, or crashes when operating the software. If this is the case, a higher amount of cutout or down sampling of the reference cloud be necessary.

When performing the actual comparison, the first step is to manually align the two clouds, this does not have to be done with super precision as the program has a self-alignment feature. The feature uses an iterative closest point algorithm to align the point clouds. As this alignment feature is made iterative (up to 999999 times), the end results are not constant after repeating the process. Therefore, several iterations of manual matching and alignment were made, but the differences are comparably small, in the millimeter range.

For calculating the Cloud to Cloud (C2C) distance, CloudCompare uses the two matched point clouds and computes the euclidean distance between the two closest points in both maps, for all the points in the manufactured map, and divides that

distance with the number of points. If the reference point cloud is dense enough, the approximated distance using the nearest neighbor method is considered acceptable. In other words, a dense point cloud results in the nearest neighbour distance being close to the true distance, both seen in Figure 5.5. In the case of the reference not being dense enough, local modelling should be used instead. Local modelling generates a better model for comparison, by locally approximating underlying surfaces for the nearest points in the reference. When the comparison is done the program displays the mean distance and the standard deviation of the two clouds.

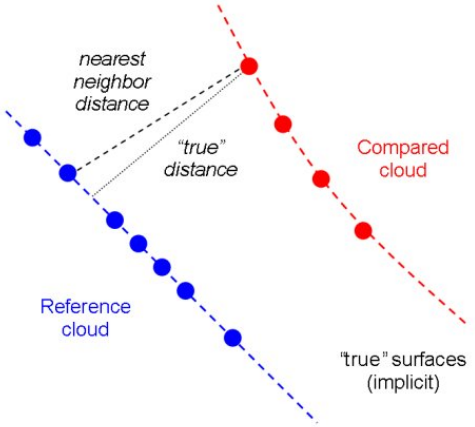

Figure 5.5. The nearest neighbour distance and the true distance between the compared cloud and the reference cloud [20].

CloudCompare features a plugin named Multiscale Model to Model Cloud Comparison method (M3C2). It is a method for point cloud comparison in 3D that is robust to changes in point density, missing data points and point cloud noise, which is not the case for the C2C method. The plugin tool also outputs an uncertainty estimation based on local surface roughness and test for the statistical significance of measured changes [48].

The method searches correspondences along surface normals and further uses local averaging around each point to reduce the influence of the surface roughness. The first step of the methods is to define the surface normal, N, by fitting a plane to the cloud points within a user-defined diameter, D, from each core point (left picture in Figure 5.6). Then a cylinder with diameter p is centered on the core point and projected along the same axis as N, both above and below the reference cloud. The maximum search distance, δ, is the cylinder length, which describes an area in which matching points in both point clouds are averaged and differentiated to calculate the M3C2 point cloud distance, PC_{dist} (right picture in Figure 5.6. See Figure 5.6 for a Simplified illustration of the M3C2 algorithm [49].

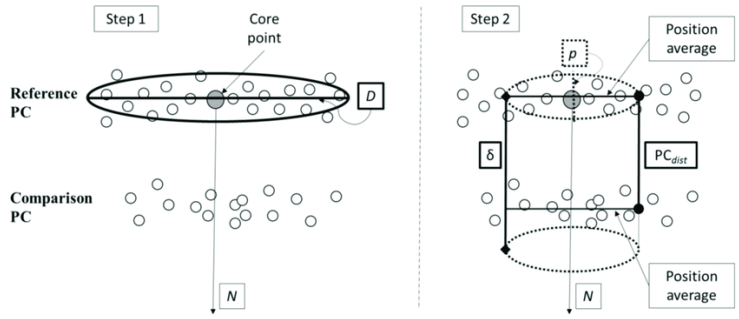

Figure 5.6. Simplified illustration of the M3C2 algorithm [49].

5.2.4 Meshing

The next step towards using the point cloud in a simulated world is to transform the points into a solid 3D structure using one of the surface reconstruction algorithms.

Comparing Reconstruction Algorithms

To assess the accuracy of the resulting meshes from the various reconstruction algorithms, it was relevant to examine how well the reconstruction was represented in comparison to the point cloud and how detailed it was. Accuracy in these assessments also took the amount of unwanted holes into consideration, where a high amount of holes was considered as low accuracy. Each mesh was visually compared to check how reliable and accurate each mesh was. This was done manually.

The comparison resulted in the use of the Greedy Triangulation algorithm, implemented using PCL in a C++ script. While the computation time exceeds some of the other methods the quality of the mesh triumphed among the tested algorithms. Utilizing PCL also allowed the implementation of voxel grid filtering in the same script, which removes clusters of points and improves the mesh result. A pass through filter was also implemented to divide the point cloud into several smaller clouds before meshing them individual to reduce the size of the models generated. Using the maximum and minimum values for the x and y coordinates of the points in the point cloud as starting points, the script generates a grid of desired size and meshes each square separately. Obtaining a smaller file size for the 3D models was crucial when importing into Prescan, as it rejected too large files. Using PCL, the output could either a .obj file or a .stl file which was opened with the 3D modelling software Blender, where the model was simply imported then exported as a .dae file which is preferred by Prescan.

5.2.5 Vector maps

The final part of the mapping process is generating vector maps that correspond to the point cloud. This was done in two ways - depending on the use case. For

use in the Prescan environment and setting up experiments, the built in assets for generating vector maps was used. By placing road segments inside the program, corresponding two dimensional vector maps could be generated using a previously generated script that utilized positional data from the road segments in the simulation world.

When generating vector maps for real world driving, the Autoware tool VectorMap-Builder was used. By placing linked nodes roughly every 1-2 m driving lanes could be created while viewing the point cloud in a 3D environment. This allowed nodes to differ in their z coordinate, and each individual node could be adjusted to fit the ground level of the point cloud. Additional information such as stop signs, curb placement, lane width, traffic signs could be added if needed in the same procedure as the driving lane nodes. With the relevant information added the vector map could be downloaded and transferred to the Nvidia Drive PX2 for use in driving both in the real world but also in simulation.

5.3 AD-EYE simulation

The AD-EYE platform has mainly been built with the help of testing through scenarios in Prescan. These scenarios have been manually created to evaluate several different functions in the AD-EYE platform, such as the trajectory planner and the safety channel. For the purposes of this project, Prescan has been used to test the functionality of the hardware to be used in the final solution. To begin with, a simulation computer was set up that ran Prescan and the basic scenarios developed for other experiments with AD-EYE. When these ran successfully on the PX2 computer, specific experiments were developed in Prescan to test the AD-EYE's ability to navigate with real-world data.

5.3.1 Primary hardware simulations

The first simulations that were run on the hardware to be used were simple trajectory planning verification simulations. These simulations were confirmed to be working on other platforms, thus were used as a base for testing the hardware for the final product. The simulations showed that some minor modifications needed to be done in the implementation code on the PX2 computer for the hardware to function properly. These modifications were mainly start time delays of certain functions in the Autoware platform, but also some hardware configurations to enable the full capability of the PX2 computer processors.

When everything functioned properly, the AD-EYE platform ran on the PX2 computer and connected through ethernet to another stationary computer running Prescan, MATLAB and Simulink. In Prescan a car was simulated and the PX2 computer acts as a controller for linear velocity and angular velocity. The messages from the Prescan simulation consisted of sensor data, goal position, current position (GNSS

fake localization position) and current state control. The sensor data consisted of odometry data, point cloud data from the simulated Lidar and real time camera images for detecting road signs. The current state control determines what functions the AD-EYE platform is running, such as if it is currently running trajectory planning, safety channel or motion planning. The messages from the PX2 computer to the Prescan computer consisted of Autoware's Twist messages that contains linear and angular velocity.

At this stage, all communication between the two computers were done using ROS and the ROS functionality in MATLAB and Simulink. This set up allowed for very simple integration of ROS over two different computers that are connected to the same network as network connections are set up by ROS.

5.3.2 AD-EYE real-world simulation

For finding how well the AD-EYE platform would behave in a real world environment, an experiment within Prescan was created that would create an environment as close to the real world as possible. A mesh, generated from the point cloud of KTH Campus, was used in Prescan to simulate the real world. The point cloud was loaded in the PX2 computer together with a vector map created in Prescan with vector map extraction scripts from the AD-EYE repository.

The simulations showed general issues with this setup as the PX2 platform had issues of localization using NDT-matching in this context. The issues were thought to originate from the Prescan setup and the mesh, as these were the most significant changes to the setup when the issues arose.

5.4 AD-EYE on the RCV

The integration of the AD-EYE platform and the RCV was conducted at ITRL. To integrate the AD-EYE platform with the RCV, the AD-EYE platform was installed on a Nvidia Drive PX2 computer. The PX2 was connected to an already mounted ethernet switch on the RCV, that in turn was connected to a MicroAutoBox on the RCV. A Velodyne VLP16 Lidar was also connected directly to the ethernet switch.

The Simulink file for the RCV was flashed onto the RCV using dSPACE Control Desk. A GUI was created inside dSPACE that enabled testing of various settings on parameters for control without needing to re-flash the RCV, as well as monitoring signals from the Simulink model, for example the vehicle's velocity.

The Nvidia Drive PX2 was mounted in the box under the driver's seat on the RCV, described in section 3.3 and Figure 3.5. This was done by screwing holes in the bottom metal plate of the box and attaching the PX2 with screws. Since it did not become warm while running the AD-EYE platform and the box had one open side,

the decision to not implement any fans in the box was made. Since the RCV could not drive during bad weather, water protection was discarded. The mounting could support the driving forces from the RCV as well as be mounted without disturbing other components.

5.4.1 Power adapter

A power adapter was created in order to power the Nvidia Drive PX2 on the RCV, see Figure 5.7. The power connector used a Molex Minifit 2x5 housing with additional cables that could fit the power connector on the PX2. The circuit used for this power adapter followed the manufacture's documentations [50] with some modifications such as a larger fuse of 20 Amps and a Zener diode of model 1N5361BG [51].

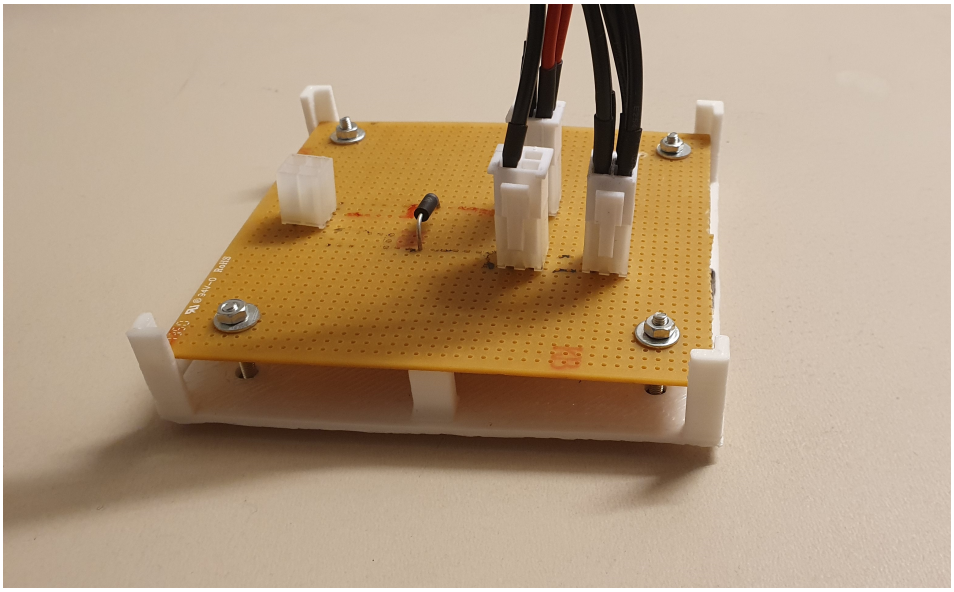

Figure 5.7. The power adapter for powering the Nvidia Drive PX2.

The power adapter input was connected to a fuse box on the RCV, that in turn was connected to a DC/DC converter on the RCV that could supply 12V, and the power adapters output was connected to the PX2.

5.4.2 UDP communication

Since the MicroAutoBox on the RCV is not ROS-compatible, Simulink ROS-blocks could not be used for the communication between AD-EYE and the RCV. This factor, together with the aspect that UDP communication was already implemented on the RCV-, resulted in that UDP became the natural choice for communication.

The UDP connection was set up between the PX2 and the RCV's MicroAutoBox. From the PX2 to the RCV, control signals that AD-EYE transmitted were encoded and sent as a 12 byte vector containing the UDP-package number, angular velocity and linear velocity. These were then decoded in Simulink and used in the controlling of the RCV, seen in Appendix B Figure B.1. From the RCV to the PX2, a 52 byte vector was encoded and sent with data containing velocity, IMU-values and more sensor data. The AD-EYE platform need these values as feedback, as decoded UDP-messages. These signals are shown in Appendix B Figure B.2.

