Situation Awareness for an Autonomous Vehicle

Oliver Lynggaard Topp

Master's thesis

April 2018

DTU Electrical Engineering
Department of Electrical Engineering

Situation Awareness for an Autonomous Vehicle

SITUATIONSBEVIDSTHED FOR ET AUTONOMT KØRETØJ

Report written by:
Oliver Lynggaard Topp

Advisor(s):
Jens Christian Andersen
Henning Si Høj
Jesper Schramm

DTU Elektro
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

elektro@elektro.dtu.dk

Project period:	2. October 2017 - 23. April 2018
ECTS:	35
Education:	MSc
Field:	Electrical Engineering
Class:	Public
Edition:	1. edition
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.

Abstract

Autonomous vehicles have seen significant development in recent years, and are leaving the prototype stage. This development is made possible because of technological progress and is driven by both environmental and safety incentives.

The student-driven project, DTU Roadrunners, at the Technical University of Denmark is going to compete at the Autonomous UrbanConcept Competition at the Shell Eco-Marathon 2018, for the first time. This competition will consist of challenges designed explicitly for self-driving vehicles.

This thesis is aimed to extend the perception capabilities of the existing autonomous platform of the DTU Dynamo, the competing vehicle of DTU Roadrunners. The primary focus of this project is LiDAR perception, and a Velodyne VLP-16 LiDAR has been implemented into the existing system. The LiDAR is used to inspect the local environment, particularly for detecting elements of the racing track. Processing the raw LiDAR data is performed and combined with odometry. Graph-based ground detection is implemented and used to achieve robust obstacle detection. Obstacle tracking and maintenance are successfully performed. The solutions are investigated in various in environments and designed to create a foundation for navigation and planning.

Keywords
Autonomous Vehicle, Mobile Robots, Self-driving Vehicle, LiDAR Perception, Graph-based Ground Detection, Obstacle Detection, Obstacle Tracking, Situation Awareness.

Table of Contents

Chapter Descriptions

1. **Introduction**
   A brief introduction to the premises of the problem statement of this thesis.
2. **Design & Analysis**
   In-depth background analysis of the challenges of the project and previously performed work for the AUC. Furthermore, a detailed analysis of the project requirements and possible solutions, outlining the necessary work.
3. **LiDAR Data Acquisition**
   Description of sensor data retrieval and initial processing of raw data.
4. **Sensor Fusion**
   Description and brief testing of the advantageous fusion of odometry and LiDAR data.
5. **Ground Detection**
   Detailed description and evaluation of ground classification approach, using LiDAR data.
6. **Downsampling & Segmentation**
   Brief descriptions of downsampling of LiDAR data and the segmentation performed for obstacle point clouds.
7. **Obstacle Generation**
   Detailed description and evaluation of information reduction, converting 3D point clouds to 2D Hulls.
8. **Obstacle Tracking**
   Detailed description and evaluation of proposed approach for obstacle tracking and information maintenance.
9. **System Implementation Overview**
   System overview and explanation of the combination of all the previous processing steps. Includes a description of final data preparations for its intended usage in navigation and planning.
10. **System Tests & Results**
    Tests on the complete system implementation are performed.
11. **Conclusion**
    Summary of the results obtained throughout the thesis, in respect to the intended goals.
12. **Future Work**
    Proposals for important future work, based on current and projected future perception requirements for DTU Roadrunners.

1 Introduction

Though the concept of autonomous vehicles has been around many years, modern technology has enabled great development for autonomous vehicles in the recent years. Autonomous cars are leaving the prototype stage, with either full or high levels of autonomous driving. Several tech giants and car manufacturers such as Google, Tesla, Intel, and Nvidia, are all working on the topic, based on both environmental and safety incentives. This development generates more widespread access to the necessary technology, enabling projects such as this thesis.

1.1 Background

This Master's thesis focuses on the perception for autonomous driving, specifically for DTU Roadrunners project at the Technical University of Denmark (DTU). The DTU Roadrunners project is managed by the DTU Department of Mechanical Engineering, and works in collaboration with the DTU Automation and Control to enable the autonomous aspects of the project.

1.1.1 Shell Eco-Marathon

The Shell Eco-Marathon (SEM) is an annual competition, hosted by the Royal Dutch Shell company on a global scale. The main competition (the Mileage Challenge) is focused on achieving highest fuel efficiency for driven vehicles [1]. In 2017, more than 200 student-driven teams from around the world, raced their specialized cars, competing for lowest energy consumption. The SEM is held in Europe, Asia, and America, and in July 2018, the European competition takes place in London.

Competitors are divided into two classes of cars: Prototype, and Urban Concept (UC). Prototype cars are tiny impractical vehicles, usually with the driver laying down and only three wheels. An example of a prototype class car is the DTU Innovator shown in Figure 2. UC cars are restricted to more common car characteristics, such as four wheels and an upright driver seat, as well as security specifications such as brake range, turn radius, lighting, driver's field of view, etc. An example is the DTU Dynamo shown in Figure 1.

Figure 1 – The DTU Dynamo 13 (front), and members of DTU Roadrunners team 2017 – 2018

As of 2018, the SEM will also include the Autonomous UrbanConcept Competition (AUC), an additional competition specifically for self-driving vehicles. The competing teams will use their UC class cars and compete in five challenges with distinct requirements related to autonomous driving. To qualify for the

AUC, teams must attend a qualification event in Paris, in May 2018. The AUC is expected to continue as an annual competition, with yearly gradual increases in difficulty for successful teams.

Figure 2 – The DTU Innovator 2016. (Source: https://sg.makethefuture.shell)

1.1.2 DTU Roadrunners

Since 2004, the student driven project, DTU Roadrunners, at the Technical University of Denmark (DTU) has been participating in the Mileage Challenge at the SEM. In recent years, the involved students have mainly been Mechanical and Electrical Engineering students at DTU. The DTU Roadrunners team of 2017 – 2018, contains a group of students, dedicated to work on the autonomy of the competing car, the DTU Dynamo. At the SEM in London July 2018, DTU Roadrunners are expected to compete in both the Mileage Challenge and the AUC, which outlines the requirements of this thesis.

1.1.3 DTU Dynamo

The DTU Dynamo has been developed through multiple iterations, and the latest iteration (13th), is shown in Figure 1. The DTU Dynamo is the UC car of DTU Roadrunners, and has previously won the Mileage Challenge in the combustion engine category ten times. The engine runs on ethanol, and set the world record in 2015 with a fuel efficiency equivalent to 665 km/L of gasoline.

During 2016 – 2017, the DTU Dynamo was expanded for autonomous use in preparation for the autonomous challenges at SEM 2018. Although this expansion included numerous features and created a platform for making the car self-driving, the car was still lacking several implementations for completing the AUC challenges.

1.2 Problem Statement

The following problem statement defines the motivations and goals behind this Master's thesis:

The goal of this project is to expand and improve the autonomous capabilities of the DTU Dynamo vehicle, with the intention of competing in the Autonomous UrbanConcept Competition at the Shell Eco-Marathon 2018. The desired capabilities are heavily based on the 5 challenges in the competition, but are additionally preferred to enable further development. In this thesis, the main focus surrounds the situation awareness of the autonomous car, primarily based on LiDAR sensing.

Due to the complex nature of self-driving cars, the development of the autonomy in the DTU Dynamo contains many aspects and is conducted by several students in the DTU Roadrunners project. Collaboration with these students will be important to complete the project.

Because LiDAR is such a prominent part of this thesis, the term is briefly described in the following: LiDAR (**Li**ght **D**etection **A**nd **R**anging) is an active exteroceptive sensor, used for measuring distance to objects. By emitting laser pulses and timing the echoed light, the distance to a single point can be derived. Moving a LiDAR sensor, typically by rotation, 2-dimensional scans can be achieved. Some modern LiDAR scanners combine multiple LiDAR sensors with movement, in a single LiDAR unit, to enable 3-dimenstional

distance scans. Such 3D LiDAR scanners generate multiple parallel scan lines, referred to as scan layers or channels. A collection of LiDAR measurement, represented as points in a multidimensional space, is referred to as a point cloud.

2 Design & Analysis

In this section the requirements to achieve the goals of this thesis, are analyzed outlining key areas of interest to be discussed in the following sections.

2.1 Background Analysis

The rules for the Autonomous UrbanConcept Competition 2018 in the Shell Eco-Marathon have through the duration of this entire project been an ongoing development. As it will be the first time this competition is held, the rules were not defined from the beginning, and the organizers decided to develop these rules in collaboration with the competing students. This included several online meetings and two workshops, one in Cologne Germany in November 2017 and one in Amsterdam Holland in March 2018. Despite the efforts and everyone involved, the final rules have not been defined as of the writing of this thesis. The foundation for the requirements and rules in the competition are based on the latest revision of the rules [2] along with the knowledge of the direction of development, obtained throughout the collaboration with Shell. Finally, all rules from the regular UrbanConcept Competition [3] will also apply in the AUC, unless otherwise specified. For a car to compete in the AUC, it must also qualify and compete in the UrbanConcept Mileage Challenge.

2.1.1 Challenge Requirements

The autonomous competition is split into five challenges at level 1. Level 1 being the only level at AUC 2018, but more levels are expected at future events. Each team must complete two challenges at the qualification event, to qualify for the AUC at SEM 2018. The challenges are designed to vary in both technological requirements and difficulty. If a team completes three of the challenges at the SEM AUC, they will progress to the subsequent level the following year, which contains new and even more difficult challenges. These higher-level challenges are yet to be defined, as everyone begins at level 1 the first year of competing. With the expectation to reach level 2 after the 2018 competition, much of the work on this year's autonomous car has been performed with this in mind. It is desired to make robust and expandable solutions to create solid groundwork for future competitions.

At the AUC, the challenges will be on the same standard track as the rest of the competitions at the SEM, or on modified track sections. The track is 970 m long and will include inclinations since the track is partially set up on closed off traffic roads. For all five challenges the track will have specified barriers on both sides. The barriers are of 0.5 m height and can include minor gaps, though the size of the gaps is not specified further. No contact with the track barriers is permitted, cars are limited to a top speed of 25 km/h, and every car drives alone on the track. The minimum track width is 6 meters unless otherwise specified.

The 5 challenges' key points can be summed up with the following:

1. **Autonomous Distance**
   The goal of this challenge is for the car to autonomously drive up to one lap on the event track. This must be achieved with an unspecified minimum avg. speed, and fuel efficiency will enable bonus points.
2. **Complex Track Layout**
   The car must navigate through one out of six given possible complex track segments, illustrated in Appendix 14.1. Two of the track layouts will include on track obstacles. The selected complex track will be given 48 hours in advance, and the maximum time for completion is 30 seconds.
3. **Maneuverability**

The car must drive on a straight track section, between 60 to 80 meters long and find a dedicated parking rectangle, marked on the track. The rectangle is 4 m long and 3 m wide, and a block of 3 m width and 0.5 m height, is placed right after the end of the rectangle. Each car must park with all four wheels inside the rectangle within 90 seconds, as close to the block as possible without making contact.