5.4.3 ROS integration

The AD-EYE platform was dependent on a number of ROS-topics that needed to be created based on the values received over UDP from the RCV. A ROS-node was created for handling the UDP unpacking from the RCV. This ROS-node was created in Python and used the socket library in python [52] to unpack the UDP message from the RCV and published this message with a custom made ROS-message type on the topic `rcv_get`. Another ROS-node was created in similar fashion but was responsible for sending information from the AD-EYE platform to the RCV. This topic was named `rcv_send`. See Figure 5.8 for a graphical representation of the node communication between the RCV and AD-EYE.

After this, each topic that AD-EYE needed for operation, that was directly dependent on the RCV, was created as a separate ROS-node, where each node subscribed on the published custom message from the topic `rcv_get`. The topics were then published with the correct naming convention in order to match AD-EYE without modification on the AD-EYE code.

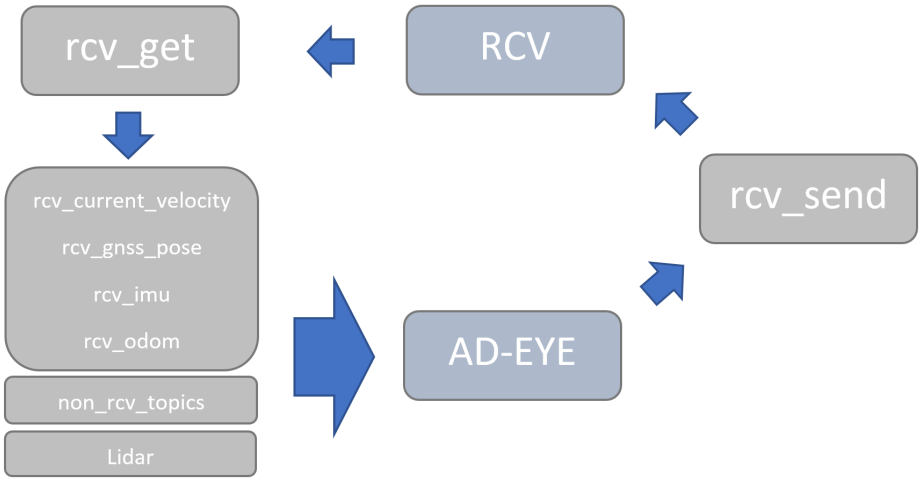

Figure 5.8. Overview of the communication between the RCV and the AD-EYE platform

A launch file was created that launched all the ROS-nodes. For integrating the Velodyne VLP16, whose information does not go through the RCV's MicroAutoBox, the official ROS-package for this Lidar [53] was also launched inside this launched file, but remapped to fit the naming convention of the AD-EYE platform.

5.4.4 Velocity control

A PI feedback controller was created inside the RCV's Simulink file, Appendix B Figure B.1, to control the throttle and braking of the RCV. The reference input velocity for the controller came from the AD-EYE platform's computed target velocity that was sent over UDP to the RCV. An "if/else" statement was put in front the controller to determine which system should be engaged, the braking or throttle. If the targeted reference velocity was negative, the PI controller controlled the braking and if the targeted velocity was positive the PI controller controlled the throttle.

In order to tune the controller, the RCV was lifted up on a stand and a targeted reference velocity from 0 m/s to 5 m/s was sent over UDP to the RCV. This step response was viewed in dSpace Control Desk, where the step input could be plotted together with the actual velocity of the RCV. During these circumstances, with no friction from the ground and a step input of up to 5 m/s, the controller sufficiently followed the velocity input, with a close to immediate response and without a noticeable overshoot.

5.4.5 Steering control

AD-EYE outputs its control signals as linear velocity and angular velocity, which had to be recalculated into a steering angle, required for the RCV. The calculation was done in the Simulink file, see Appendix B Figure B.1, according to Equation 5.1 and Figure 5.9. The variable φ is the car's front wheels' angle, ω is the angular velocity, v is the linear velocity, R is the turning radius of the vehicle and the distance between the wheel axes is 2.6 meters. The signal that controls the the RCV corresponds to the steering wheel angle, between $-\pi$ and π, whereas the steering angle φ of the RCV is multiplied with the steering ratio of 7.3 in Equation 5.2.

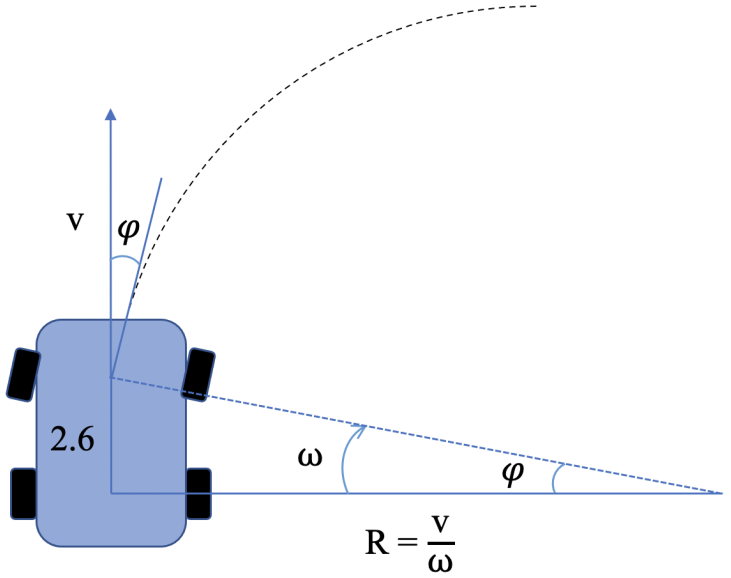

Figure 5.9. The steering angle φ of the RCV is calculated based on the RCV's angular velocity ω and linear velocity v.

$$\varphi = \arctan\left(\frac{2.6}{\frac{v}{\omega}}\right) \tag{5.1}$$

$$\text{steering wheel angle } = \varphi \cdot 7.3 \tag{5.2}$$

5.4.6 Odometry

The odometry of the RCV is used for localisation in the AD-EYE platform and is defined as the position of the vehicle with relation to the position that it started from. The odometry values in the x and y direction are calculated on the Simulink side according to Appendix B and Figure B.3. The equations for these positions are furthermore defined in Equation 5.3 and 5.4, where ω is the RCV's angular velocity and v is its linear velocity as in Figure 5.9.

$$x = \cos\left(\int \omega dt\right) \int v dt \tag{5.3}$$

$$y = \sin\left(\int \omega dt\right) \int v dt \tag{5.4}$$

Chapter 6

Verification and validation

6.1 Sensor rig

Tests of the sensor rig were conducted to verify the strength of the constructed mount and purchased suction cups so that it would withstand the stress from the mapping sessions. The main concerns were if the suction force provided was sufficient, and for how long adhesion could be maintained in different scenarios. Therefore tests were conducted on the rig with a mass that exceeds the combined mass of all sensors by a factor of at least two. The tests can be seen in Table 6.1.

TEST CASES	Pass [Y/N]	Force [N]	Material	Time [min]	Sliding [Y/N]	Fail Type	Comments
Horizontal force:							
H1	Y	47	Painted metal	300	N	N/A	Rig fastened on whiteboard, All cups firmly in place
Vertical Force:							
V1	Y	47	Vinyl (plastic)	35	N/A	N/A	(35 min: 2 cups loose)
V1.1	N	47	Vinyl (plastic)	110	N/A	Cups	(110 min: 3 cups loose)
V1.2	N	47	Vinyl (plastic)	150	N/A	Cups	Test Cancelled at 1.5hrs
V2	Y	47	Painted metal	150	N/A	N/A	Test Cancelled at 1.5hrs, All cups firmly in place
Mounted to car							
S1	Y	30	Windsheild	45	N		13 degrees celsius,86% humidity: (50 min: 1 cup loose)
S1.1	Y	30	Windsheild	75	N		
Driving							
D1	Y	30	Windsheild	10	N		13 degrees celsius, 86% humidity: 1 cup loose)
D1.1	Y	30	Windsheild	15	N		13 degrees celsius, 86% humidity: 1 cup loose) 1 cup loose before contuning driving

Table 6.1. Test cases and results for tests conducted on the suction cups and sensor rig.

6.2 Map generation

When evaluating combination of parameters that produce the most accurate map with respect to ground truth, a factor contribution test was used. The three factors that were considered are sensor settings, sensor tilt and map filtering. These factors

were binary, meaning they only had two states that were tested. This means there are 8 possible combinations that could be tested. By producing maps using all three combinations of factors, a test could be made where one factor is varied and two others are held constant to see what the difference in map accuracy is. The results of this process is presented in the results section.

In this project there are two ways of looking at precision, namely local and global precision. Local precision is the amount of accuracy at objects in the map, for example corners or road networks. The global accuracy is a measurement of how the position of these objects correspond to the real world. Local and global accuracies differ in the way they affect certain aspects of the mapping and practical use. The local precision is more important for the NDT-matching as it directly influences the ability to match objects. However, the global precision is more important when looking at implementation for driving autonomously in combination with the use of GPS.

When evaluating the map several sections were tested. The first part of the evaluation was conducted by cutting out two small parts of the created map at different locations. The focus was static objects, unlikely to change between the created map and the reference map. This was done primarily to evaluate the local precision of the map.

The second step was to evaluate a larger section of the map. For this, the whole area around the KTH M-building was chosen as a proper testing ground as there are several connecting buildings. This is also the area where the final automated driving test run would be conducted. For this test we can evaluate how the mapping performed on a bigger scale. The last test was to see how the whole map aligned against 3D-Interactive's map, however this may not be as important as the local precision as we can not validate 3D-Interactive's map for a section as big as KTH.

Furthermore, it was interesting to study which confidence interval should be attached to each distance measurement and test if a statistically significant change has occurred in the comparison. There are many causes for uncertainty in comparing point cloud data but for high precision Lidars, the total error on change detection is dominated by the point clouds registration error and the surface roughness. Therefore, the confidence interval depends on these sources of uncertainty. These tests were performed using the Multiscale Model to Model Cloud Comparison method (M3C2) in CloudCompare.

After manually aligning the clouds, the registration error is decreased using the Iterative Closest Point (ICP) tool in CloudCompare. ICP is an iterative process in which the registration error gradually decrease. This process is stopped either after a maximum number of iterations, or as soon as the error (RMS) difference between two iterations becomes lower than a given threshold. The three tests for

each location (M-building corner and U-building corner) are shown in Table 6.2 with their respective Cloud to Cloud (C2C) mean distance and standard deviation.

A change significance test was performed for each of the comparisons in order to validate the precision tests. The results of the local precision test can be seen in Table 6.3, Figure 6.3 and Figure 6.4. Figure 6.1 and 6.2 displays the uncertainties related to the distance measurements for a 95% confidence interval for each M3C2 distance comparison and shows that the distance uncertainties are close to zero.

TEST CASES	C2C Mean Distance [m]	C2C Standard Deviation[m]
M corner test1	0,029378	0,042363
M corner test2	0,02939	0,042383
M corner test3	0,030201	0,042178
U corner test1	0,050547	0,091007
U corner test2	0,053833	0,089361
U corner test3	0,049793	0,091447

Table 6.2. Iterations for alignment and ICP local precision using the final Lidar settings.

TEST CASES	C2C Mean Distance [m]	C2C Standard Deviation[m]	M3C2 Mean Distance [m]	M3C2 Standard Deviation[m]
M corner	0.029656	0.042308	0,005054	0,0019555
U corner	0.051391	0.090605	0,001807	0,017416

Table 6.3. Mean value for test cases for local precision using the final Lidar settings.

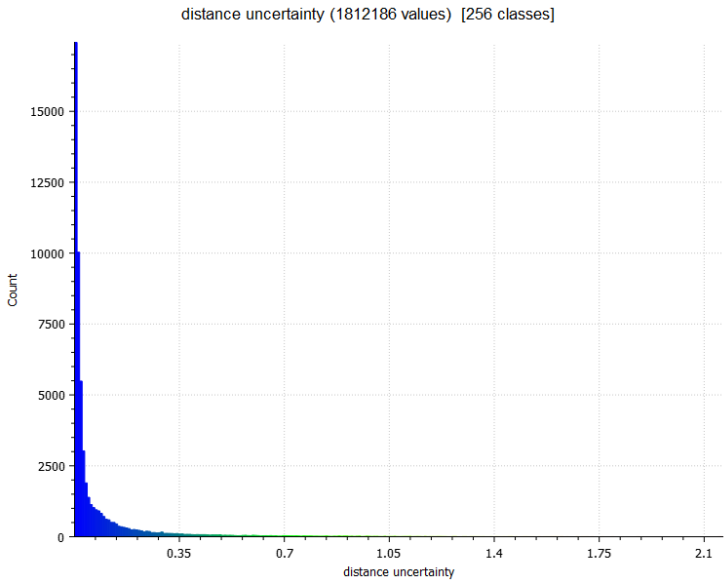

Figure 6.1. Histogram showing the distance uncertainty for local precision [m] at M-building corner.

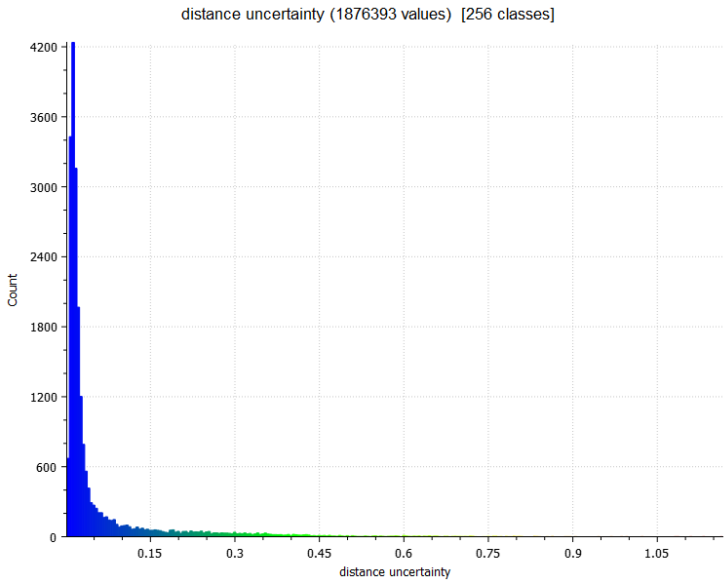

Figure 6.2. Histogram showing the distance uncertainty for local precision [m] at U-building corner.

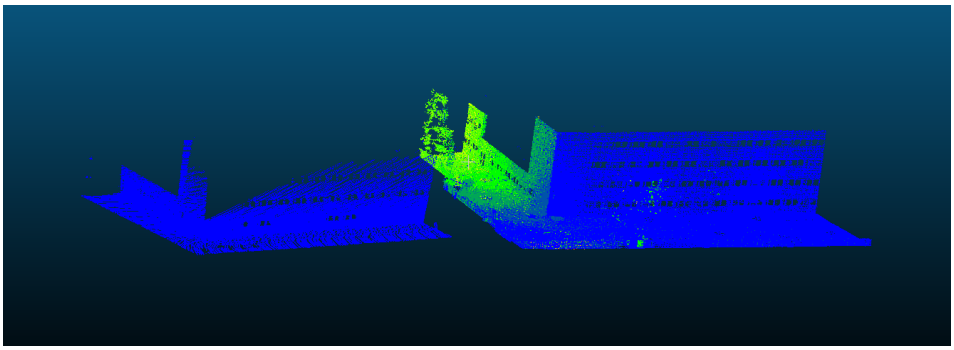

Figure 6.3. Side by side view of created map and 3D-Interactive's map at the M-building before alignment.

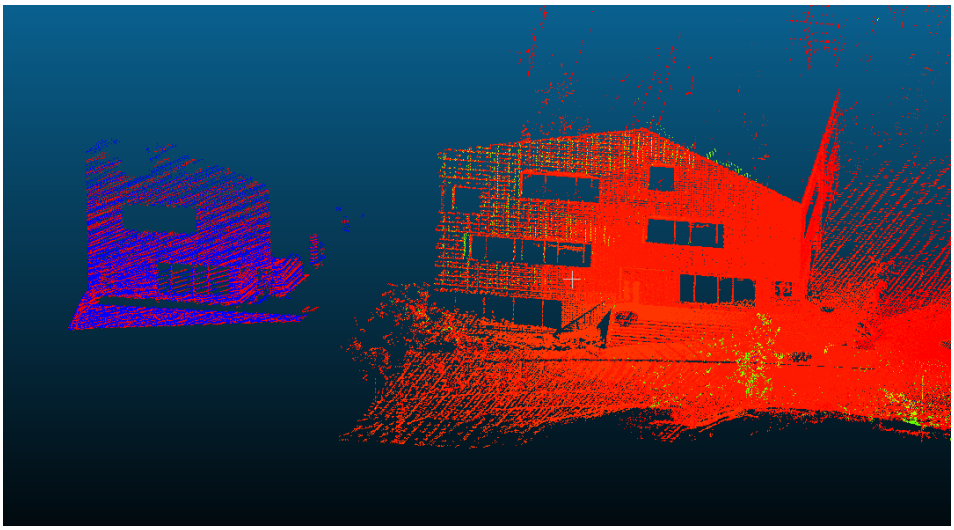

Figure 6.4. Side by side view of created map and 3D-Interactive's map at the U-building before alignment.

For the testing of the bigger maps the results can be seen in Table 6.4, Figure 6.5 and Figure 6.6. Figure 6.7 shows the uncertainties for the distances and the significance test can be displayed in Figure 6.8. In Figure 6.5 the blue points are a part of the map created by the team, and the other colors, representing different intensities are points from 3D-interactive map. The two different maps can be seen side by side in Figure 6.6. As can be seen in table 6.4, the C2C mean distance is reduced when cars are removed on the created map, this implies that the resulted mean distance is probably smaller than what is calculated using the C2C computation tool.

TEST CASES	C2C Mean Distance [m]	C2C Standard deviation[m]	Comment
M	0,277243	0,347574	
M, reduced 3D interactive noise test2	0,282171	0,356196	
M removed cars on created map	0,25161	0,311258	using standard 3D-int map
created map test1	0,660004	0,574315	
created map test2	0,658373	0,581591	Realignment

Table 6.4. Test cases for larger area precision using the final Lidar settings.

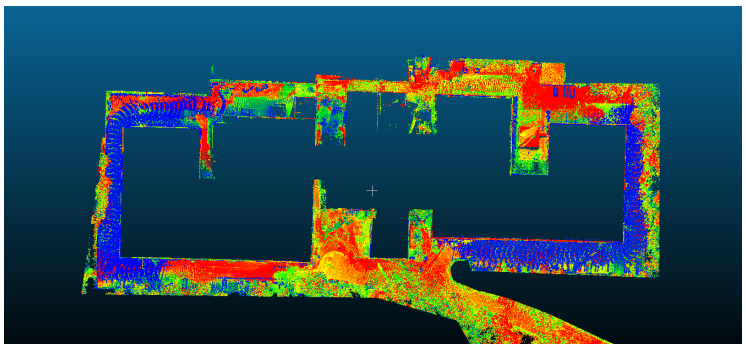

Figure 6.5. Overlap of created map and 3D-Interactives map at the M-building.

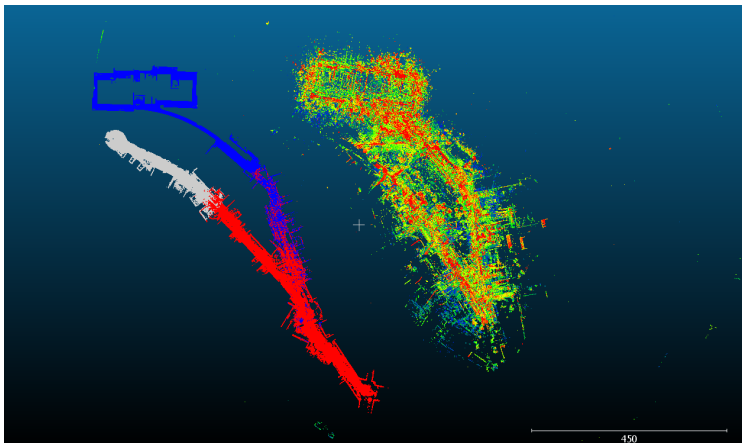

Figure 6.6. Side by side comparison of full size created map and 3D-Interactive's map.

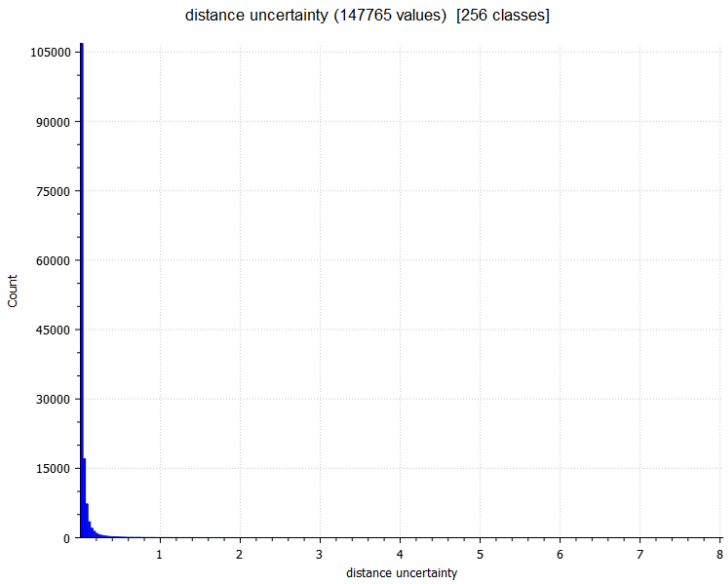

Figure 6.7. Histogram showing the distance uncertainty for local precision [m] at larger area, in this example M.

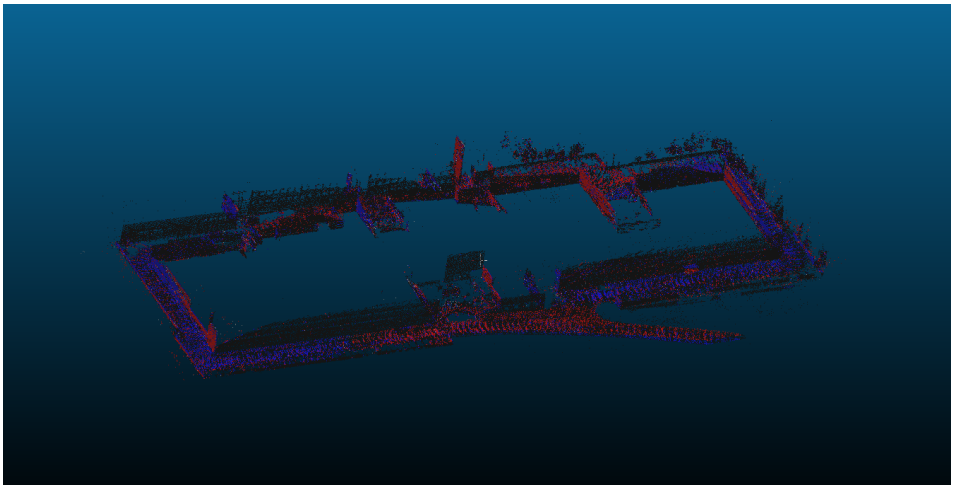

Figure 6.8. Significance test for M building, the colored points correspond to significant points.

A high speed test was conducted to see how the higher speed would affect the local and global precision. The speed during the mapping was around 25 km/h. Mapping at this speed resulted in huge artifacts. Results can be seen in Figure 6.9.

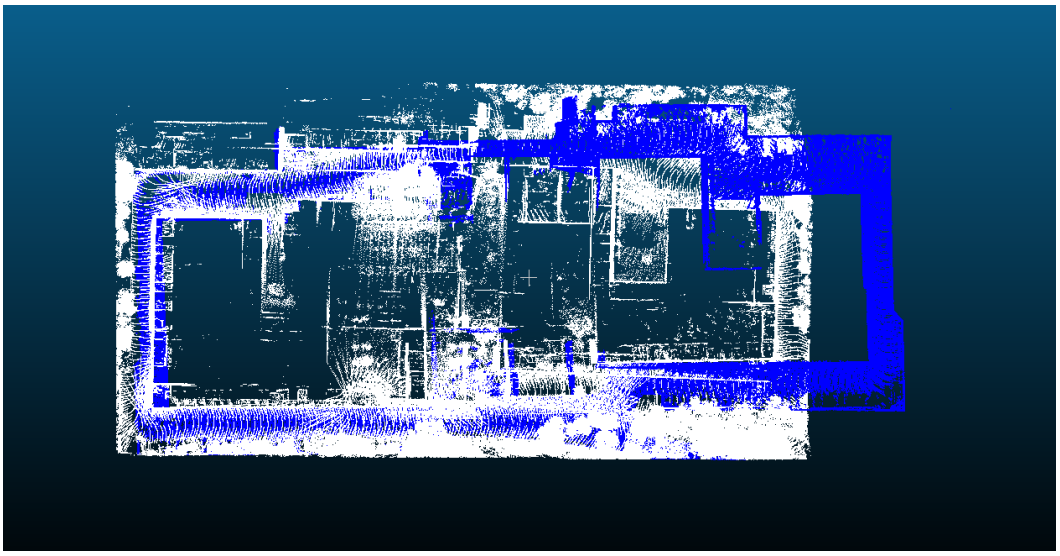

Figure 6.9. Side to side comparison of our map (blue) and the high speed map (white).

A comparison of the two maps in Figure 6.9 shows that the NDT algorithm fails to match objects properly giving a false shape of the building at M, when conducting high speed mapping.

6.3 RCV

To verify that each node worked properly, each node was tested separately by checking that the ROS-node published the values that corresponded to the right value in the dSPACE GUI.

In order to test the controller of the RCV's throttle and steering, the RCV was lifted up on a stand indoors at ITRL and a Hardware-In-The-Loop test was conducted. The AD-EYE platform was connected to the RCV as well as to a separate Windows 10 computer running a Prescan simulation experiment. As the simulated experiment started, a visual verification that the RCV turned and followed the targeted velocity was done by looking at the values plotted in dSPACE.