4. **Obstacle Avoidance**
   The car must drive on a straight track section, between 60 to 80 meters long and detect gates and road markings. The objective is to drive through the gates and over the road markings. The gates' poles are specified to be of 45 cm in diameter and 1 m high. The road markings are squares of 30 cm by 30 cm.
5. **The Last-Minute Challenge**
   The car must drive along the track containing four parking rectangles. Each rectangle will be only be marked visually on the road and contain a number from 1 to 4. In contrast to challenge 2, the rectangles will not have a block at the end. Before the beginning of the challenge, a number from 1 to 4 will be communicated to the car, and the car must locate and park within the respective rectangle.

2.1.1.1 Braking Range

It is stated in the rulebook [3] of the SEM, that an UrbanConcept vehicle must meet certain braking requirements, specified by force and road inclination. The brakes of the DTU Roadrunners' vehicle are always designed to meet these requirements. Additionally, DTU Roadrunners has an internal requirement of achieving a maximal autonomous braking distance of 20 m from 25 km/h. Using a 20 m braking distance should be more than sufficient, and will also prepare the autonomous system for future competitions that might allow higher velocities.

2.1.2 The Previous Platform

DTU Dynamo contains various custom PCBs with ARM based microcontrollers, which handles engine control and auxiliary functions in the car. This system of electronics for non-autonomous driving is referred to as the *basic system*. Throughout 2016/2017, preparations for the AUC were initiated, and an initial platform for the autonomy in the DTU Dynamo was developed [4]. This platform referred to as the *previous platform,* consists of modular additions for autonomy, added to the *basic system*. The *previous platform* was the foundation for the hardware and software, used for further development throughout 2017/2018, in preparations for the AUC 2018. Hence, the *previous platform* is a basis for the work in this thesis.

2.1.2.1 Hardware

Since it is required for cars competing in the AUC to also compete in the Mileage Challenge, the *previous platform* was designed to be modular and removable. This was done to avoid affecting the fuel efficiency when driving in the Mileage Challenge.

The essential autonomy components of the *previous platform* were:

- **Computer**
  A small-form-factor computer with low power consumption was implemented to control the autonomous system (BRIX GB-BKi5A-7200). The computer is running a 64-bit Ubuntu system with ROS, powered by a dual core Intel Core i5-7200U CPU[1], with 8 GB RAM and SSD storage.

[1] Upgraded in March 2018 to an Intel NUC with an Intel Core i7-8650U CPU.

- **Steering Control**
  A stepper motor with encoder was mounted on the steering wheel shaft, along with a potentiometer for absolute position feedback. This allows for a turn rate of approximately 25 degrees per second.
- **Braking Control**
  A stepper motor with encoder was mounted to the brake pedal, along with two hydraulic pressure sensors for feedback.
- **GPS**
  GPS with RTK capabilities was added to the car (SwiftNav Piksi Multi). When using a base station with a fixed position, the GPS can position the car with an accuracy of a few centimeter at a 10 Hz rate.
- **LiDARs**
  Two types of LiDARs were implemented for distance sensing. The first type was single point LiDARs (Garmin LIDAR Lite v3), one on each side of the car, pointing parallel to the ground, orthogonally to the forward direction of the car. These side-LiDARs have a range of 40 m, an accuracy of 2.5 cm, and a sample rate of 500 Hz.
  The second type of LiDAR was a 2D rotating LiDAR (Lightware SF40/C) measuring 360 degrees with a range of 100 m, an accuracy of 12 cm, and a sample rate of 4.5 Hz. One of these LiDARs was mounted parallel to the ground at the very front of the car, 30 cm above the ground.
- **Gyro**
  A single axis (used for orientation) gyroscope was implemented (Cruizcore XG1010). The sensor has a typical bias drift of less than 10 degrees per hour, with a 100 degrees per second resolution and an output rate of 100 Hz.
- **Wheel Encoder**
  Two inductive sensor mounts were mounted on the driven wheel (back right), along with a metal rotary encoder disk, enabling a distance measurement with a 1.44 cm resolution.
- **Web Cam**
  A simple PC webcam (Logitech C615) was also implemented for remote monitoring and image capturing. The purpose of this camera was not to perform computer vision.

The *previous platform*'s autonomous additions to the *basic system* are shown in Figure 3 in yellow. As shown, some of the autonomous sensors are taking advantage of the already existing CAN bus in the car, to communicate data to an added Autonomy Board (AUT Board). The AUT Board is also a custom PCB with an ARM based microcontroller, which has a direct serial connection to the PC over USB. The communication from the AUT Board to the PC has a 100 Hz frame rate.

All in all, the *previous platform* includes hardware enabling several possibilities for perception, localization and motion control, allowing the car to drive without the physical presence of a human driver.

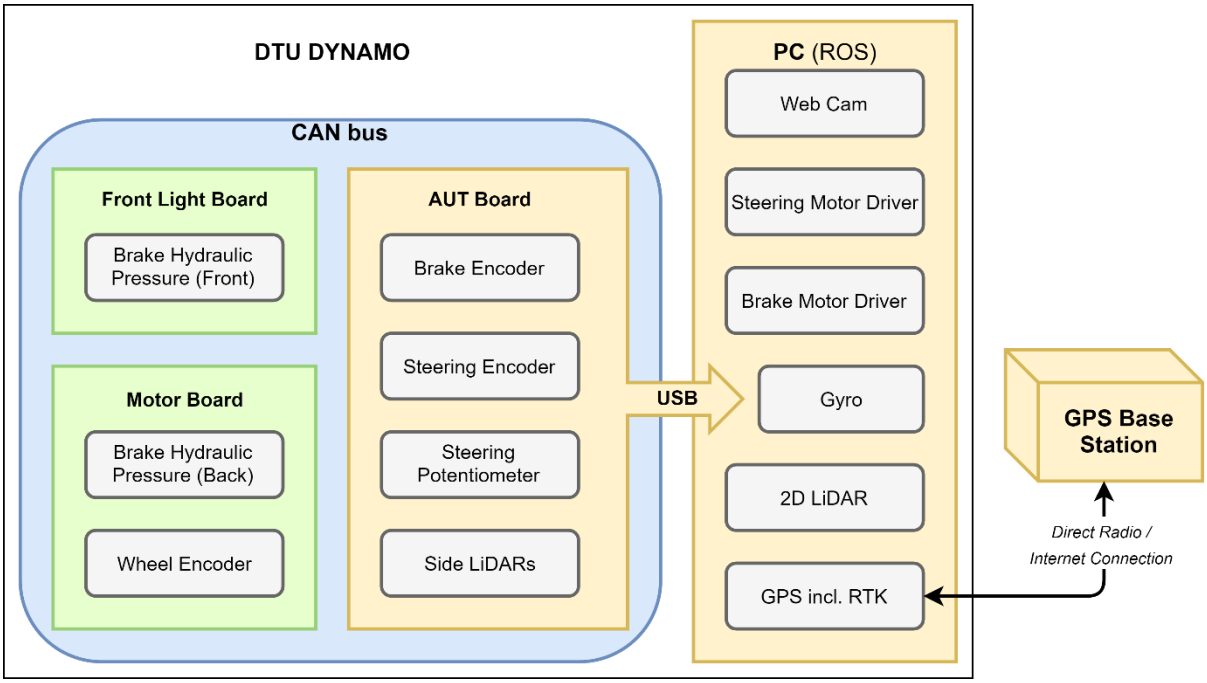


Figure 3 - Overview of communication in the previous platform. The overview is only showing the autonomy specific additions (Yellow) and the augmentations to the basic system (green), and therefore not showing the whole basic system.

2.1.2.2 Software

The software of the autonomous system in the car is implemented using the Robot Operation System (ROS). ROS is an Open Source, C++ framework, initially developed at Stanford University. The system is based on individual programs (*nodes*) working together in cooperation, coordinated by a ROS core program. The architecture of the system is highly modular, and includes many popular software libraries for robotics, such as "Open Source Computer Vision" (OpenCV) and "Point Cloud Library" (PCL). Communication between *nodes* happens using *messages* (data structures) that can contain various data types. *Messages* are linked to *topics*, which are labels to which *nodes* can subscribe or publish to. The embedded electronics in the *basic system* and the AUT board, are all programmed in C++.

The software in the *previous platform* contains software drivers for the hardware and ROS *nodes* allowing control of sensors and actuators in the augmented car. Finally, software for interfacing with the car remotely, through an internet connection, was also implemented in the *previous platform*, allowing both remote monitoring and control.

2.2 Project Analysis

The autonomous part of the DTU Roadrunners project is a large collaboration between several students, working individually on different aspects of the project. The project can be roughly decomposed into four main components:

- Sensing
- Perception
- Navigation and Planning
- Motion Control

The *previous platform* made a fully working foundation for sensing and motion control, but did not include implementations for intelligent autonomous driving. While all four components are vital to enable

autonomous driving of the DTU Dynamo, this thesis' main focus is on perception, and mainly work on situation awareness using LiDAR is performed. Camera requirements and implementations are established, but the image analysis is delegated to other students on the project. Necessary upgrades for sensing are also implemented, and preparations for navigation and planning are made.

2.2.1 Solution Requirements

First and foremost, the main priority for the capabilities of DTU Dynamo is to be able to follow the track and avoid obstacles. Following the track is essential for all five challenges, and requires detection of the barriers along the sides of the road. Detecting obstacles will be required in challenge 2 for the roundabout and chicane track layouts, in challenge 3 for the block at the end of the parking rectangle, and in challenge 4 for gate detection. Secondly, to fully enable completion of all five challenges, challenge 4 and 5 requires visual detection for road markings. Challenge 3 could use detection of road markings and obstacles for locating the parking rectangle.

Another important consideration is the expectations to the track itself, which is obviously not ideal. It cannot be expected that the road is completely smooth nor flat. The track's path in London contains slopes and changes in altitude, road bumps, and the track ground itself can be curved. Even though the barriers along the track are intended to facilitate path detection, they can contain gaps of various lengths. The barriers may even vary in shape along some track sections due to security features, such as the emergency exits for the cars.

An important aspect of the work throughout this thesis, is to enable further expansion to the autonomous solutions as the challenges increase in complexity in upcoming competitions. DTU Roadrunners aim to proceed to higher levels of challenges in future competitions, and this requires robust solutions, with future development in mind.

2.2.2 Previous Limitations

The *previous platform* does include sensors able to handle most of these technical challenges to a certain extent. The first limitation for the LiDAR setup is handling slopes or hills, since the 2D LiDAR has no way to differentiate driving towards an incline or an obstacle. The position of the 2D LiDAR is also very sensitive to the roughness of the road, because it is mounted on the very front of the car close to the ground (shown in Figure 4), in a very vulnerable position. Not only does the LiDAR's position depends highly on the suspension in the front wheels, but keeping the direction of the LiDAR's 2D scan parallel to the ground is impossible. For example, when driving on a plane, a single degree of tilt of the 2D LiDAR will cause the scan to detect ground as an obstacle approximately 17 m away. The low scan rate of the 2D LiDAR can also cause long detection delays, as further discussed in LiDAR Selection.

Another limitation of the *previous platform* is the camera implementation. Since several parts of the competition relies on computer vision for road marking detection, a dedicated camera solution with solid mounting and clear field of view is required.

Previously, all testing of the autonomous driving, was limited to roads and squares at DTU. Test tracks more similar to the AUC in London, would facilitate development of solutions.

Figure 4 - DTU Dynamo with the 2D LiDAR from the previous platform mounted on the front.