To ensure that the NDT-matching worked properly, a point cloud map of the indoor environment at ITRL, where the RCV was located, was created. The map was loaded into AD-EYE and a test of the localization was conducted with the Velodyne VLP16 from the RCV. A visual confirmation of the localisation was done in Rviz, where it could be seen that the map and the surroundings were matched correctly. The RCV was lastly taken outside and tested with the final map.

Chapter 7

Results

The stakeholder and technical requirements have been evaluated, as pass or fail, to summarize how well the project goal has been fulfilled, in the evaluation matrix in Table 7.1. A total of 35 out of 41 technical requirements have been fulfilled for the final solution.

Nr.	**Requirement**	**Fulfillment**
1.1	The point cloud map shall be recorded into a ROS-bag.	Pass
1.2	The point cloud map shall be exported as .pcd format.	Pass
1.3	The point cloud shall be generated by utilizing Lidar and IMU.	Pass
1.4	The ROS-bag should include GPS-data.	Pass
1.5	The sensor data should include camera pictures, to localize traffic signs etc.	Fail
1.6	A method for filtering away noise and disturbances in the recorded ROS-bag should be implemented.	Pass
1.7	The point cloud should have sufficient accuracy in order for the RCV to utilize ndt-matching.	Pass
1.8	The Lidar should be calibrated with a rotational speed, angle and return type that provides the desired precision.	Pass
1.9	The driving speed should be optimized to provide the desired precision.	Pass
1.10	The point cloud shall be converted into a mesh sufficient enough for the simulation to run.	Fail
1.11	The mesh shall be exported as a .dae-file.	Pass
1	Create a detailed point cloud map of the KTH campus.	9 / 11
2.1	All subsystems of the AD-EYE platform should be tested in simulation before driving autonomously in real life.	Pass
2.2	Performing final self driving tests shall be done at KTH in conjunction with the safety plan set at ITRL.	Pass

2.3	The RCV shall provide the AD-EYE platform with its GPS position.	Pass
2.4	The RCV shall provide the AD-EYE platform with its current angular velocity.	Pass
2.5	The RCV shall provide the AD-EYE platform with its current linear velocity.	Pass
2.6	The RCV shall provide the AD-EYE platform with IMU data.	Pass
2.7	The AD-EYE platform shall provide the RCV with a reference angular velocity.	Pass
2.8	The AD-EYE platform shall provide the RCV with a reference linear velocity.	Pass
2.9	The communication between the RCV and the AD-EYE platform should follow the UDP protocol.	Pass
2.10	The computer that runs the AD-EYE platform shall be powered by the RCV.	Pass
2.11	The RCV shall provide the AD-EYE platform with data from at least one Lidar.	Pass
2.12	The RCV should provide the AD-EYE platform with camera imaging for its detection.	Fail
2.13	The RCV shall provide the AD-EYE platform with its odometry position.	Pass
2.14	The RCV shall handle linear velocity and angular velocity as control inputs.	Pass
2.15	AD-EYE should be able to use Lidar data for localisation.	Pass
2.16	AD-EYE should be able to use GPS data for localisation.	Fail
2.17	AD-EYE should be able to use odometry data for localisation.	Pass
2	Autonomous driving from point A to B on KTH campus.	15 / 17
3.1	The sensor mount shall fit most types of car roofs or windshields.	Pass
3.2	The sensor mount shall not cause any damage to the car.	Pass
3.3	The sensor mount shall protect the sensors from any external forces and vibrations that might harm them. Handling a 30 N load in all directions while driving for at least 30 minutes is required.	Pass
3.4	The Lidar shall be placed so that it can operate without hindering the field of view.	Pass
3.5	The sensor rig, including sensors, should be water resistant.	Pass
3.6	The water sensitive components shall be placed within the vehicle.	Pass

3	Design a custom cluster of sensors needed to record maps of KTH, mounted on a vehicle.	6 / 6
4.1	The mount shall protect the Nvidia Drive PX2 from any external driving forces that might harm the card.	Pass
4.2	The mount shall protect the Nvidia Drive PX2 from any water damage in case of precipitation.	Fail
4.3	The mount shall make sure that the Nvidia Drive PX2 is not overheated during operation, maximum 60 °C.	Pass
4.4	The Nvidia Drive PX2 shall be placed so that it does not interfere with existing components.	Pass
4	Design a custom mound for the Nvidia Drive PX2 on the RCV.	3 / 4
5.1	The vector map shall be exported as .csv format.	Pass
5.2	The vector map shall be compatible to run on the AD-EYE platform and give global path planner the ability to create a trajectory from point A to point B.	Pass
5	Create a vectorized map of the KTH campus.	2 / 2
6.1	The budget shall not exceed 50 000 SEK, hence as many components as possible should be components that are already available for use at ITRL or MECS.	Pass
6	Keep the project within the budget.	1 / 1

Table 7.1. Evaluation matrix of stakeholder and technical requirements.

7.1 Sensor mounting

The conclusion from the tests on the sensor rig was that when mounted properly on suitable materials, the suction force had minimal to no issues maintaining a firm grip for a well exceeded load over a time longer than typical usage. In the cases where one or more suction cups lost its grip the remaining cups had no problem of maintaining the load, even when extra force was applied by pulling on the mounts by hand. The results were deemed sufficient to pass the the rig and use it for the sensors. When implemented in the real world mapping, the results were the same.

7.2 Mapping

To evaluate how sensor settings affect the resulting point cloud map compared to ground truth, a factor contribution analysis was conducted. Three factors were taken into account - sensor settings, sensor tilt and map filtering.

The two sensor settings tested had the ”dual” return type at 1200 RPM (1200 D) speeds versus the ”strongest” return type at 600 RPM (600 S). This is based on initial testing, where it showed that the RPM had a lower limit of 1200 for the

dual setting and 600 for the strongest setting. This is due to the implementation of the normal distribution transform algorithm stated in section 5.2.2. Furthermore, sensor tilt was either angled around 30 degrees towards the ground (denoted ”V” in the tests) or ”straight”, meaning it was parallel to the ground (denoted ”R” in the tests). An ”F” is added to the test sample if the map was filtered, otherwise !F is used to indicate that the map is unfiltered.

Due to their binary nature, the 3 factors have 8 possible combinations. Each combination was compared to determine which factor contributes the most to minimizing the average distance of the points in the map from the closest ground truth point. The results of each of these factor combinations can be seen in Figures 7.1 and 7.2.

The factor contribution test was conducted by varying one factor at a time, such as the map being filtered or not, and seeing the difference it makes to the average distance of points in the map generated with these settings compared to ground truth. The results and the average are shown in Figures 7.3 and 7.4 and Tables 7.2 and 7.3.

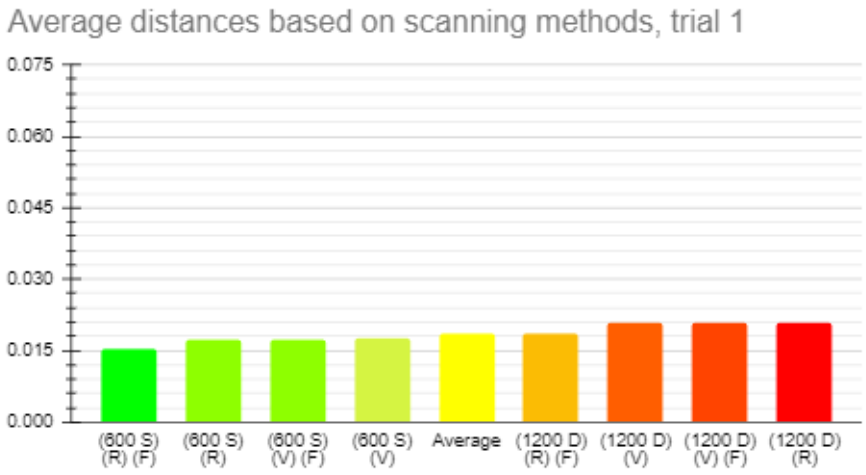

Figure 7.1. The results of the first sensor comparison trial where map pieces of an M-building corner were compared. The y axis shows the average distance deviation from ground truth in meters.

Since 2 binary factors have 4 combinations, each variation happens 4 times. Since this is done with 3 factors - 12 different values are obtained. This is illustrated in Tables 7.2 and 7.3, where x is the variable and the other two factors are constants.

As seen from the factor contribution figures, using the correct sensor settings had a major impact and accounts for roughly 60% to 80% of the total factor contribution. Filtering the map also has a small positive impact on map quality. Sensor tilt seems to have less of an impact and might require further testing to confirm or reject it

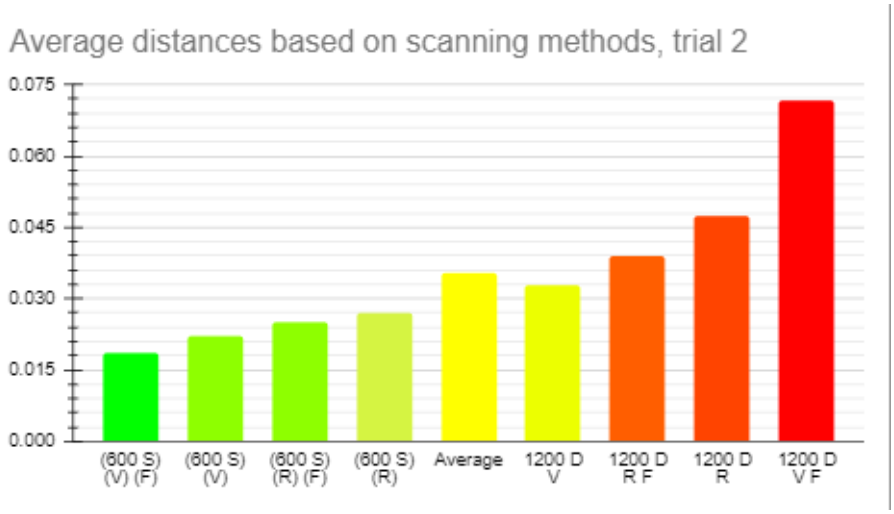

Figure 7.2. The results of the second sensor comparison trial where map pieces around the B-building entrance were compared. The y axis shows the average distance deviation from ground truth in meters.

Operation	Sensor settings	Sensor tilt	Filtering	Difference [m]
(!F-F)	600 S	V	x	1.75e-3
(!F-F)	600 S	R	x	0.34e-3
(!F-F)	1200 D	V	x	2.30e-3
(!F-F)	1200 D	R	x	-0.05e-3
(!F-F)	Average			1.09e-3
(V-R)	600 S	x	!F	0.43e-3
(V-R)	600 S	x	F	1.84e-3
(V-R)	1200 D	x	!F	-0.22e-3
(V-R)	1200 D	x	F	2.13e-3
(V-R)	Average			1.05e-3
(1200 D - 600 S)	x	V	!F	3.78e-3
(1200 D - 600 S)	x	V	F	3.23e-3
(1200 D - 600 S)	x	R	!F	3.12e-3
(1200 D - 600 S)	x	R	F	3.52e-3
(1200 D - 600 S)	Average			3.41e-3

Table 7.2. The factor contribution combinations from the first trial visualised. !F meaning unfiltered, V meaning straight sensors, R meaning tilted sensors, 1200 and 600 numbers are the Lidar RPM and D and S are the ”dual” and ”strongest” return types, respectively. A positive difference here means the second factor in the operation equation (1st column) is the better one, since it has a greater impact on minimizing the average distance in the resulting point cloud map.

as a meaningful factor.

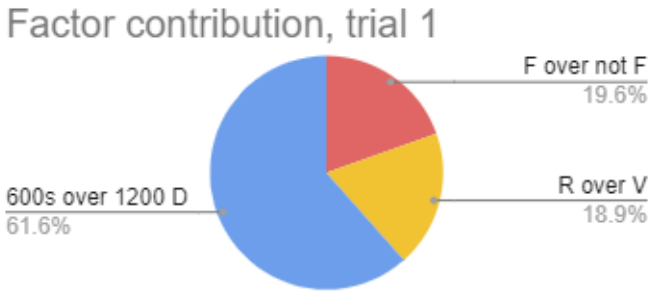

Figure 7.3. The factor contribution averages from trial 1 visualised as percentages.

Operation	Sensor settings	Sensor tilt	Filtering	Difference [m]
(!F-F)	600 S	V	x	3.70e-3
(!F-F)	600 S	R	x	1.77e-3
(!F-F)	1200 D	V	x	-6.04e-3
(!F-F)	1200 D	R	x	24.46e-3
(!F-F)	Average			5.97e-3
(V-R)	600 S	x	!F	-0.47e-3
(V-R)	600 S	x	F	-6.61e-3
(V-R)	1200 D	x	!F	-14.42e-3
(V-R)	1200 D	x	F	32.85e-3
(V-R)	Average			1.78e-3
(1200 D - 600 S)	x	V	!F	20.41e-3
(1200 D - 600 S)	x	V	F	13.80e-3
(1200 D - 600 S)	x	R	!F	10.68e-3
(1200 D - 600 S)	x	R	F	53.26e-3
(1200 D - 600 S)	Average			24.54e-3

Table 7.3. The factor contribution combinations from the second trial visualised. Factor descriptions the same as during the first trial.