2.2.3 New Platform

LiDAR is a popular choice for local proximity perception in autonomous driving, as it can provide large amounts of long range measurements of high accuracy in 3-dimensions. The development in LiDAR technology is expected to have a major impact on the adoption of autonomous driving, as sensors are getting better and more affordable [5]. For the AUC it can be used to detect physical structures such as barriers and other obstacles. Other sensing technologies, often used in self-driving cars, are radar, sonar, and stereo vision, which have also been considered for perception purposes in the DTU Dynamo.

Radar can have a much longer detection range than LiDAR, but has much lower resolution when it comes to environmental imaging, because single measurements cover larger areas. Because of this, radar is mostly used in self-driving vehicles for detecting ranges to large obstacles and not precise scene interpretations.

Sonar sensors have a low range of detection, but can cover large areas, and are often comparatively cheap. In self-driving cars they can mostly be used at low speed maneuvering, for obstacle avoidance. Particularly, sonar sensors have been considered for parking assistance at the AUC.

Stereo vision is a popular technology for self-driving cars, as it provides regular camera images that allow for both computer vision and depth perception. The depth perception is of low accuracy, low resolution and is limited to shorter range intervals. Stereo cameras are also very dependent of light conditions and rely on textures or other visual features. Similar solutions, such as structured light or time of flight cameras, can also provide visual imaging that includes depth perception. Unfortunately, these technologies are still of low range, and sensitive to light conditions.

For the perception system of DTU Dynamo, LiDAR was chosen for obstacle detection. It covers large areas, provide precise and robust measurements in most common light and weather conditions, and even allows for precise scene interpretations. The large rise in use of LiDAR technology may cause better and cheaper future LiDAR solutions, and has already generated large software availability for LiDAR hardware, such as ROS libraries for the chosen new LiDAR for the DTU Dynamo. Since LiDAR will not meet all the requirements to complete all five challenges, a computer vision dedicated camera is also needed, because it can handle the visual markers, the LiDAR does not detect.

2.2.3.1 LiDAR Selection

The LiDAR implemented in the *previous platform*, the SF40 from Lightware, has several limitations as mentioned in section 2.2.2, that makes it difficult for DTU Dynamo to meet the necessary requirements for completing the AUC. To find a new LiDAR that can overcome these issues, many LiDARs were considered, but specifically two LiDARs were evaluated as substitutions to the SF40. As shown in Table 2, the MRS1000 from Sick and the VLP-16 from Velodyne, were considered and their advantages and disadvantages relevant to the AUC were compared.

Figure 5 - VLP-16 LiDAR from Velodyne. (Image from http://velodynelidar.com)

The VLP-16 (shown in Figure 5) was chosen, mainly due to its popularity that has generated high amounts of software availability and credibility due to the large amounts of field tests described by various sources. It is well tested for multiple purposes and confirmed to work well, even in light rain [6]. Furthermore, the VLP-16 has a higher amount of scan layers, the highest scan rate, and a higher detection range than the MRS1000. A detailed comparison of the technical specification of the MRS1000 and the VLP-16 are shown in Table 1, based on their datasheets [7] [8]. Apart from Velodyne's own visualization software (VeloView), a vast amount of dedicated software already exists. As previously mentioned, the VLP-16 is a common choice for autonomous vehicles and the Robot Operation System (ROS) used in the Ecocar, has its own maintained software package for the VLP-16[2].

	SICK MRS1000	VELODYNE VLP-16	LIGHTWARE SF40
Approx. price [DKK]	30000 (16000 incl. DTU discount)	53000	6400
IP Rating	67	67	00
Horizontal field of view [deg]	275	360	360
Horizontal angular resolution [deg]	0.25	0.1 - 0.4	0.22 - 0.18
Scan rate [Hz]	12.5	5 - 20	1 - 4.5
Horizontal layers	4	16	1
Vertical field of view [deg]	7.5	30	-
Vertical angular resolution [deg]	2.5	2	-
Min. detection distance [m]	0.2	1	~0.6
Min. measurements distance [m]	0.4	1	~0.6
Max. measurements distance [m]	64	100	100
Wavelength [nm]	850	903	905
Time delay (all layers) [ms]	80	1.33	*Unknown*
Power consumption (avg.) [W]	*Unknown*	8	4.5
Power consumption (max./peak) [W]	13	31	30
Evaluated echoes/returns	3	2	1
Systematic error/Accuracy @ max s[±mm]	60	30	30 - 120 @1 - 4.5 Hz
Weight [kg]	1.2	0.83	0.27

[2] http://wiki.ros.org/velodyne

Temperature range [°C]	-30 – 50	-10 – 60	0 – 40

Table 1 - Technical specification comparison of MRS1000, VLP-16 and SF40

Sick MRS1000	**Velodyne VLP-16**	**Lightwave SF40**
Advantages		
• **3D measurements** 4 channels provide more robust obstacle detection and enables track 3D mapping • **Cheapest 3D LiDAR** Approx. 16000 DKK (incl. DTU discount) • **Well known manufacturer** • **3 evaluated returns** Enables detection of semi-transparent obstacles, like fences or bushes • **Resistant enclosure** IP 67 rating • **Built-in hardware filters** • **High scan rate** Scan rate of 12.5 Hz	• **3D measurements** 16 channels provide more robust obstacle detection and enables track 3D mapping • **Well known manufacturer** • **Popular product and tested in the field** The common choice of LiDAR among current autonomous vehicle developers, and appears in many field tests • **Great software availability** Has lots of available software libraries, including one for ROS • **Long range** 100 m detection range • **2 evaluated returns** Enables detection of semi-transparent obstacles, like fences or bushes • **Resistant enclosure** IP 67 rating • **Highest scan rate** Scan rate of 5 – 20 Hz	• **Cheap 2D LiDAR** Approx. 6400 DDK • **Light weight** • **Long range** 100 m detection range
Disadvantages		
• **Limited availability field tests** • **Low software availability** Limited available software libraries for data handling. • **Limited range** The detection range is very dependent on light remission according to datasheet. It can go as low as 16 m at 10 % remission. The maximum detection range of 64 m would be sufficient, but requires 100 % remission.	• **The most expensive choice** Approx. 50000 DDK	• **Only 2D measurements** Fragile obstacle detection, especially in case of slopes on track • **Fragile enclosure** • **Requires exposed mounting** Must to mounted inconveniently in the front of the vehicle, exposed to collisions and vibrations • **Low scan rate** Scan rate of 4.5 Hz • **Very low accuracy at highest scan rate** • **Single return** • **Limited temperature operation range** 0 °C – 40 °C

Table 2 - LiDAR comparison for key features

To determine the required detection range of an obstacle or barrier by the VLP-16, the combination of braking distance, detection delay, and computational delay are inspected.

Considering the VLP-16 can run at a scanning frequency of 20 Hz, and requiring 3 consecutive possibilities for detection of an obstacle, would result in up to a 200 ms delay. A single detection would also be sufficient, but allowing up to 3 consecutive detections enables filtering over time, and leaves a higher safety margin. Adding a data transfer delay of 1.33 ms to the detection delay results in a 201.33 ms detection delay, corresponding to approx. 1.4 m at 25 km/h. Running the LiDAR at a scanning frequency of 10 Hz will provide better horizontal resolution, and if chosen, the equivalent detection delay, will be 401.33 ms, corresponding to approx. 2.8 m at 25 km/h.

To estimate a computational delay, the desire of interpreting the LiDAR measurements in real-time is set as a requirement. This demands a computation time corresponding to at least a full LiDAR scan at the 20 Hz frequency. This delay corresponds to approx. 0.35 m at 25 km/h.

Combining the maximum braking distance, detection delay, and computational delay, results in an approximate minimal desired detection range of 23.2 m. This is well within the measurement range of 100 m of the VLP-16, even when including a large safety margin.

2.2.3.2 Camera Selection

As mentioned in 2.2.1 - Solution Requirements, completion of all five challenges requires visual detection for road markings. The road markings are as of the writing of this thesis, still not fully defined by Shell. It can however be expected that weather conditions will be light and dry, as in the event of rain, the rules state that the challenge will be postponed. With good lighting conditions, the possible camera shutter time can be expected to be low, and combined with the maximum driving speed of *25 km/h*, motion blur is expected to be minimal, especially since the camera and driving direction are almost colinear. In these conditions, the main concern for blurred images, are caused by vibrations of the car are. Another advantage of the low speed limit, is a low required frame rate of the camera, as we can expect the road marking to be clearly visible from afar, and we can expect to car to slow down the closer it gets to the parking rectangle.

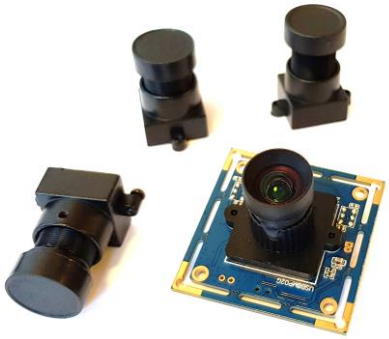


Figure 6 - Camera: USB8MP02G-L75 from EL, and M12 lenses

Four cameras were considered for selection, shown in Table 3. These cameras were mainly considered due to their low cost, common interchangeable lens format, and small form factor. These priorities are largely based on the fact, that the requirements for the camera are still not definitely defined, but expected to be low compared to more expensive industrial cameras. The Raspberry Pi camera was considered because of the large software availability provided by the huge Raspberry Pi community. This solution was not chosen because it requires an additional computer system in the car (a Raspberry Pi), instead of connecting directly to the main computer.

The oCAM-1CGN-U (referred to as oCam in the following) was considered because it is a very cheap global shutter camera, and because the manufacturer provides software examples for computer vision. The global shutter sensor will provide more stable images when moving. Both the camera from ELP and E-CON SYSTEMS are low cost cameras compared to industrial standard, and have high resolution.

The camera from ELP is not from a well-known manufacturer and has limited documentation. Despite this, the camera from ELP (shown in Figure 6) was chosen as an initial choice, because of the low cost and high resolution. In case this choice is too sensitive to vibrations, the oCam will be left as the second choice. If the resolution of the oCam is too low to see both near and far markings, two oCams with different focal lengths could solve this at a relatively low cost. The lacking commitment to a permanent choice, is a result of the lack of definite challenge requirements at AUC.

BRAND	ODROID	ELP	E-CON SYSTEMS	RASPBERRY PI
Model	oCam-1CGN-U	USB8MP02G-L75	BoardSee3CAM_CU135	Camera Module v2
Approx. price [DKK]	1148	426	1653	259
Shutter type	Global Shutter	Rolling Shutter	Rolling Shutter	Rolling Shutter
Sensor	OnSemi AR0134	SONY IMX179	OnSemi AR1335	Sony IMX219
FOV [degrees]	65	75	67	62.2
Standard Lens Focal Length [mm]	3.6	3.2	3.2	3.04
Lens mount	S M12	S M12	S M12	S M12 (requires adapter)
MP	1.2	8	13	8
Pixel Size [sq. µm]	14.1	1.96	1.21	1.26
Sensor Size [sq. mm]	17.8	16.7	13.1	10.2
Temperature [celcsius]	0 to 70	-10 to 70	-30 to 70	-20 to 60
Case/protection	Yes	No	No	No
Communication type	USB 3.0	USB 2.0	USB 3.0 Type C	Raspberry Pi dedicated
Compression	MJPEG / YUV	MJPEG / YUV	MJPEG / UYVY	MJPEG / YUV
UVC (USB Video Class)	Yes	Yes	Yes	No
Computer vision project examples available	Yes	No	No	Yes

Table 3 - Technical specification comparison of considered cameras

2.2.3.3 New Platform Overview

Replacing the LiDAR system and camera in the *previous platform* with the new hardware, a *new platform* is derived. An overview of the communication in the *new platform* is shown in Figure 7.

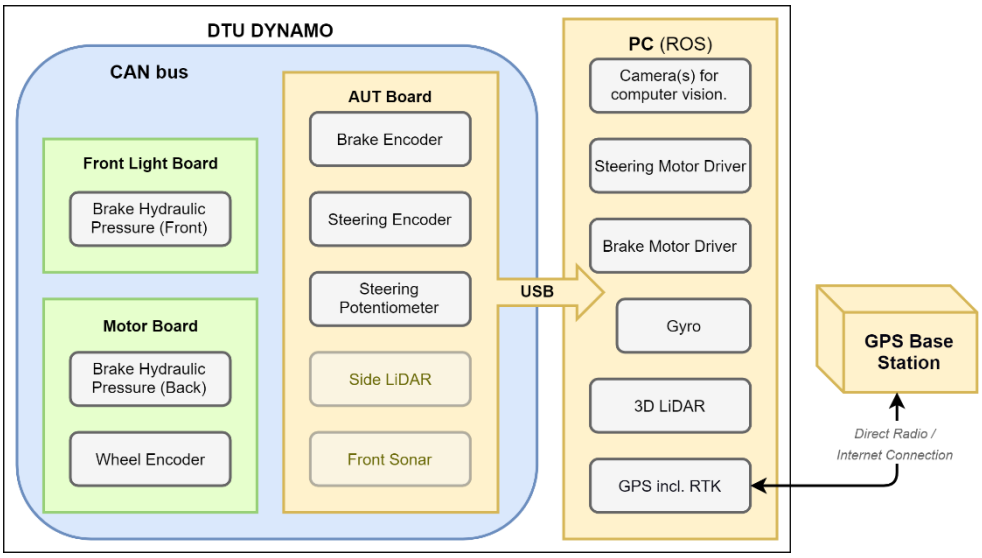


Figure 7 - Overview of communication in the new platform. The overview is only showing the autonomy specific additions (yellow) and the augmentations to the basic system (green), and therefore not showing the whole basic system.

The side LiDARs are no longer essential to the platform, as the information provided by the 3D LiDAR covers the equivalent information. They are however kept for data test purposes and redundancy. Sonar might also be added to the platform for close range obstacle ranging, specifically for the Maneuverability challenge.

To mount the VLP-16 and selected camera, a mount for the top of the DTU Dynamo was designed and produced by 3D printing in collaboration with the mechanical student on the team, shown in Figure 8.

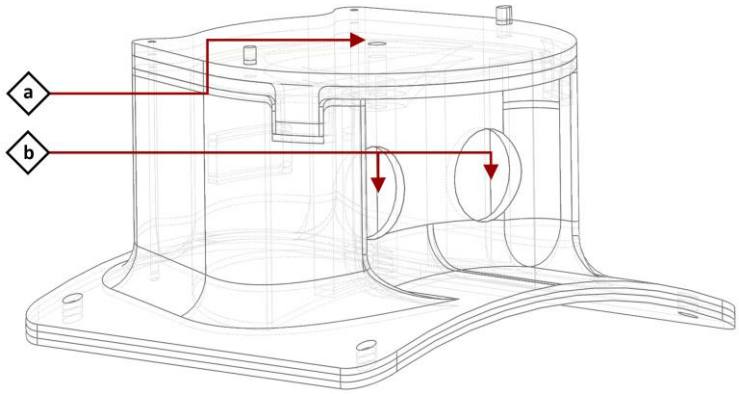


Figure 8 - Sketch of mount for 3D LiDAR scanner and cameras(s). The first arrow (a) point to the mounting place for the VLP-16, with adjustable tilt. The second arrow (b) points to the holes for camera lenses in the camera chamber.

The mount has adjustable tilt for the LiDAR, and is mounted to the car with vibration and shock absorbent material between the mount and car chassis. The mount is also designed for two cameras, in case a dual camera setup is needed for two different field of views, as mentioned in 2.2.3.2 - Camera Selection.

With this positioning of the LiDAR, the detection range of a barrier, can be calculated. If the LiDAR is positioned horizontally, as depicted in Figure 9, the detection range of a horizontal layer (distributed as in Figure 10) of the VLP-16 can be calculated. l_h is the height of the LiDAR position. l_d is the horizontal distance from the LiDAR to the front of the vehicle. l_a is the horizontal angle of the LiDAR scanning layer. b_h is the height of a barrier as defined by the AUC rules. bdr_{bottom} is the barrier detection range (BDR) to the bottom of a barrier. bdr_{top} is the BDR distance to the top of a barrier.

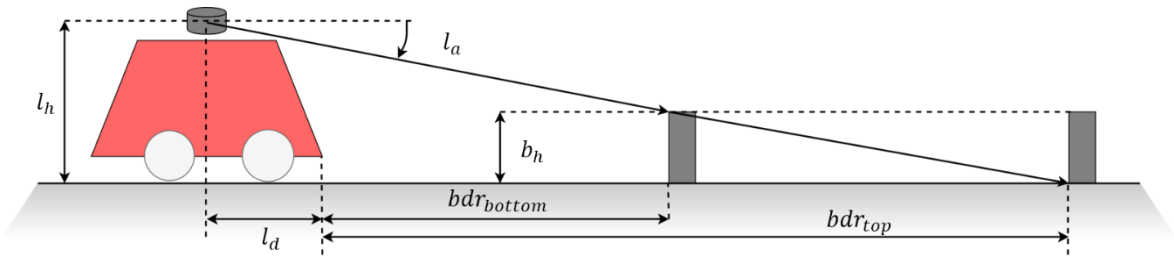


Figure 9 - Geometric sketch of the LiDAR positioning in DTU Dynamo.

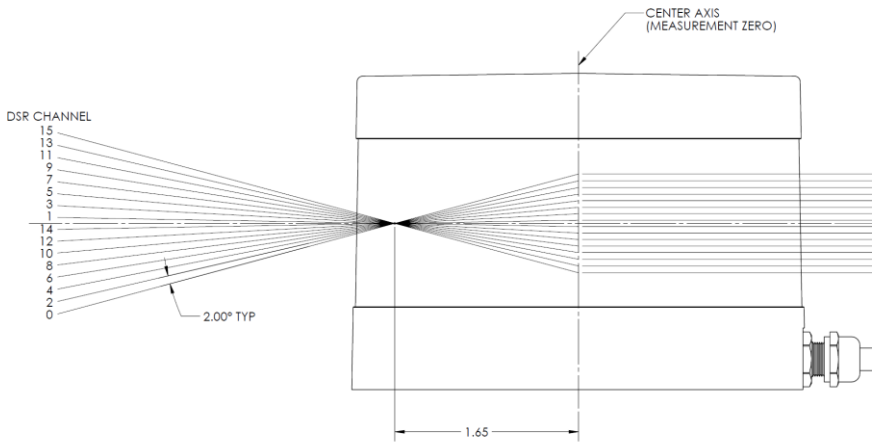


Figure 10 – Drawing of the beam distribution from the Velodyne VLP-16 datasheet

As l_h is *1.1 m*, l_d is *1.4 m*, b_h is *0.5 m* and the angle between each horizontal layer is 2 degrees, the expected detection range intervals for barriers are calculated for each layer in Table 4. It is notable that the detection ranges go well beyond the calculated required detection range of *23.2 m*. It is however noteworthy that the layers create a blind gap for barrier detection, between *19.6* m and *33.0 m*. This gap should not heavily influence barrier detection, since the LiDAR will detect the stationary barrier before it enters this gap. Gaps between the LiDAR layers will always be present, and this emphasizes the importance of obstacle tracking. Other obstacles such as the gates, should ideally be detectable up to the 61.6 m of channel 14. Finally, it should be noted, that the bdr_{bottom} is an ideal detection range, as the measured point will need to detect some higher altitude on the obstacle to differentiate from the surrounding ground. In general, the calculated detection ranges of Table 4 are only used for roughly estimating the detection ranges, as the LiDAR will be mounted on a vehicle driving on a rough and curved plane.

In case it becomes eminent to detect barriers directly in this gap which surrounds the calculated detection range, adjusting the mount to add a single degree tilt to the LiDAR, would change the BDRs of the horizontal scan layers 12 and 14 to *7.18 - 14.3 m* and *15.8 - 30.1 m*.

Horizontal scan layer [channel]	bdr_{top} *[m]*	bdr_{bottom} *[m]*	l_a *[degrees]*
0	0.83	2.70	15.0
2	1.20	3. 36	13.0
4	1.67	4.26	11.0
6	2.39	5.55	9.0
8	3.49	7.56	7.0
10	5.45	11.2	5.0
12	10.0	19.6	3.0
14	33.0	61.6	1.0

Table 4 - Overview of barrier detection ranges based on the LiDAR mounting position on the DTU Dynamo

2.2.4 Barriers and Test Track

As mentioned in Previous Limitations, all testing of the autonomous driving, was previously limited to roads and squares at DTU. To enable testing autonomous driving in conditions similar to those expected at AUC, a test track has been established. A total of 84 m of racetrack barriers with the same exact dimensions and colors as the barriers used at the AUC (shown in Figure 11), have been purchased for this purpose. The test track was set up on grounds already used for autonomous vehicle testing by a company called Autonomous Mobility, who allowed us to use the area.

Figure 11 - Image of test track with racetrack barriers similar to barriers used at AUC

The grounds of the track is relatively flat and smooth, which enables both long and curved track sections to be build. The sides are covered by tall grass fields and a few trees. A wide-angle image of the area is shown in Figure 12.

Figure 12 - Wide angle image of the track grounds

2.2.5 Software

The software used in the *previous platform* is still used. Additionally, ROS libraries for OpenCV and PCL are added. Furthermore, Gazebo, a robot simulation tool, has also been used to generate simulated track data. The dedicated ROS packages from Velodyne for the VLP-16, which includes the hardware drivers for its ethernet data connection, have been added to the ROS package libraries on the DTU Dynamo. RViz, a built-in 3D visualization tool for ROS, was used to generate 3D visualizations, such as the ones shown in this thesis.

2.3 Project Overview

The perception is split into several steps, shown in Figure 13, and are explained further in subsequent chapters. The diagram is a simplified overview, of the steps implemented to achieve the desired LiDAR perception, essential for the self-awareness of DTU Dynamo. Designed to meet requirements established in Solution Requirements, the processing begins with raw sensor data, achieving a simplified but sufficient interpretation of the car's environment.