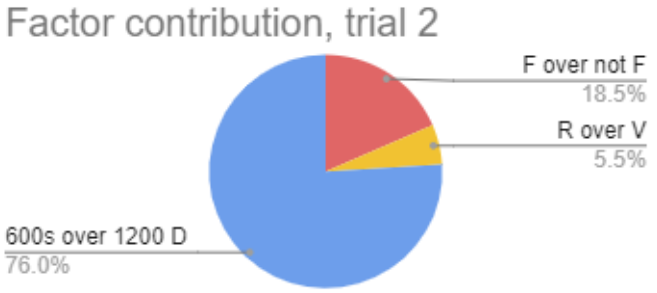

Figure 7.4. The factor contribution averages from trial 2 visualised as percentages.

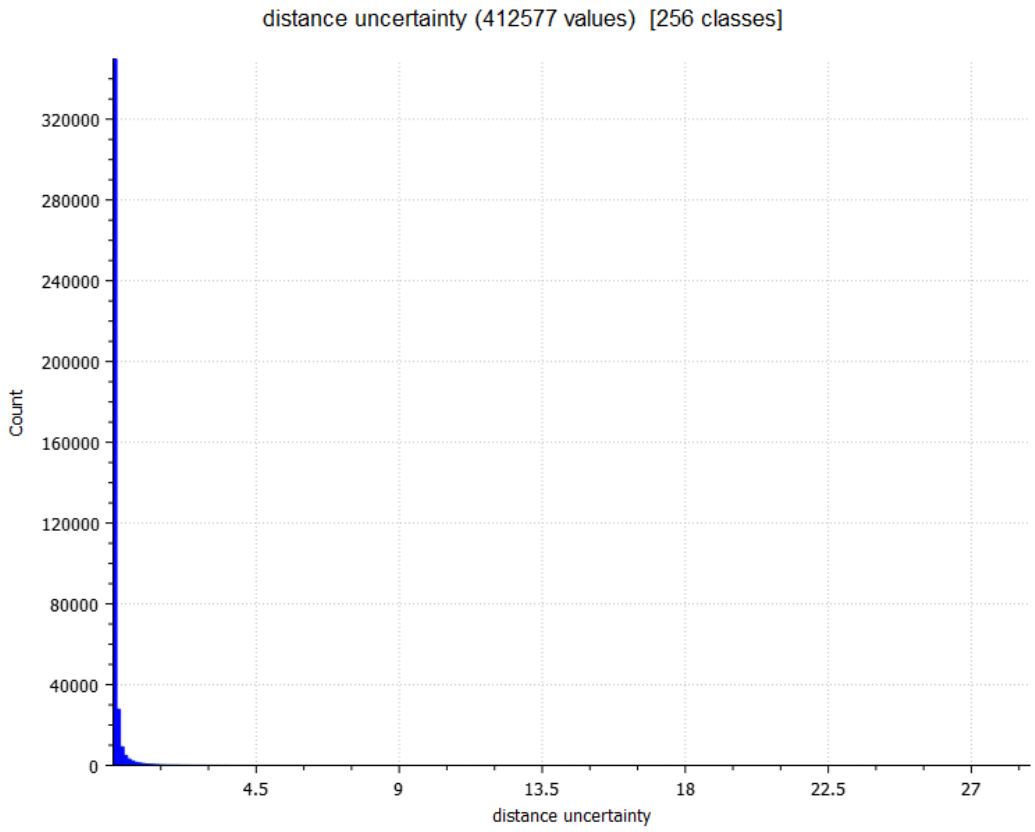

Figure 7.5. Uncertainty of final map.

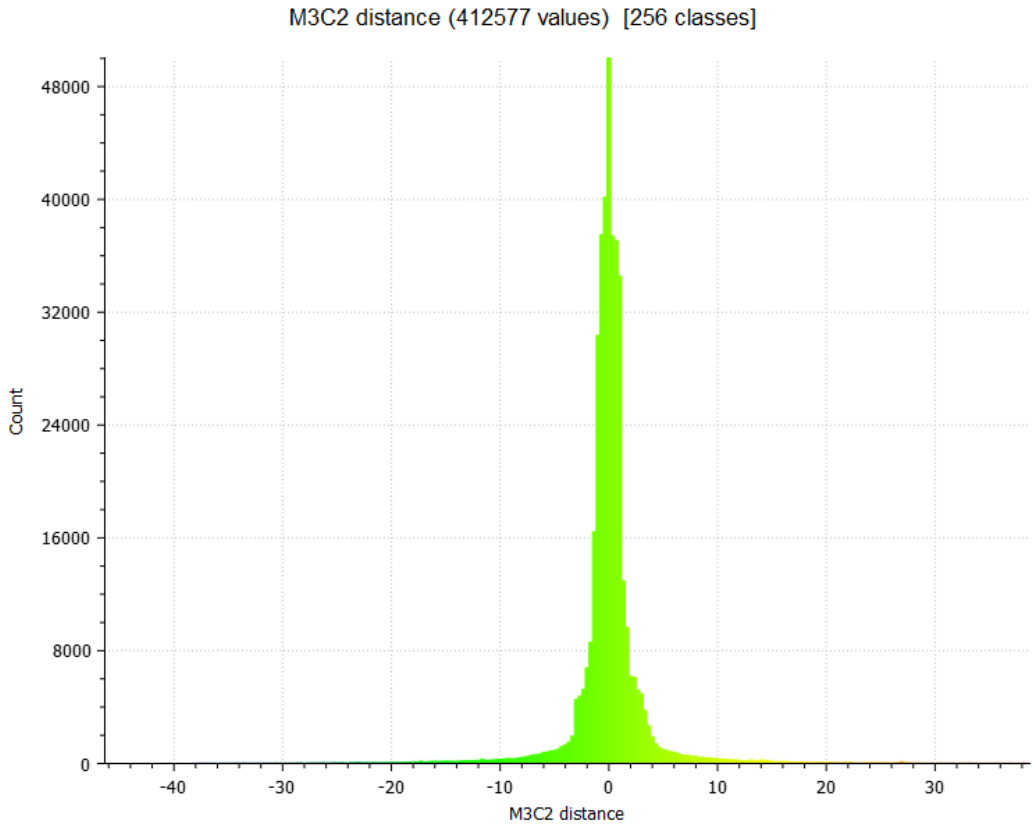

Figure 7.6. Histogram of the M3C2 distances of the final map [m].

To summarise the values for precision, Table 7.4 shows how the local precision first within a small region compare to a larger region, this was done to see how the precision changes when looking at object of smaller and larger size, the last one is showing the precision of the whole created map.

FINAL MAP	M3C2 Mean Distance [m]	M3C2 Standard Deviation[m]	C2C Mean Distance [m]	C2C Standard Deviation [m]
Local Precision [5m]	0.0034305	0.0096858	0.0405247	0.0664565
Local Precision [150m]	0.364851	0.659144	0.270341	0.338343
Global Precision	0.763199	1.026716	0.659189	0.577953

Table 7.4. Results with two types of local precision, and global precision using mean values.

7.3 Vector map

The vector map that was created in Vector Map Builder was proven sufficient for driving the RCV autonomously. A simple vector map with with a road segment around the M-building was made and was used when driving the RCV. A single

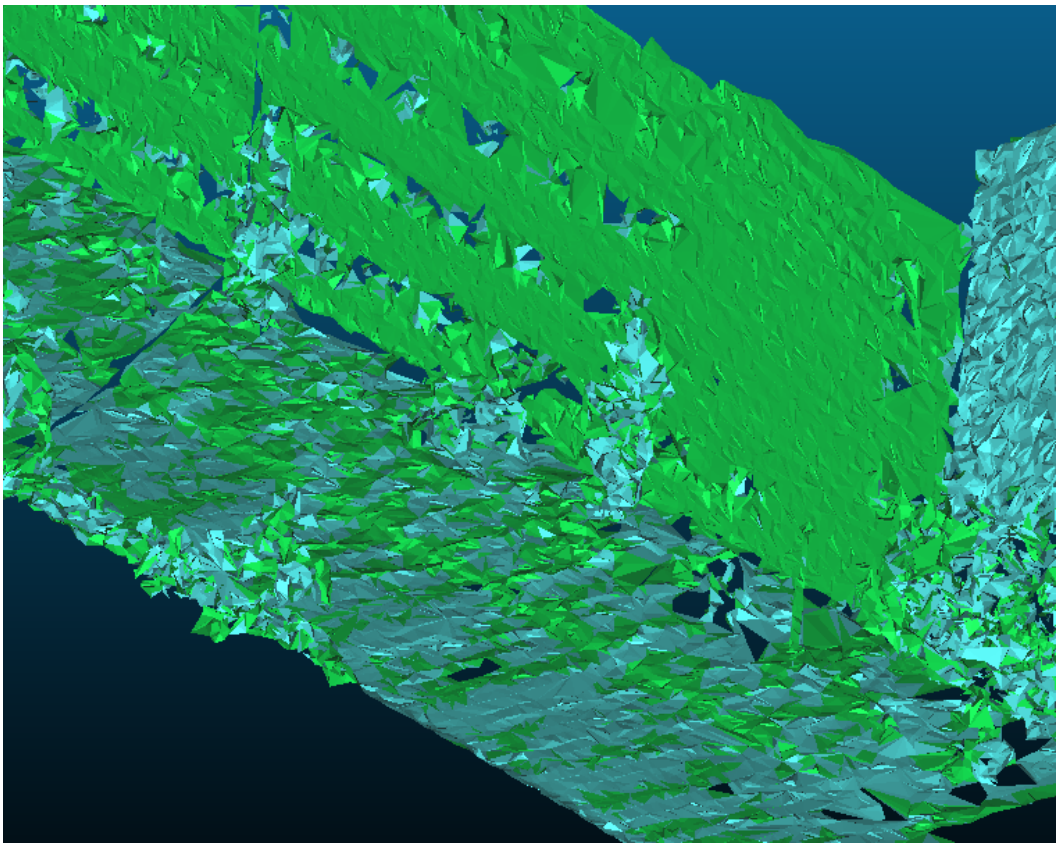

Figure 7.7. Part of mesh, depicting a corner of the B-building at KTH.

lane, displayed in Figure 7.10, was successfully used in the real world testing of the RCV in order to create the trajectory.

7.4 Meshing

The best resulting mesh was obtained after applying a voxel grid filter on the pcd before using the greedy triangulation method. The road and walls of buildings are mostly intact but they are not near a true representation of the real world, as seen in Figure 7.7. Trees, cars and other smaller objects are very messy. The mesh is implementable in the Prescan environment and could be utilized together with the original point cloud, but issues with localization arose when not near well defined features such as corners. Larger overviews of the mesh can be seen in figures 7.8 and 7.9.

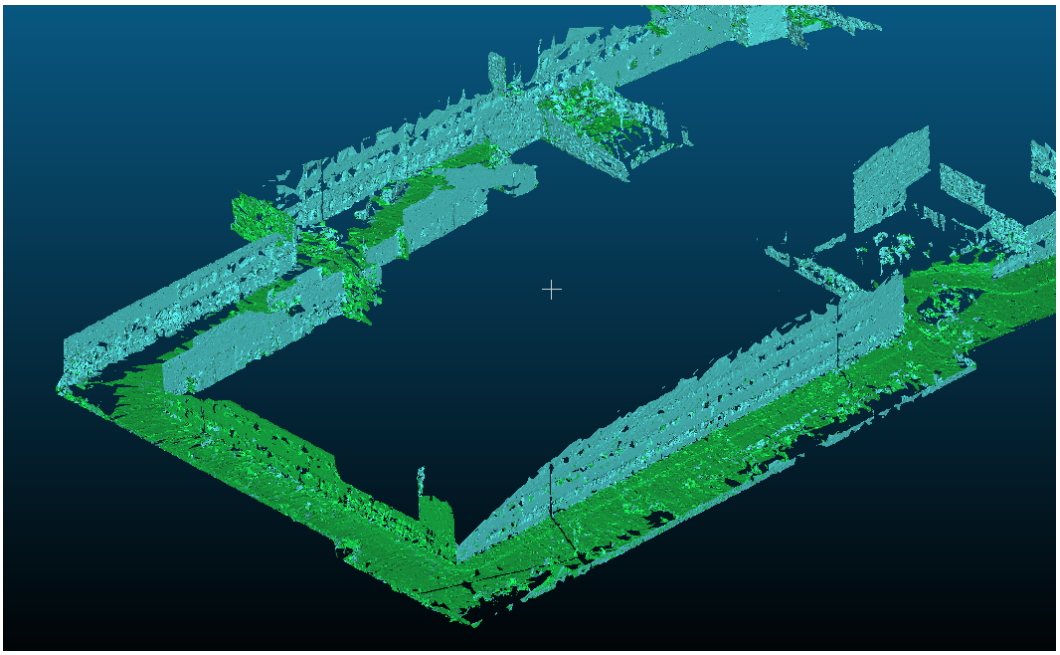

Figure 7.8. Part of the mesh depicting the M-buildings at the end of Brinellvägen at KTH.