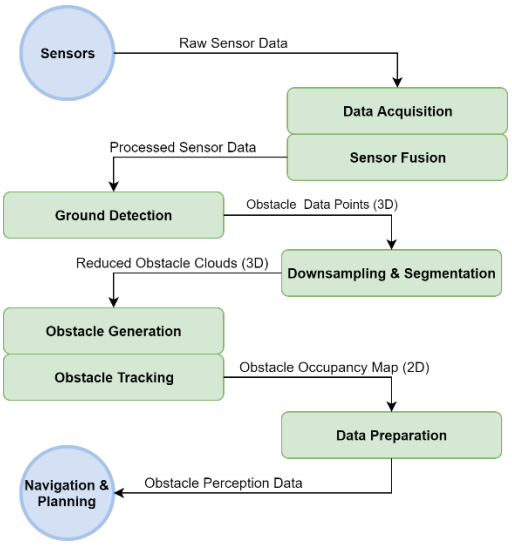


Figure 13 - Simplified functional overview of LiDAR perception implementation in steps.

The steps in the diagram of Figure 13, can be roughly described:

- **Data Acquisition**
  Retrieving and converting the data from the LiDAR and odometry.
- **Sensor Fusion**
  Combining odometry with LiDAR measurements.
- **Ground Detection**
  Separation of ground and obstacles.
- **Downsampling and Segmentation**
  Reducing and splitting point clouds, representing individual obstacles.
- **Obstacle Generation**
  Reducing point clouds to simplified obstacle interpretations.
- **Obstacle Tracking**
  Tracking obstacles between LiDAR scans.
- **Data Preparation** (Part of System Implementation Overview)
  Preparing results for Navigation and Planning

3 LiDAR Data Acquisition

With the chosen Velodyne VLP-16 mounted and connected to the DTU Dynamo, data scans can be obtained through an ethernet connection. The raw data points (single LiDAR measurements) from the VLP-16 are compressed into a sequential data form, containing spherical coordinates, wrapped in small data packets of 24 vertical scans (384 points). A ROS code implementation for DTU Dynamo was used to decompress and convert the data to cartesian XYZ-coordinates in the LiDAR's local frame.

The XYZ coordinate system used for the car was defined so the positive direction of the x-axis points in the forward direction of the LiDAR. The positive direction of the y-axis points orthogonally to the x-axis, horizontally left. The z-axis' positive direction points upwards, following the *right-hand rule*.

To achieve the conversion to this XYZ coordinate system from the spherical coordinates, equation (1) is used, where r is the radial distance, α is the azimuth angle (or orientation), and ω is the polar angle (inclination angle).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = r \begin{bmatrix} \cos(\omega) \cdot \cos(\alpha) \\ -\cos(\omega) \cdot \sin(\alpha) \\ \sin(\omega) \end{bmatrix} \tag{1}$$

The VLP-16 measures the intensity of each returning laser pulse reflection emitted by 16 spinning laser projectors inside its LiDAR system. Apart from measuring the intensity of each laser pulse reflection, it also provides both/either the distance to the strongest and/or farthest return. This enables the VLP-16 to generate approx. 576000 points per second (288000 points per second for single return mode).

3.1 Position Filtering

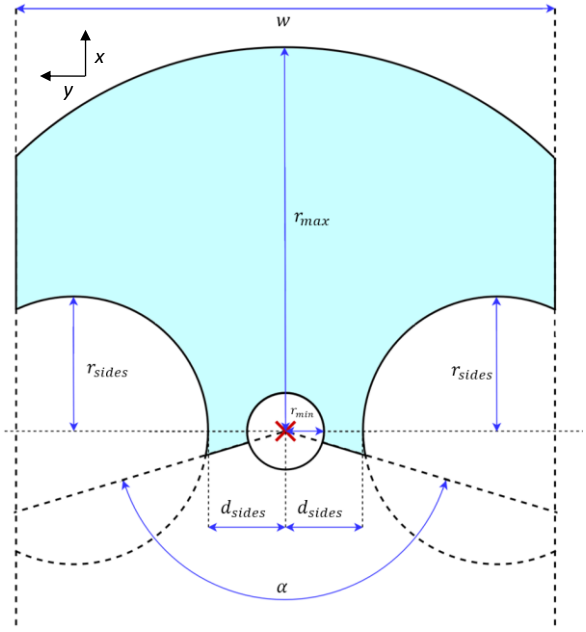


Figure 14 - Depiction of position filtering of points. Red cross indicates the LiDAR position, and the blue section indicates the area of remaining points.

The high amount of data provided by the VLP-16 can be filtered to lower the computational work required for data processing. Apart from methods such as down-sampling, discussed further in chapter 6 -

Downsampling & Segmentation, many points can be removed, solely based on their position. An adjustable filter for such removal has been implemented and is depicted in Figure 14.

Points too far from the car to be of importance for the autonomous driving, can be removed by setting a maximal distance to the car, r_{max}. Points within a distance of r_{min} can also be removed. Since DTU Dynamo cannot drive backwards, points within an angle interval of backwards orientation, defined by the angle α, can be removed as well. Due to the limited turning radius of the car, points farther away from the x axis, requires a longer driven distance to reach, and can be approximately removed by a maximal width, w, of a scan. To remove additional points in areas considered unreachable or unimportant, two circular sections, defined by their distance to the car, d_{sides}, and radius, r_{sides}, can be set. Not all the parameters for the position filter is necessarily used. Finally, a part of the position filter, not depicted in Figure 14, is an interval of z- coordinates, removing irrelevant points above the car and points far below ground due to reflection errors.

3.2 Tests & Results

Based on measurements performed on the test track, import and conversion to XYZ coordinates are on average performed in 3.94 ms (standard deviation of 0.25 ms) of computation time when the position filter is not applied. When filtering is included, the process is performed on average in 1.62 ms (standard deviation of 0.27 ms). All the execution times are only for relative comparison, and are performed on a CPU (Intel Core i5-5200U) with significantly slower single core performance than the CPU used in DTU Dynamo.

In Figure 15, an example of 3D measurements is shown. In Figure 16, images with a closer look at the LiDAR position is shown, before and after the position filter is applied. Tests were both performed on the parking lot shown in Figure 16, and on the test track, with the following parameters, also used in later tests:

$$r_{max} = 40\ m, \qquad r_{min} = 1\ m, \qquad \alpha = 160°, \qquad w = 30\ m, \qquad d_{sides} = 6\ m, \qquad r_{sides} = 5\ m$$

These parameters generally reduce the amount of points by 70-80 %, although it depends on the environment. As confirmed by the tests, applying the filter also reduces the computation time in the tested environment.

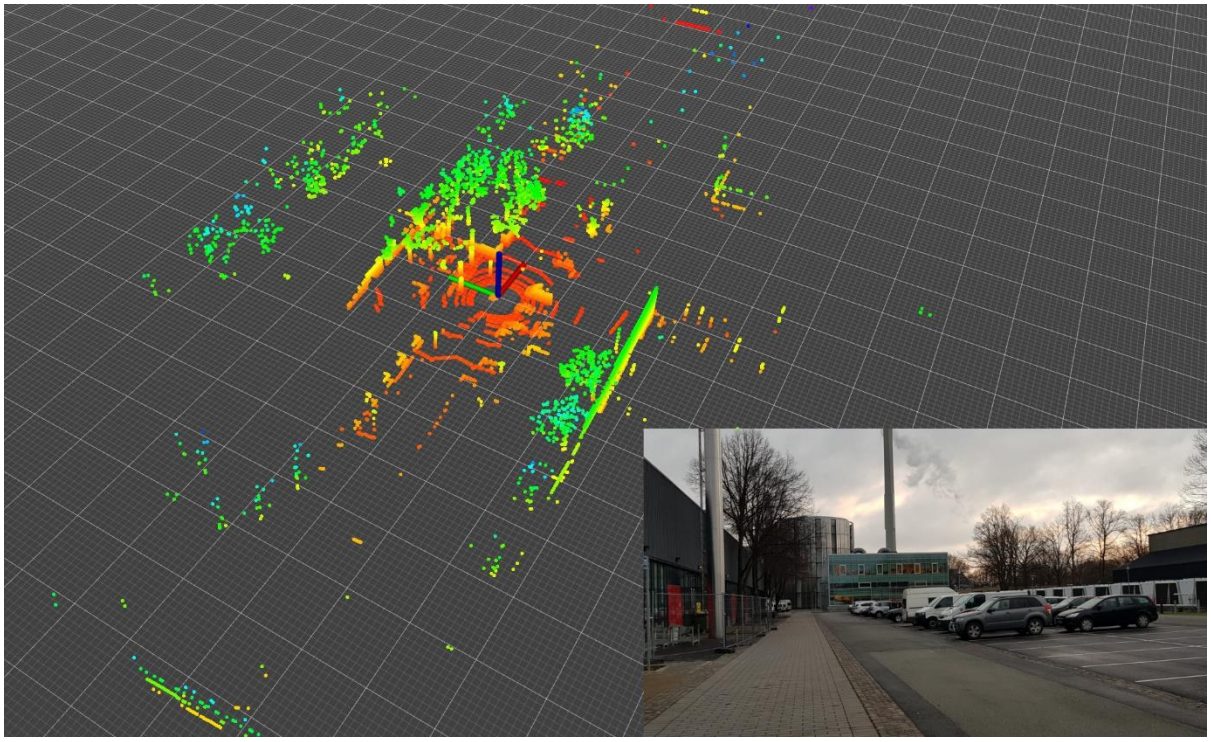


Figure 15 - Image of 3D LiDAR measurements on the parking lot in a 3D coordinate system. The color of each point corresponds to the altitude of the point. The grid consists of large squares of 10 m by 10 m, and smaller 1 m by 1 m squares. The x, y, and z axis are shown by corresponding green, red, green and blue markers. The image in the right lower corner depicts the location of the measurements.

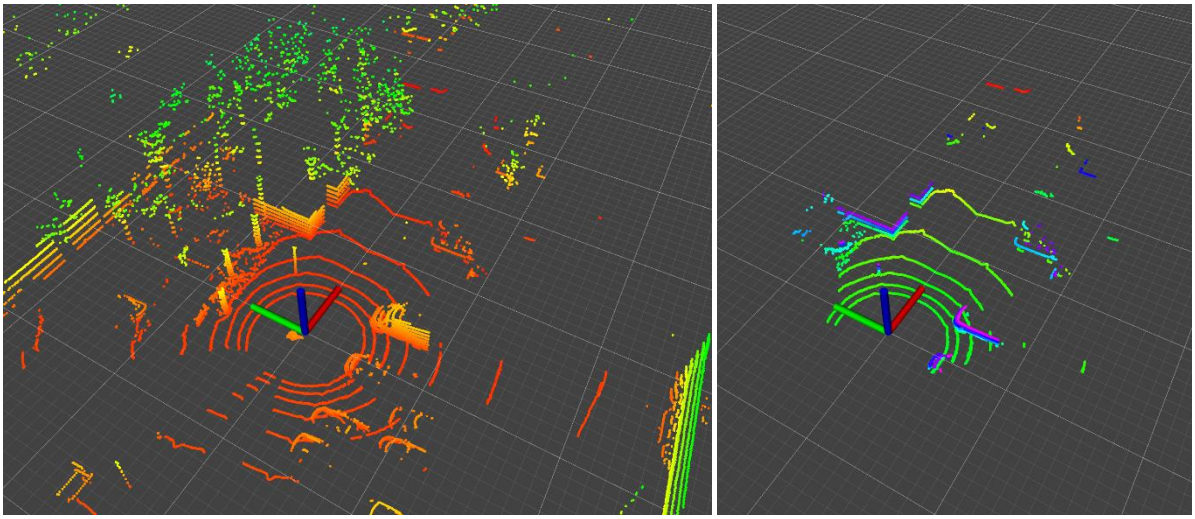


Figure 16 – Images of 3D LiDAR measurements on the parking lot in a 3D coordinate system. The left image shows the points before position filtering and the right image shows the points after position filtering. The color of each point corresponds to the altitude of the point. The grid consists of large squares of 10 m by 10 m, and smaller 1 m by 1 m squares. The x, y, and z axis are shown by corresponding green, red, green and blue markers.