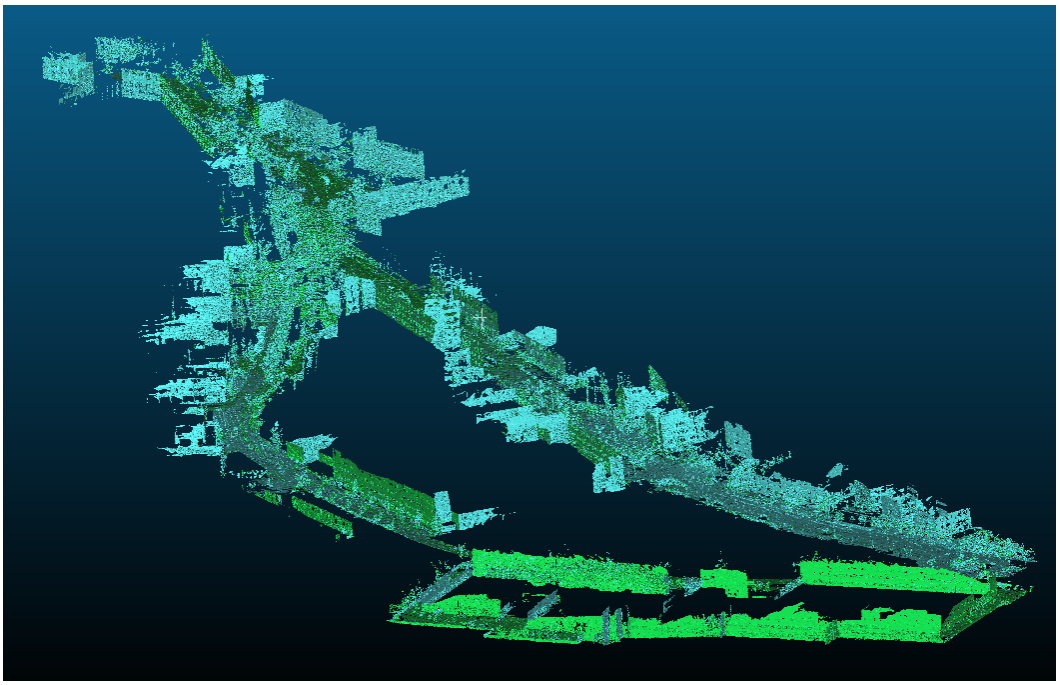

Figure 7.9. The entire resulting mesh.

7.5 AD-EYE on the RCV

The AD-EYE platform managed to localize itself on the map when using Rviz to point out the approximate position together with the NDT matching. The RCV had to be placed accurately on the road, so that it was placed directly on a road vector in the map. The autonomous driving was tested twice, where the RCV autonomously drove around a corner of a building, as depicted in the point cloud map in Figure 7.10. After passing the corner and reaching the straight path with a lot of trees, the AD-EYE platform lost the RCV's position in the map in both tests, and therefore the tests were ended at this point.

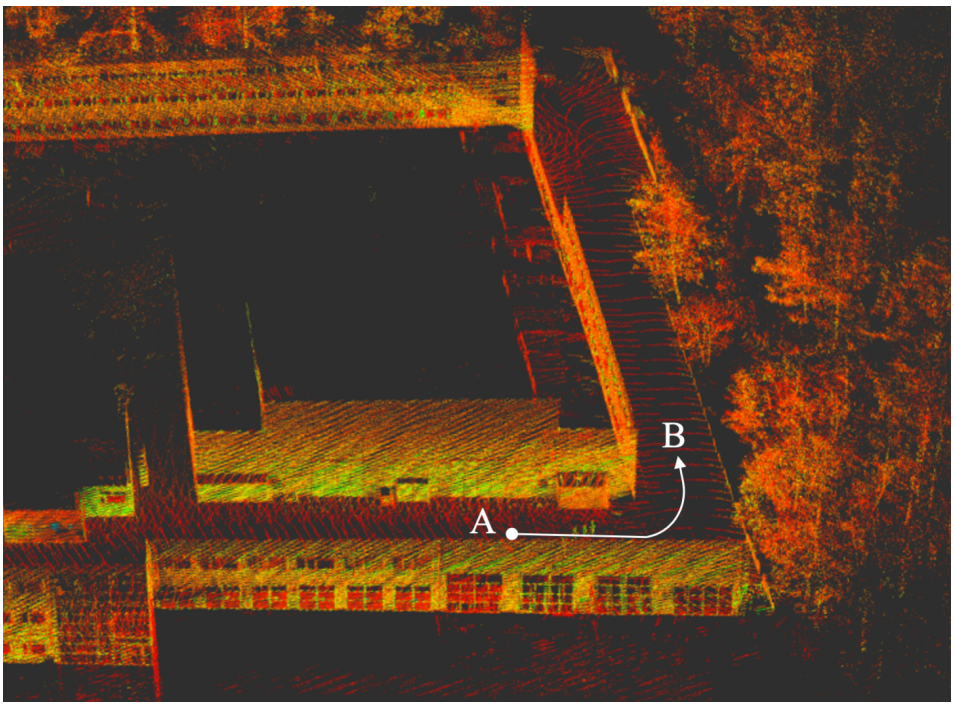

Figure 7.10. The route on KTH Campus of where the RCV drove autonomously together with the AD-EYE platform.

Chapter 8

Discussion and Conclusion

8.1 Discussion

8.1.1 Sensor mounting

The sensor rig fulfilled its requirements and had minimal to no issues when used during the mapping process. During longer (two hour) mapping sessions one or at the maximum two suction cups had partially or fully lost its grip but it was at no risk of fully dislodging. It was able to grip on most surfaces without any damage to the surface at all, but was very reliant on getting a good initial grip to grip at all. While water resistant and able to handle light rain it needs to be mounted on a dry surface, and should not be operated in rain which is not recommended anyways with the risk that it might interfere with the Lidar readings.

8.1.2 Point cloud map

The evaluation of the map has been done with the assumption that the map provided by 3D-interactive is a ground truth. The comparisons are done to get a difference between our map and the ground truth. However, it does not say much about its performance in its primary area of use, autonomous driving. One could do an evaluation on the map based on the fitness score of the normal distribution transform matching implementation used for localization in the RCV. This would provide another metric for evaluation more relevant to the application. It could also provide a larger discussion related to what global and local precision is needed for the localization in the RCV when GPS usage is implemented.

The local precision could have been further verified by manually measuring the distance between two static objects in the world, such as two walls and compare this to the same distance in our map. This means that any errors in the map considered as ground truth could be bypassed. This procedure was not done due to time constraints.

Although it is a tedious work to stitch maps together in CloudCompare, it provides another functionality to the mapping process. Namely, that the solution should be able to update smaller parts of an already existing map. The tool also gives the user a good way of segmenting out points that are not static and can therefore be regarded as noise, such as cars, people and more. This in turn means that a lot of work and time is inevitably spent on manual work to finalize a map with good quality.

Between the two methods for comparing the distance, the C2C method does not take the local surface orientation of the point clouds into account when computing the error and this leads to C2C global change being underestimated. However, the results produced by this method are still reasonably good and this could be explained by the carefully pre-processing done to isolate the areas of interest for the comparison and removing outliers. Also, because the reference cloud is dense and there is a good overlap between the clouds, the precision obtained is acceptable. To get a better accuracy in a global point of view, the local model based refinement can be tested for the C2C comparison. This method would give a statistically better result for the C2C measurement, but it would also give results that are locally inaccurate because the approximations computed on a few neighbors points are too simple and narrow. For the M3C2 algorithm, the distances are instead overestimated because the estimation accounts for both the RMS error computed in C2C and roughness and scanning aspects of the point clouds. To increase the accuracy of this algorithm, the conditions for point cloud registration that might generate noise and the parameters in the comparison need further research.

8.1.3 Vector map

In the original concept presented in 3.2.1 the idea was to use OpenStreeMap [23] to create a vector map. The process of creating a vector map was however chosen to Vector Map Builder [24]. The reason for this was that it was possible to create a vector map with the point cloud as a reference, which enabled the vector map to inherit the coordinate system and its orientation from the point cloud. Further, this made the process of matching the vector map with the point cloud in the AD-EYE platform much easier.

A comment on this is that using vector map builder as a tool for creating vector maps is not necessarily better than OpenStreetMap. The process for creating vector maps with Vector Map Builder is done manually and can be time consuming if the maps should cover a large area. Also, since the vector map is based on a point cloud, the vector map can inherit incorrect coordinates from the point cloud.

8.1.4 Meshing

While several tests regarding the best surface reconstruction algorithm were conducted it would probably be wise to either revisit some or spend more time tweaking

and adjusting the parameters. Since the mesh ”only” is used for the simulation it was not prioritized, and a sufficient result that presented the roads and buildings relatively clearly was settled upon. Researching other tools could perhaps lead to a better solution.

If chosen to utilize the same code it could probably be made more efficient, as it in its current state only utilizes one thread at a time. A larger mesh could even on the powerful computer seen in 5.1 take several hours to compute fully. While not a massive issue as a mesh (once the correct parameters are found) only needs to be computed once, it would still be of use at the least with testing parameters. Although a benefit now is the possibility to start several iterations of the same script for an overnight mesh-generation, this functionality could easily be achieved with a simple script. The parser flags could be expanded so that parameters for algorithms cloud be passed through as well, allowing for easier parameter testing without the need for compiling after each change. This in combination with the automation update, would allow generation of several meshes with different settings to be created one by one, aiding in the parameter evaluation.

Ultimately the mesh could not be said to sufficiently good for the simulation to run. Further testing would be need to fully determine if the localization errors were due to the meshes quality, flaws in the localization algorithm, or other issues in Prescan. While the mesh also could to some degree visually represent most buildings and roads, other objects such as trees and cars provided a lot of disturbance and could at times ruin areas of the mesh.

8.1.5 Simulation

Unfortunately, the simulation took more time than expected to work with. With several small issues arising during setup and installation, the work was often delayed. There were also issues when incorporating our own mesh and point cloud. The origin of the mesh and point cloud did not align and needed manual adjustment. The reason for this is that the origin of a point cloud is set when starting mapping, while the Prescan models have origin in their bottom right corner. This was solved by finding the coordinates of the point cloud corner, and moving the model in Prescan as such.

A large issue when implementing the mesh in the environment was that our models were shifted on the z-axis. This meant that our model was suspended in the air when imported. While not an issue for simulated driving, this ran into issues with the localization as the simulated lidar sometimes found several point on the ground of the simulation world. As this is not a part of the point cloud generated when mapping the real world it caused the localization to malfunction. It could be solved by remapping inside Prescan and generating a new point cloud but this would somewhat nullify the original work done. However it could perhaps be argued

that the mesh was a sufficient yield from the original point cloud regarding the implementation into Prescan.

8.1.6 AD-EYE on the RCV

There are many things with the integration of the AD-EYE platform with the RCV that could be discussed. As of now, the PI-controller have been tuned inside the lab at ITRL with the RCV lifted up on a stand. With this setup, the friction between the RCV's tires as well as the weight of the RCV was not taken into consideration during the tuning of this PI-controller. In reality, it will affect the driving in a real life situation. A low pass filter was also implemented in order to smoothen the steering control. The low pass filter introduced a small delay in the controller that affected the speed of which the RCV would turn. The plan was to fine tune the controller outside, but this was limited by the fact that the RCV could not drive for a long enough time period without losing its position in the map.

Furthermore, the same controller was used both for the braking and the acceleration. Since these two tasks have different dynamic characteristics for the vehicle, an improvement could be to tune two separate controllers.

The RCV drove autonomously a short distance, until its position was lost in the map. One of the reasons for this could be that the AD-EYE platform was not integrated to use the GPS data. The recorded ROS-bag had GPS data and the AD-EYE platform received the GPS location that the RCV registered with the IMU, however the GPS positions were not defined in the actual map. To implement the GPS data received from the RCV to enable localisation with GPS, the data would have to be transformed into x and y coordinates relative to the the map's coordinate system. The transformation would have to rotate the RCV's coordinate system so that the RCV's coordinate system matched the map's as well as offsetting the RCV's GPS location with the maps GPS location. To rotate the coordinate system, the first idea was to use the compass inside the IMU to get a true north of the map that could be compared with the compass reading from the RCV's IMU. However, the IMU that was used in this project was too noisy to get a proper reading of the compass. This led to that the implementation of GPS was discarded when driving. By integrating the GPS data the localisation could have been improved since, if the NDT-matching stopped working, the GPS could set an estimated location where the NDT-matching could start trying to match again.

Another reason for the RCV losing its position might be due to that the odometry is not a very exact measurement, since it accumulates error while driving from for example wheel slipping or if the velocity measurements are not exact [54]. There was also the issue that the odometry did not reset if the Simulink model was not rebuilt. So, if a new experiment would be run successionally, the RCV's Simulink model had to be rebuilt and re-flashed in order for the odometry to be reset.