3.3 Partial Conclusion

It is clear from several tests, that the data from the VLP-16 contains very few random false positive obstacle detections, which can easily be removed as they appear momentarily and isolated. Furthermore, the hardware lives up to its specifications, measuring distances longer than the *100 m* specified in the datasheet, and achieving the specified accuracy of the LiDAR measurements. Even measurements performed in light rain were barely affected, only losing few measurement points, and the sensor even performed well in snow.

4 Sensor Fusion

Combining LiDAR data with odometry can achieve enhanced data. Firstly, the odometry of the car can correct for the error in LiDAR measurements caused by displacement of the LiDAR scanner during scans. Another application needed is transforming data from the LiDAR's local frame to a world frame. By performing this transformation, data from different scans can easily be combined and compared, to achieve a better understanding of the car's environment.

4.1 Odometry

To attain a sufficient odometry, lateral movement measured by the wheel encoder and rotational movement measured by the gyroscope are combined. The position of the steering wheel could also be added, but is mainly used for motor control of the steering motor. GPS is not used because of its sensitivity to the shape of the surrounding landscape and because we are mainly interested in relative positioning for LiDAR perception. GPS can however be used for later odometry corrections for mapping, providing absolute position estimations.

A moving average is applied to the encoder measurements to diminish the stepping behavior of the encoder, and combined with the gyroscope, odometry is calculated. The resulting odometry estimates are saved in a ring buffer along with timestamps.

4.2 Correction

The time for the VLP-16 to perform an entire single rotation scan is 50 ms at the maximal scan rate of 20 Hz. This allows DTU Dynamo to move approximately 0.35 m at 25 km/h between scans. If the scan rate is set to 10 Hz (in case a higher scan resolution is desired), this distance will double. Rotation/turning of the car also causes displacement of LiDAR measurements. At 25 km/h, a turn radius of 6 m results in 3.31 degrees turn between scans, at a 20 Hz scan rate. At 10 m distance, this can cause 0.58 m displacement of a detected object. Again, this displacement will also double at the 10 Hz scan rate. These displacements not only cause errors for positioning detected obstacles, but can cause difficulty for obstacle tracking.

To minimize these displacements, the ring buffer containing odometry estimates with corresponding timestamps, is used to perform linear interpolation over time. Each LiDAR data packet (which includes 24 vertical scans) is corrected using the interpolated odometry. This could potentially be improved further. Performing the interpolation for each vertical scan would be more precise, but since a packet only spans over 1.33 ms of scan time, it is evaluated to not be worth the extra computation. Furthermore, more complicated interpolation could be used to achieve better modelling, but is again disregarded because the effect is deemed minimal.

4.3 Tests & Results

To evaluate the accuracy of the odometry, tests drives on the test track were performed. The odometry can be compared to the measurements of the very accurate GPS with RTK. Two tests with a left and a right turn were performed. The conditions of the track were very poor, due to snow on the road. This made the track more uneven and increases the change of wheel slips. Hence the following test result can be interpreted as a worst-case scenario. In Figure 17, Figure 18, and Figure 19, the results of the two test drives are shown.

The position error of the odometry clearly accumulates, when compared with the GPS measurements. But inspecting the driven distance and orientation, the difference is quite small. It is clearly not sufficient for

mapping, as the Euclidian position error is 2.0 m (left turn) and 2.9 m (right turn), equivalent to 5.3% and 5.2% of the driven distance. With the scan frequency of the LiDAR being 10 – 20 Hz, these accumulated errors are satisfactory for the data correction and later obstacle tracking.

Another noticeable difference between the two types of position estimates, is the instability of the GPS at close to stationary positions, for estimating orientation.

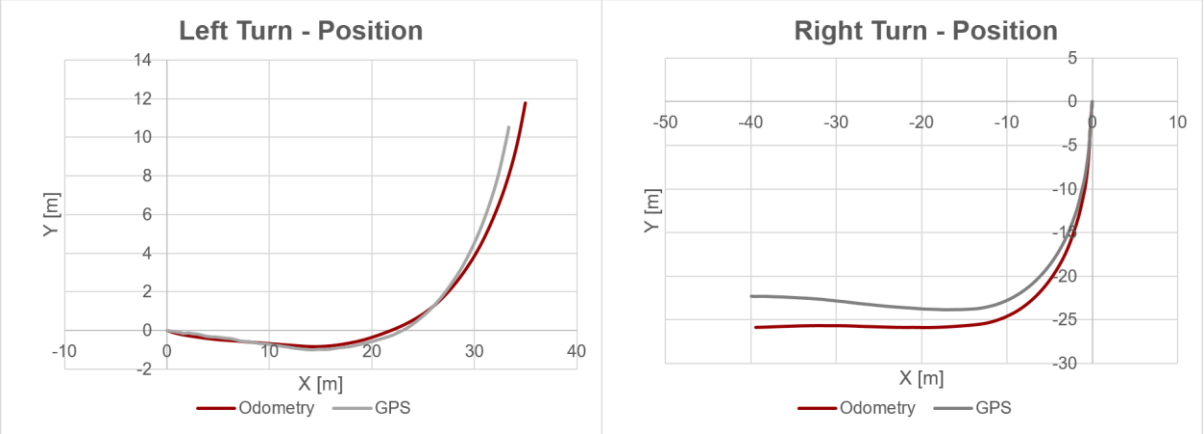


Figure 17 - Position measurements based on odometry and GPS from test drives on test track.

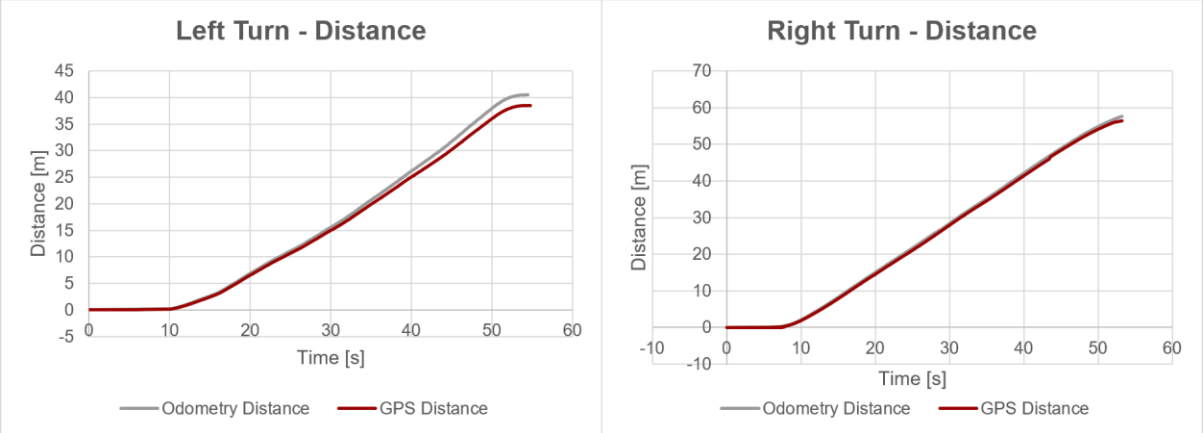


Figure 18 - Driven distance measurements based on odometry and GPS from test drives on test track.

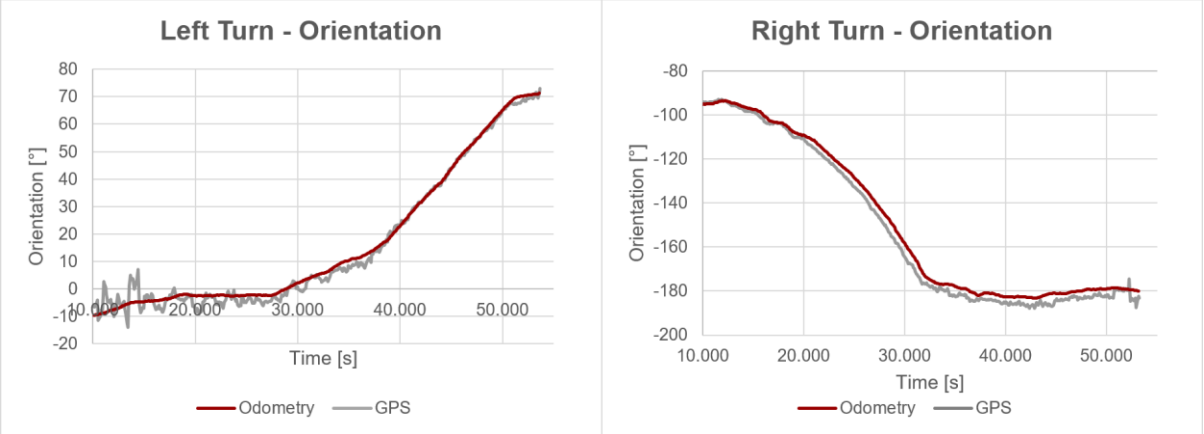


Figure 19 - Orientation measurements based on odometry and GPS from test drives on test track.

With a sufficient odometry estimate, correction of the LiDAR scan points can to be implemented. To inspect the correction, the overlap of points in each scan, shown in Figure 20, can be used. The overlap is provided by the VLP-16 by default and means that each scan contains up to a whole data packet (24 vertical scan lines) in addition to a whole rotation. After applying the correction, improvements like the two examples in Figure 21 appeared consistently. Both scenarios are examples of displacements, and where captured driving with a turn rate of approximately 10 degrees per second, at approximately 6.5 km/h, with a 10 Hz scan rate, from 41.2 m (top) and 39.7 m (bottom) distance. To quantify the improvements, features such as corners are acquired, along with consistent odometry measurements, which also vary in precision. These requirements make it very difficult to perform tests with large amounts of essential observations, necessary to derive a statistical average of the performance. An estimated reduction of the Euclidian distance between corresponding features, based on tests similar to those in Figure 21, gives an estimate of reductions of 70 – 90 %.

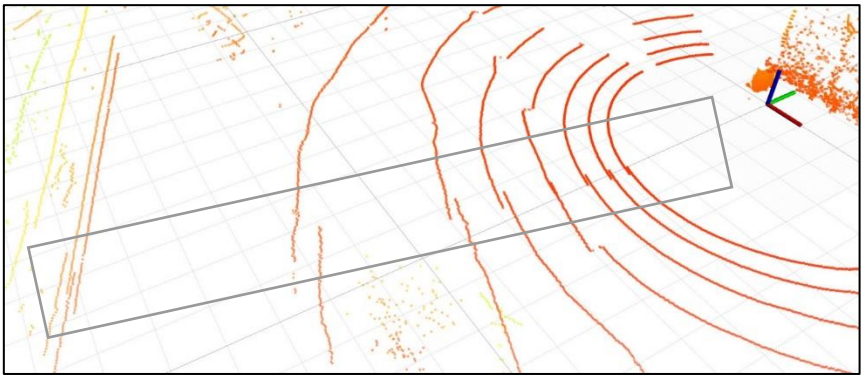
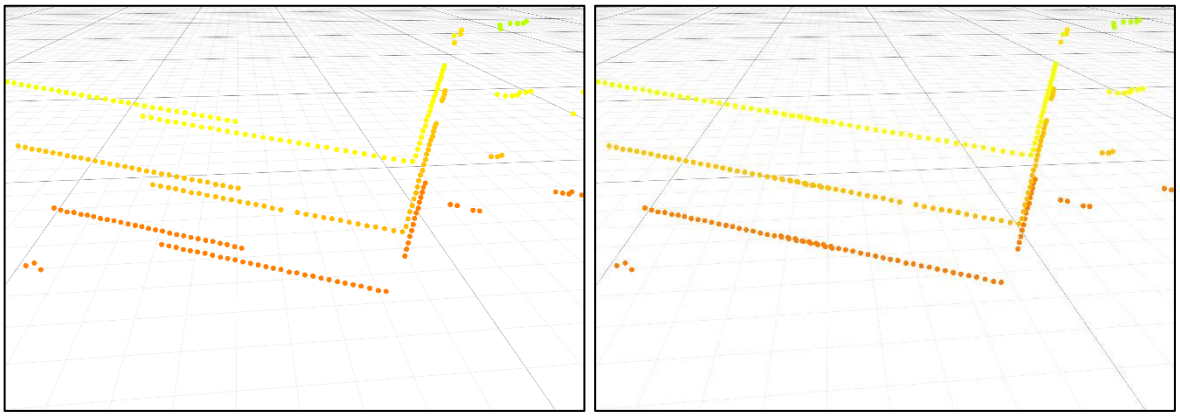


Figure 20 - 3D plot of LiDAR point measurements from a full scan, including a visible overlap, marked by a gray rectangle.



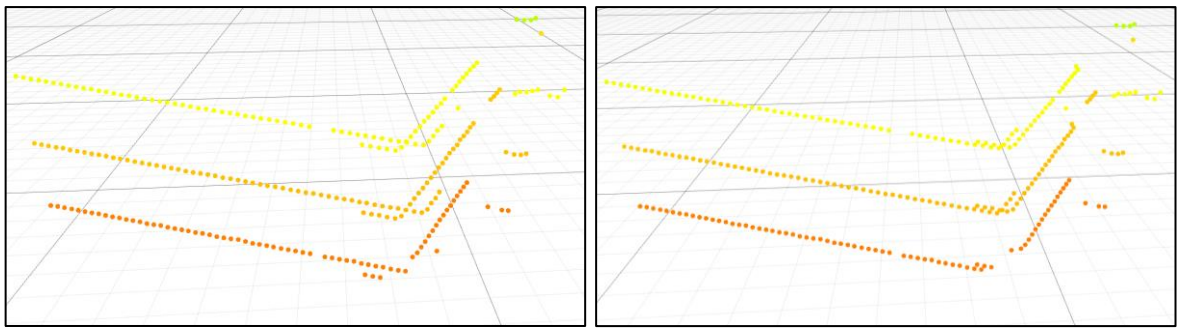


Figure 21 - 3D plots of LiDAR measurements before (left) and after (right) odometry correction. Small grid size of 1 m by 1 m, and large grid size of 10 m by 10 m, only shown on the z plane.

4.4 Partial Conclusion

The odometry achieved using only the wheel encoder and gyroscope appears to perform sufficiently to achieve local estimates of the cars position between scans, even when performed in suboptimal conditions. The VLP-16's fast scan rate and DTU Dynamo's low movement speed at the AUC, lowers the expected errors, and the odometry can both be used for correction of LiDAR measurements, and provide sufficient information for obstacle tracking. Improvements of LiDAR data by fusing it with odometry were achieved through testing, although it is difficult to quantify the effect.

5 Ground Detection

An essential requirement to detect obstacles based on the LiDAR data, is to separate obstacles from the ground. By removing the drivable ground from the LiDAR generated point cloud, the remaining points can be interpreted as obstacle detections.

Two common ways to perform ground segmentation to divide and classify point clouds is by either model-fitting or graph structures.

5.1 Model-fitting

Model-fitting is a fast way to group points by simple descriptors such as lines, planes and even more complex shapes like cylinders or curves. For environments with plane ground, i.e. inside structures, this is the effective solution. Two of many possible choices for this purpose, are the RANSAC algorithm (RANdom SAmple Consensus) and the Hough Transform algorithm.

Traffic roads are mostly uneven, curved, and sometimes even rough and irregular, especially in urban environments. This causes problems when attempting to perform ground plane detection. A RANSAC algorithm has been implemented, both to test the effectiveness in our test environments, but also because it can be utilized as a tool for additional ground detection (after graph structure), or for obstacle type classifiers.

The RANSAC algorithm is an iterative stochastic process for finding a 'best' model fit for given data [9]. Every iteration, the algorithm fits a model to random samples of a dataset. It then finds all data observations which are sufficiently close to the model, also known as inliers, and evaluates the model. When a certain number of iterations has been executed, the model which achieved the 'best' evaluation is used.

This process makes the RANSAC algorithm very resistant to outliers, and therefore great for finding a set of observation which fit a certain model of an object (like a plane), within a larger dataset (like a point cloud). RANSAC relies on randomly picking samples which are part of the modeled object. This means the chance of finding the correct model, depends on the fraction of observations which are part of the searched for object, and the number of iterations performed. Since the ground should represent a large fraction of the LiDAR data, RANSAC can be a great choice for ground detection.

5.1.1 RANSAC Implementation

Although PCL does contain a RACNSAC algorithm variation, it is very computationally heavy. This is because it refits a second model using all inliers from the respective iteration, every iteration. The purpose of the second model is to calculate a more accurate error estimate for every evaluation. The PCL algorithm is explained in detail on their website[3].This is not essential for our use, because the planes of interest are assumed to be somewhat uneven, and do therefore not need a high precision model fit.

The variation of the RANSAC algorithm implemented (referred to as *custom RANSAC*), relies on the number of inliers of each fitted plane, to evaluate how well the plane fits. This is similar to the RANSAC algorithm described in [9]. Finally, the *custom RANSAC* performs a final model fit for the 'best' plane with most inliers. The *custom RANSAC* allows for filtering by plane parameters, to avoid random sampled planes which are far from horizontal or vertical, to increasing the computational efficiency when searching for ground planes

[3] http://pointclouds.org/documentation/tutorials/random_sample_consensus.php

or walls. Additionally, the *custom RANSAC* uses downsampling (elaborated in chapter 6 - Downsampling & Segmentation) to further increase computational efficiency.

In the end, it can be expected that using RANSAC for ground detection will lead to errors when the ground is not sufficiently flat, or when the stochastic properties of RANSAC, fails to generate a suitable fit.

5.2 Graph Structure

For LiDAR scans, graph structures rely on the relationship between neighboring data points, often referred to as nodes. Each node is connected to neighboring nodes (referred to as neighbors), specified by some criteria. This allows for local evaluations, such as distances, gradients, curvature, etc. An example of the connections in a graph structure is visualized in Figure 22.

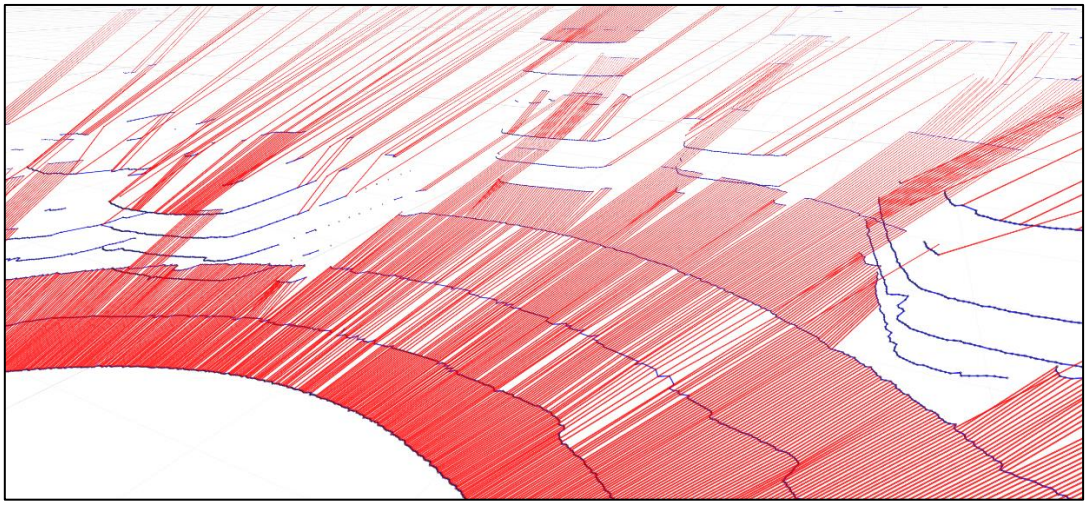


Figure 22 - Visualized graph structure. Red lines represent horizontal node connections. Blue lines represent vertical node connections. Black dots represent nodes.

One type of graph structure analysis, is an image based method, where all points are projected onto a 2D plane. This allows the use of common image analysis methods, where calculations using neighboring pixels are essential. It is expected that the VLP-16 setup, will not provide enough vertical resolution to achieve good results using such image-based methods, because it discards positional information. By keeping the data in a graph structure, relational calculations between neighbors can be performed for each node, and node connections can be created or cut between all points. This provides a great resource for both classification and segmentation, and can be implemented into the LiDAR data import, as data is delivered in a sequential manner.

Such graph-based methods show promising results in [10] and [11], specifically for uneven ground detection with focus on driving. Both [10] and [11] investigate the performances of graph based ground detection. [10] introduces an effective method for smooth environments, thought it appears to be quite computationally heavy for real-time use. [11] has further developed a more tolerant method for rough areas, with a faster computational executions time.

The desired solution would be a method that can handle the low vertical resolution (64 layers used in both [10] and [11]), with relatable parameters for easy track adjustment. The proposed method is very similar to [11] but is designed with such specifications in mind, to successfully detect traversable road from obstacles.
Examples of relatable parameters can be found from a more mechanical perspective, since DTU Dynamo

has driving-limitations. The car has a maximal change in slope or curvature of the road it can handle, because of the low-level chassis. The power of the motor and brakes also limit the maximal slope the car can handle. Lastly, small but immediate alterations on the road, due to roughness or small bumps and pits, should be tolerated. Some examples of such potential appearances on the SEM track are steel road plates, thick cables or road transitions.

5.2.1 Implementation

The proposed implementation for graph-based ground detection will be referred to as PIGD.
To enabling a graph structure implementation, node objects are created. Each node can have up to 4 neighbor references:

- An upper neighbor, referring to the LiDAR scan point one scan layer above the current node, in the same vertical scan line.
- A lower neighbor, referring to the LiDAR scan point one scan layer below the current node, in the same vertical scan line.
- A right neighbor, referring to the LiDAR scan point in next vertical scan line, in the same scan layer/horizontal line.
- A left neighbor, referring to the LiDAR scan point in previous vertical scan line, in the same scan layer/horizontal line.