Furthermore, the narrow path at point B in Figure 7.10 has only a straight wall to the left end trees to the right. This results in an environment without a lot of unique features that NDT-matching can use to match its position relative to the map with. The trees are not consistent objects and therefore hard to use as a reference for matching with the Lidar data. In an environment like this, the NDT-matching and odometry is not enough, hence using GPS could improve this result.

Another thing that was not implemented in the real world tests that were originally meant to be included was the functionality of road sign identification using cameras due to time restrictions and in favor of completing other sub-tasks.

The tests that were done on the RCV were, as mentioned in section 6.3, mainly verified by visually verifying that each test worked. This type of verification is far from optimal and proves at most that the test passes, however makes it hard to compare values of how well a test passes with future work.

In hindsight, integrating the AD-EYE platform onto the RCV should have been done earlier on in the project. A lot of focus was put on creating an own simulation experiment with our own map as a mesh. Since the mesh was not used by the RCV, these tasks should have been started simultaneously instead of trying to finish the simulation experiment first. This would have allowed us to incorporate the two functionalities mentioned above, namely GNSS localisation and camera integration. It would also have allowed us to fine tune the controller for the steering and make the RCV drive even better autonomously.

8.2 Ethics and sustainability

8.2.1 Development goals

The UN's sustainability development goals are an important aspect to incorporate in all research project [55]. The research of developing autonomous driving, especially for electric vehicles as in this project, is working towards more environmentally sustainable transportation, development of new innovative technology and decreasing the amount of vehicles needed in society. A project like this also provides teaching opportunities for students at technical universities and lay a knowledge foundation to continue similar development work in the future.

8.2.2 Safety

When working with autonomous technologies, there are less opportunities for direct human intervention. It is especially important to consider any risks such technologies might pose and whether such technologies may even pose a threat to human life. Autonomous vehicles specifically have already been responsible for killing humans, with a notable case in 2018 involving an Uber vehicle [56]. With an autonomous system making the decisions that led to human deaths - who is to blame? Fur-

thermore, the incentives for car companies may be such that they would prioritize time-to-market rather than perfecting the safety aspects of an autonomous vehicle.

With the work being done by this project and having successfully driven the first few meters autonomously, this question might eventually become relevant to AD-EYE to start addressing more directly. However, as part of this project, no such concerns were taken into account.

8.2.3 AD-EYE

Future work of the AD-EYE platform could include improving its set up procedure to make it more ”plug and play”. This would make it easier for others to implement, use and improve the platform. Another development aspect of the AD-EYE platform is to make it possible to have multiple autonomous vehicles interacting and driving autonomously around each other. For the platform to be further developed, along with the improvement of the vehicle integration, the use of localisation algorithms would benefit from being improved in a way that prevents the vehicle from losing its location in the map.

8.3 Conclusion

For the mapping, the team managed to successfully create ROS-bag with recorded topics from a Lidar, an IMU and a GPS. The team also managed to create a point cloud map from the recorded ROS-bag which have proven to be sufficient enough for driving the RCV autonomously. For evaluation, the map was compared with a separate map, created by 3D Interactive, that was regarded as a ground truth.

The results differ for local precision at smaller areas (for driving a distance of 5 m with mapping vehicle), in which the greatest precision is achieved, 0.0034-0.041 m depending on the method for distance measurement. For the local precision at a larger area (for driving a distance of 150 m with the mapping vehicle) the precision is 0.27-0.36 m. The global precision for the all of KTH campus is 0.66-0.76 m.

Taking the time to find sensor settings that produce a better map was shown to have a significant impact on final map accuracy with respect to ground truth. Filtering points based on a statistical outlier method has a smaller beneficial effect as well. The process of creating a 3D map demands, as of today, manual labor by stitching and matching parts of the entire map in order to get more globally precise map. As discussed in section 8.1.2, a way to bypass this would be to increase limitations of ROS-messages that is passed between ROS-nodes. However this in turn would increase the cost for computations of the system that is used for processing.

The tool that the team used to create vector maps was Vector Map Builder. This tool, as discussed in section 8.1.3 has limitations in terms of time consuming, manual

labor as well as inherited incorrectness from the point cloud that works as a reference when creating the vector map. Nevertheless, the tool is simple to use and able to solve the issue with relative coordinate systems between the point cloud map and vector map, thus making it a good tool for creating simple vector maps when testing.

For the meshing of the point cloud map, a variety of algorithms was considered and finally the Greedy Triangulation algorithm was picked. As mentioned in section 8.1.4, this task was less prioritized due to that the mesh was not a vital part for driving the RCV with help of the AD-EYE platform. Thus, further development should be considered for creating better solid structure mesh. However, the solution for generating these types of meshes can work as base for further development.

For the integration of the AD-EYE platform with the RCV, the tests showed that the integration worked as expected. The autonomous driving that took place on campus also showed that localisation was possible in a real life scenario, but with some limitations due to inactive GPS and odometry as discussed in section 8.1.6. In order to get the RCV to run fully autonomous for a longer period of time, the GPS and odometry integration needs further development and testing as well as the controller for steering. There is also a need to integrate cameras to get the AD-EYE's visual recognition algorithms for detecting traffic lights to work 8.1.6. However, the work that have been done so far will work as solid foundation for continuous development.

Chapter 9

Future work

9.1 Mapping

When creating maps, a limiting factor was the size of messages transmitted over the ROS network, as earlier discussed. This could be bypassed by changing the setup of ROS, however it would still be computationally demanding. A scenario with powerful computers could be evaluated and therefore reduce the amount of work in post-processing with CloudCompare. The solution would still be somewhat limited in size based on the available RAM in the computer, as well as having problem with scaling up. However if the scenario is to map KTH campus it should be considered, especially with the provided automatic script to create the point cloud.

Sensors used could be a topic for future development. Evaluating different sensors performance on the created map was not within the scope of the project, nonetheless it could potentially impact the quality of the map greatly.

For future work, manually driving the RCV could prove a valid way of evaluating both the RCV localization as well as the different maps, as earlier mentioned in section 8.1.2.

9.2 Vector map

For creating vector maps, the team utilized the Vector Map Builder tool [24]. This tool, as mentioned in 8.1.3, is not optimal when dealing with large maps due to the manual process, as well as the potential inherited incorrectness when using a point cloud as a reference for creating vector maps. For creating a more scalable solution when creating vector maps, future work should consider to expand on this field and use a a verified source for vector maps. In order to do so, there must also exist a solution for matching this vector map with a corresponding point cloud map.

9.3 AD-EYE on the RCV

The control for the RCV's throttle and braking was created and tuned with the RCV lifted up on a stand. A more careful tuning should be consider when continuing the development of the integration. Also, a separate controller for the braking should be consider, as discussed in 8.1.6 due to the various dynamical properties between the throttle and braking.

As mentioned in section 8.1.6, the GPS was not implemented in the final solution. This part should be further developed to ensure a better localisation when driving in real life.

The low pass filter for the output of the steering angle, mentioned in 6.3, is now implemented inside the Simulink file. The filter was designed to the point that it gave a decent result, however it could be redesigned while taken a deeper consideration to the dynamics of the RCV to minimize the latency.

Also, as of now, cameras has not been integrated with the AD-EYE platform that as a result makes AD-EYE's visual recognition algorithms for detecting traffic lights useless. These cameras must be integrated in order to drive the RCV autonomously in an environment with traffic lights, since without these cameras the driving could result in a catastrophic, and even lethal, event.

Bibliography

[1] (2019) Volvo Cars and Uber present production vehicle ready for self-driving. [Accessed 19-Sep-2020]. [Online]. Available: https://www.media.volvocars.com/global/en-gb/media/pressreleases/254697/volvo-cars-and-uber-present-production-vehicle-ready-for-self-driving

[2] Future of driving. [Accessed 19-Sep-2020]. [Online]. Available: https://www.tesla.com/autopilot

[3] We're building the world's most experienced driver. [Accessed 19-Sep-2020]. [Online]. Available: https://waymo.com/

[4] AD-eye. AD-EYE Automated Driving Simulation Platform. [Accessed 19-Sep-2020]. [Online]. Available: https://www.adeye.se/home

[5] N. Mohan and M. Törngren. (2019) Ad-eye: A co-simulation platform for early verification of functional safety concepts. [Online]. Available: https://arxiv.org/abs/1912.00448

[6] V. Hanefors, C. Jensen, J. Jungåker, F. Larsson, H. Lind, A. Phan, J. Röing, M. Svjatoha, and A. W. de Val, "AD-EYE state of the art report," KTH - The Royal Institute of Technology, Stockholm, Sweden, Tech. Rep., 2020.

[7] 3D Interactive. [Accessed 19-Sep-2020]. [Online]. Available: https://3dinteractive.se/sv/index.html

[8] Starlino. A guide to using IMU (accelerometer and gyroscope devices) in embedded applications. [Accessed 13-Dec-2020]. [Online]. Available: http://www.starlino.com/imu_guide.html

[9] Wikipedia. Global positioning system - wikipedia. [Accessed 19-Sep-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Global_Positioning_System

[10] N. O. Service. (2020) What is lidar? [Accessed 13-Dec-2020]. [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html?fbclid=IwAR03wHecV4tQRpdK7KytxQXNgd64ZIlaIBeiQtLw3jvILG2qEAJCJ8J1Aqo

[11] L. RADAR. (2018) Advantages and disadvantages of lidar. [Accessed 13-Dec-2020]. [Online]. Available: http://lidarradar.com/info/advantages-and-disadvantages-of-lidar?fbclid=IwAR0zP-COECUGSmletfBbQuVSjiE5hxcvBtuI8LjDQvfcoGu8PmRGFB5T8EY

[12] Velodyne Inc., *VLP16 User Manual 63-9243 Rev. D*, [Released 2017-12-20].

[13] ROS Wiki. (2015) Rosbag. [Accessed 24-Sep-2020]. [Online]. Available: http://wiki.ros.org/rosbag

[14] Andy Wilson. (2015) Rqt_bag. [Accessed 24-Sep-2020]. [Online]. Available: http://wiki.ros.org/rqt_bag

[15] Autoware. Autoware home page. [Accessed 24-Sep-2020]. [Online]. Available: https://www.autoware.org/

[16] ——. Autoware ai home page. [Accessed 24-Sep-2020]. [Online]. Available: https://www.autoware.ai/

[17] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems*, 04 2018, pp. 287–296.

[18] T. C. A. Revision. Algorithm walkthrough for tuning. [Accessed 5-Nov-2020]. [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html

[19] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[20] CloudCompare. 3D point cloud and mesh processing software. [Accessed 11-December-2020]. [Online]. Available: http://www.cloudcompare.org/

[21] ——. Cloud-to-Cloud Distance. [Accessed 13-December-2020]. [Online]. Available: https://www.cloudcompare.org/doc/wiki/index.php?title=Cloud-to-Cloud_Distance/

[22] ——. ICP. [Accessed 13-December-2020]. [Online]. Available: https://www.cloudcompare.org/doc/wiki/index.php?title=ICP

[23] OpenStreetMap. OpenStreetMap förser tusentals webbsidor, mobilappar och fysiska apparater med kartdata. [Accessed 4-Nov-2020]. [Online]. Available: https://www.openstreetmap.org/about

[24] I. Tier IV. Autoware tools. [Accessed 24-Sep-2020]. [Online]. Available: https://tools.tier4.jp/

[25] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[26] PCL. Point cloud library tutorials. [Accessed 4-Nov-2020]. [Online]. Available: https://pcl-tutorials.readthedocs.io/en/latest/index.html

[27] ROS. Wiki and tutorials for ROS and PCL. [Accessed 4-Nov-2020]. [Online]. Available: http://wiki.ros.org/pcl_ros/

[28] A. Keiser, A.-M. H. B. Bech, and F. V. Bærentsen, "Mesh reconstruction using the point cloud library," Aalborg University Copenhagen, Aalborg, Denmark, Tech. Rep., 2015.

[29] J. Jansson, "Planar minimum-weight triangulations," Lund University, Lund, Sweden, Tech. Rep., 1995.

[30] Wikipedia. (2020) Marching cubes — Wikipedia, the free encyclopedia. [Accessed 14-November-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Marching_cubes

[31] G. H. Weber, G. Scheuermann, H. Hagen, and B. Hamann, "Exploring scalar fields using critical isovalues," in *IEEE Visualization, 2002. VIS 2002.*, 2002, pp. 171–178.

[32] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, ser. SGP '06. Goslar, DEU: Eurographics Association, 2006, p. 61–70.

[33] PCL. Removing outliers using a StatisticalOutlierRemoval filter. [Accessed 4-Nov-2020]. [Online]. Available: https://pcl-tutorials.readthedocs.io/en/latest/statistical_outlier.html#statistical-outlier-removal

[34] ——. Filtering a PointCloud using a PassThrough filter. [Accessed 4-Nov-2020]. [Online]. Available: https://pcl-tutorials.readthedocs.io/en/latest/passthrough.html#passthrough

[35] C. Dunkel. Applying filters on lidar point clouds. [Accessed 5-Nov-2020]. [Online]. Available: https://www.htw-mechlab.de/index.php/applying-filters-on-lidar-point-clouds/

[36] Point Cloud Library Tutorial. Downsampling a PointCloud using a VoxelGrid filter. [Accessed 11-December-2020]. [Online]. Available: https://adioshun.gitbooks.io/pcl/content/Tutorial/Filtering/pcl-cpp-downsampling-a-pointcloud-using-a-voxelgrid-filter.html

[37] AD-eye. Overview and features. [Accessed 19-Sep-2020]. [Online]. Available: https://www.adeye.se/overview-and-features

[38] Siemens. Follow a systematic physics-based simulation approach. [Accessed 29-Okt-2020]. [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html

[39] dSpace. MicroAutoBox II. [Accessed 09-December-2020]. [Online]. Available: https://www.dspace.com/en/inc/home/products/hw/micautob/microautobox2.cfm#144_24389

[40] N. M. Garcia, F. Gil, B. Matos, C. Yahaya, N. Pombo, and R. I. Goleva, "Keyed user datagram protocol: Concepts and operation of an almost reliable connectionless transport protocol," *IEEE access*, vol. 7, pp. 18 951–18 963, 2019.

[41] Thule - roof rack components and accessories. [Accessed 13-Dec-2020]. [Online]. Available: https://www.thule.com/sv-se/roof-rack/roof-rack-components-and-accessories

[42] Conrad.se. Mantona Sugkoppsfäste - Passar: GoPro, Sony Actioncams, Actioncams. [Accessed 5-Nov-2020]. [Online]. Available: https://www.conrad.se/p/mantona-sugkoppsfaeste-passar-gopro-sony-actioncams-actioncams-1528139?gclid=CjwKCAiA4o79BRBvEiwAjteoYEPiD-BHvx_qIvKEPE6EydVVMgfL6FRs0vFhLUt2RDCIzyf09xvsgRoCdH4QAvD_BwE&gclsrc=aw.ds&vat=true

[43] Wikipedia. V-model (software development). [Accessed 13-Dec-2020]. [Online]. Available: https://en.wikipedia.org/wiki/V-Model_(software_development)

[44] ROS Wiki. (2014) Ros/concepts. [Accessed 23-May-2020]. [Online]. Available: http://wiki.ros.org/ROS/Concepts

[45] P. C. Library. pcl. [Accessed 12-Dec-2020]. [Online]. Available: https://pointclouds.org/

[46] Siemens. Solid Edge. [Accessed 13-Dec-2020]. [Online]. Available: https://solidedge.siemens.com/en/

[47] Mathworks. MATLAB. [Accessed 13-Dec-2020]. [Online]. Available: https://se.mathworks.com/products/matlab.html

[48] CloudCompare. M3C2 (plugin). [Accessed 13-December-2020]. [Online]. Available: https://www.cloudcompare.org/doc/wiki/index.php?title=M3C2_(plugin)

[49] A. J. Crawford, D. Mueller, and G. Joyal, "Surveying drifting icebergs and ice islands: deterioration detection and mass estimation with aerial photogrammetry and laser scanning," *Remote Sensing*, vol. 10, no. 4, p. 575, 2018.

[50] Nvidia. Nvidia drive px 2 autocruise mechanical and installation guide. [Accessed 09-Dec-2020]. [Online]. Available: https://developer.nvidia.com/driveworks/files/drive-px2-auto-cruise-mechanical-installation

[51] Elfa. 1n5361bg - zenerdioder 5w 27v 500na surmetic 40, on semiconductor. [Accessed 09-Dec-2020]. [Online]. Available: https://www.elfa.se/sv/zenerdioder-5w-27v-500na-surmetic-40-on-semiconductor-1n5361bg/p/17006752?track=true&no-cache=true

[52] P. S. Foundation. socket — low-level networking interface. [Accessed 09-Dec-2020]. [Online]. Available: https://docs.python.org/3/library/socket.html

[53] J. O'Quin. velodyne. [Accessed 09-Dec-2020]. [Online]. Available: http://wiki.ros.org/velodyne

[54] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Transactions on robotics and automation*, vol. 12, no. 6, pp. 869–880, 1996.

[55] UN. The 17 goals. [Accessed 12-December-2020]. [Online]. Available: https://sdgs.un.org/goals

[56] www.forbes.com. What happens when self-driving cars kill people? [Accessed 12-Dec-2020]. [Online]. Available: https://www.forbes.com/sites/cognitiveworld/2019/09/26/what-happens-with-self-driving-cars-kill-people/?sh=65960620405c

Appendix A

GANTT schedule

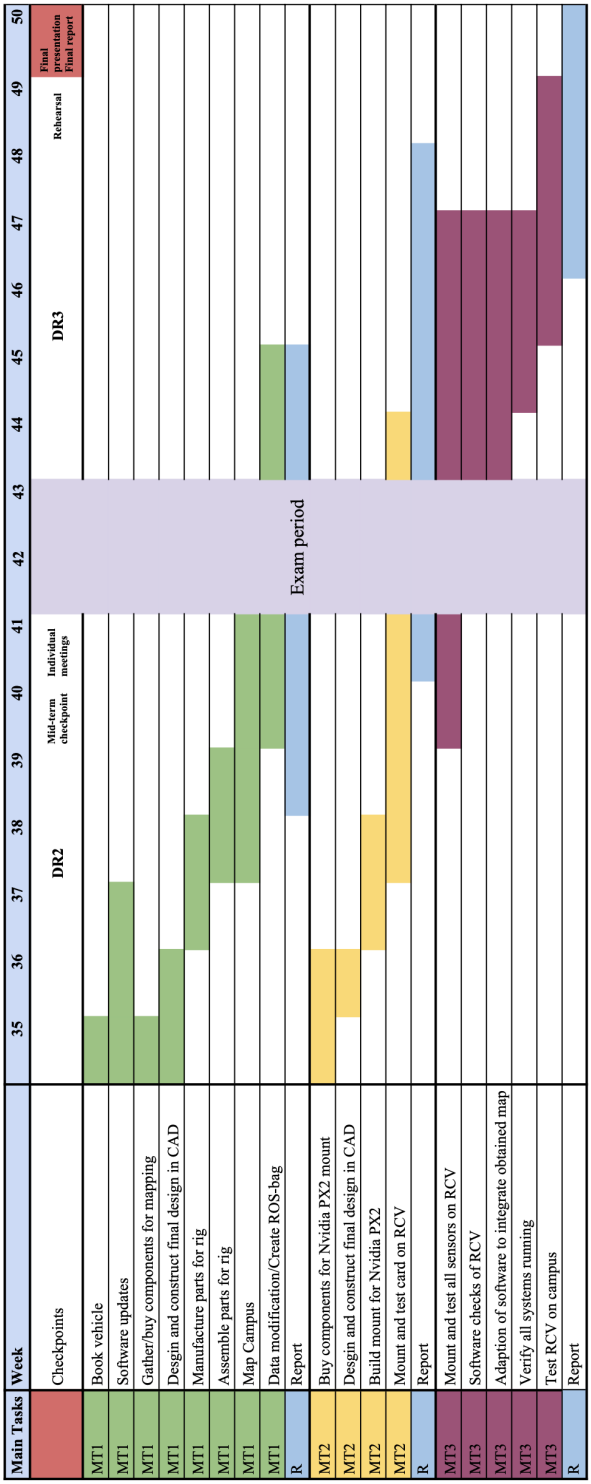

Figure A.1. Detailed GANTT schedule for fall semester with activities in the rows and time frame in the columns.

Appendix B

Simulink

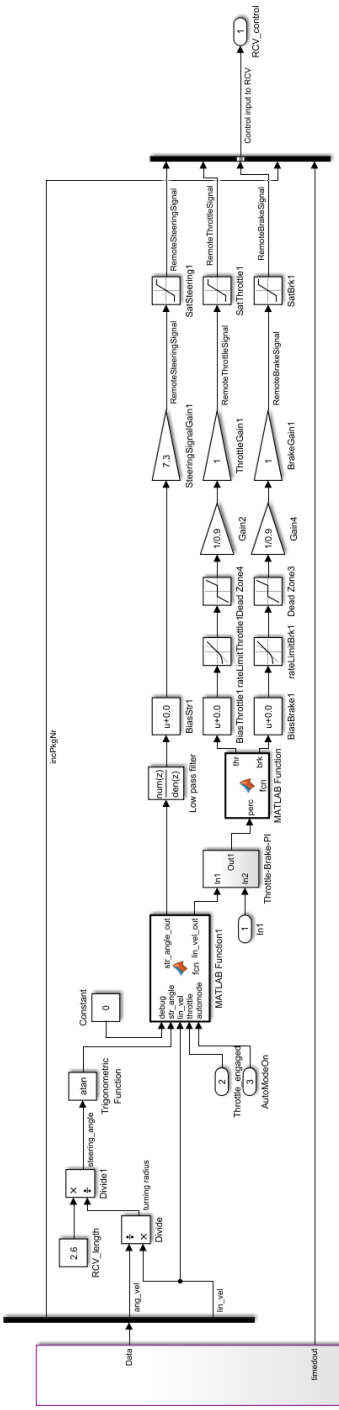

Figure B.1. UDP input, from PX2 to RCV, and controller for the RCV.

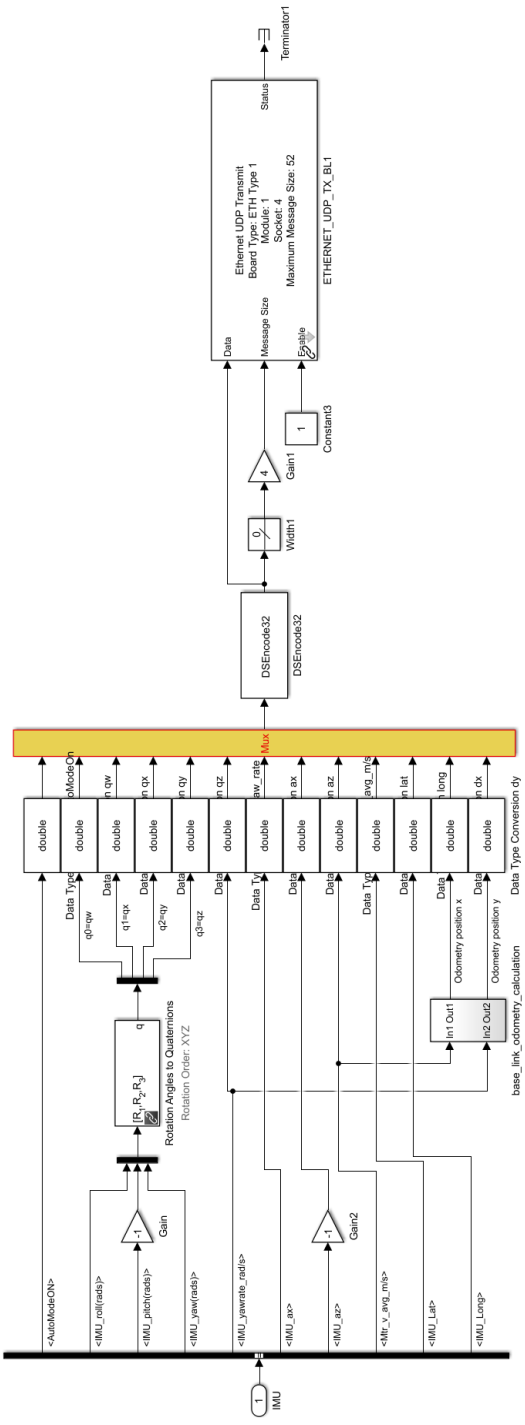

Figure B.2. Output UDP-communication in Simulink, from RCV to PX2.

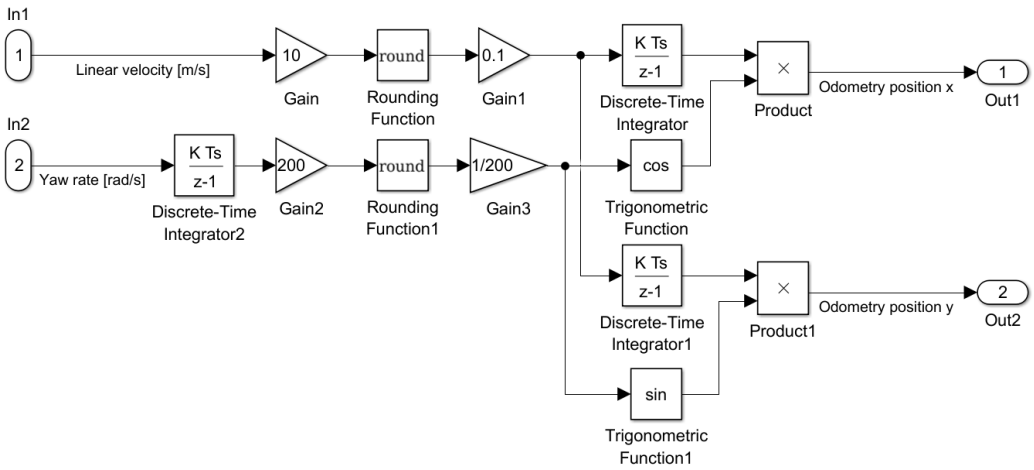

Figure B.3. Odometry calculations in Simulink.