Additionally, each node contains:

- A 3D position, both in cartesian and spherical coordinates
- A state
- A roughness estimation
- Slope value

Because the graph structure is undirected cyclic network, dynamic pointer handling has been implemented to avoid memory leaks.

In PIGD, the nodes are imported in the same order as the LiDAR data arrives, in vertical lines following the rotation of the LiDAR sensor. This allows conversion and ground detection to be performed sequentially one vertical line at the time, starting with the lowest point, moving upwards. This is managed by a frontier of nodes, propagating through the data.

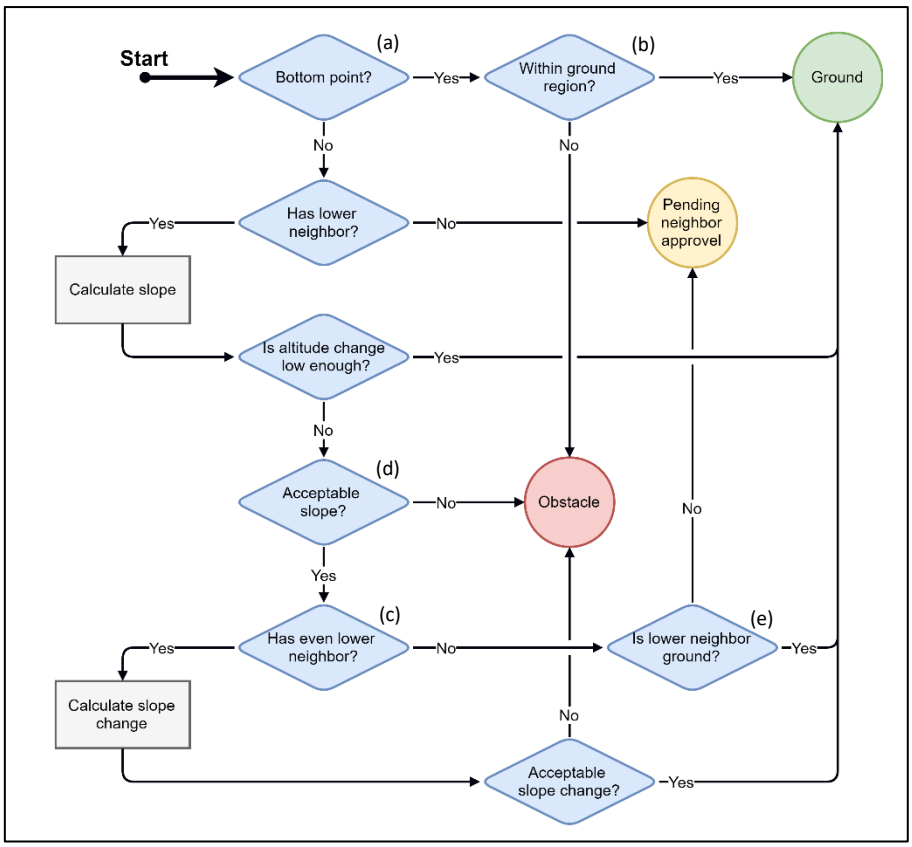


Figure 23 - State diagram of implemented vertical node evaluation for graph-based ground detection.

An overview of the vertical evaluation of every node, is presented in Figure 23. This is the first part of the graph-based ground detection in PIGD, and it evaluates points based on slopes between nodes, to assess if they are traversable from the car's point of view. Vertical node connections are only established if the slope is acceptable, exemplified by red lines in Figure 22.

The first assumption in the implementation is that the surface most adjacent to the car will be flat and level. This allows for a quick evaluation of points in the lowest LiDAR layer (channel 0) to check if they are within a ground region (step *a* and *b* in Figure 23). The remaining points are then evaluated based on their lower neighbors, where "even lower neighbor" in step *c* in Figure 23, means the lower neighbor's lower neighbor.

If a node is reachable with both an accepted slope and change in slope compared to its lower neighbor, it is classified as ground. If the node is lacking connections to lower neighbors to make these slope evaluations, it will rely on neighbors to vouch for it. Vertically, this is possible in step *e* in Figure 23, where the lower neighbor must be ground. Horizontally, the node is labeled "Pending neighbor approval", and is handled in the horizontal part of PIGD.

In the horizontal part of PIGD, connections between horizontal nodes are established based on difference in distances from the LiDAR to the neighboring nodes. This is sufficient, firstly because the high horizontal resolution causes immediate changes (caused by obstacles) to be prominent for these distances. Secondly, since the car mostly follows the direction of vertical connections, slopes between horizontal nodes will seldom be crossed, especially in the expected driving environments. The horizontal connections are exemplified as blue lines in Figure 22.

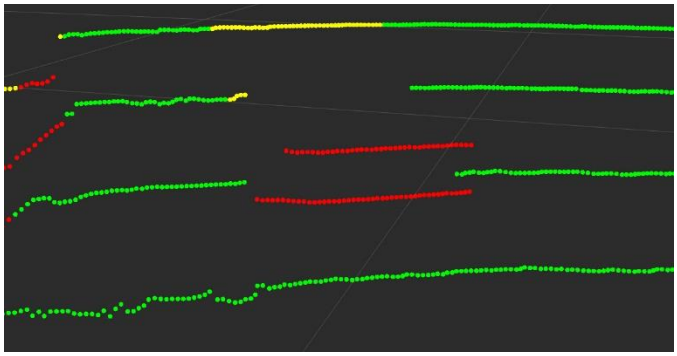


Figure 24 - Example of ground points behind an obstacle, pending neighbor approval (yellow points). Red points are labelled obstacle, and green points are labeled ground.

When a node is considered ground and a horizontal connection is established, the ground label is then propagated through all following or previous "Pending neighbor approval"-nodes. This allows for ground to be recognized behind other obstacles as shown in Figure 24.

Another considered tool for ground detection, is a roughness estimate of the nodes. Here the horizontal connections are used to calculate roughness based on standard deviation of distances to neighboring nodes within a sliding window. An example of this is shown in Figure 25, where the red rectangle indicates a grass covered area. This could be used for differentiating between the asphalt roads and rougher surfaces surrounding the track. The roughness estimation could provide a robust addition to the system, when gaps in the track barriers are present.

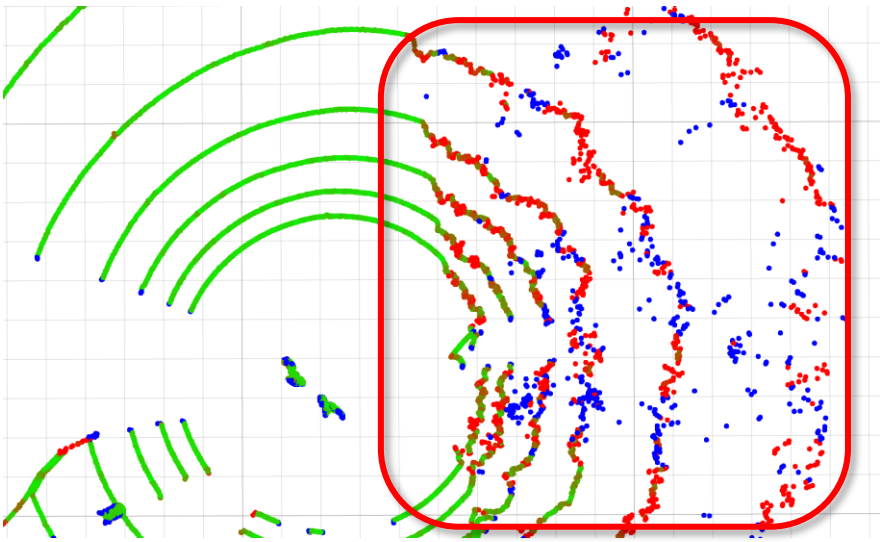


Figure 25 - Example of roughness estimation on LiDAR measurements. Blue indicates saturation of roughness due to lack of sequential points, or simply due to very high roughness. Green color indicates low roughness, red indicates high roughness. The red rectangular shape, indicates an area of interest.

5.3 Tests & Results

Testing PIGD and RANSAC ground detection for comparison, can be done by running the algorithm on the same data. It is much more difficult to make quantitative evaluations on the results, since the large size of the point clouds makes it very hard to obtain the ground truth. A solution could be to manually categorize the true ground of scans, but their sizes would make it far too. Even if a single scan was manually categorized, a single scan for testing would not make a good representation of the general outcome.

Another possibility is to generate test data based on simulations, but our Gazebo simulation is too ideal at its current stage, to be of any value for this purpose.

5.3.1 Error types

To inspect the common errors due to the limitations of the RANSAC algorithm and compare them to the performance of PIGD, both algorithms are tested in four different environments. Examples of the occurring errors are shown in Figure 26, Figure 27, Figure 28, and Figure 29, to illustrate the influence of the errors occurring using RANSAC for ground detection.

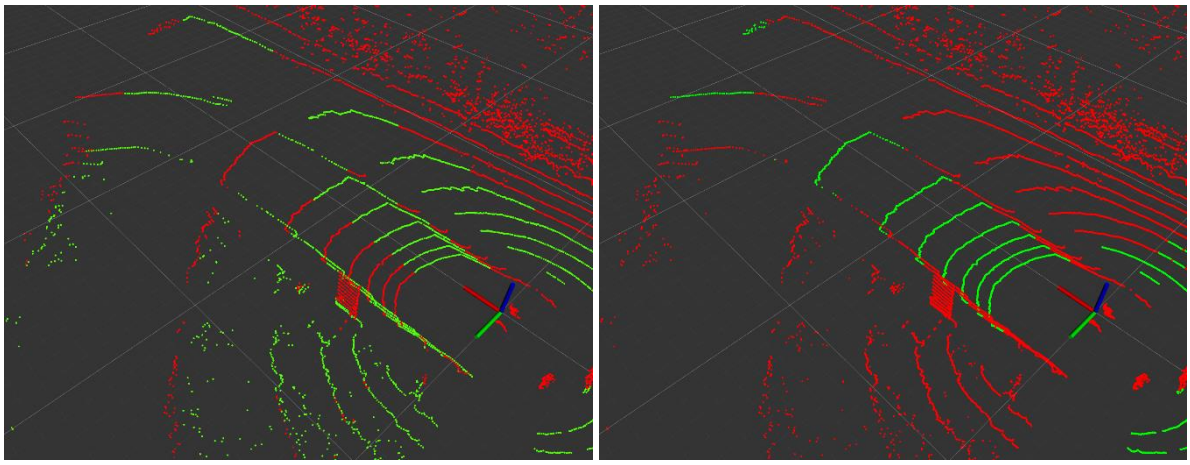


Figure 26 - Comparison of ground detection using RANSAC (left) and PIGD (right) at test track. Red points indicate the obstacle classification and green indicates the ground classification.

In Figure 26, the test was performed on the test track, and shows a frequently occurring error, where the fit of the model plane is affected by the area surrounding the track. The surrounding plane is not necessarily in the same plane as the track, or might be part of an overall ground curvature. This issue is less common when the position filter is applied, since the ground of the track occupy a larger fraction of the point cloud (In Figure 46 in appendix 14.1, the same comparison is performed including the position filter). Although the model fit performs better when filtered, the same issue is still possible, due to the stochastic nature of the RANSAC.

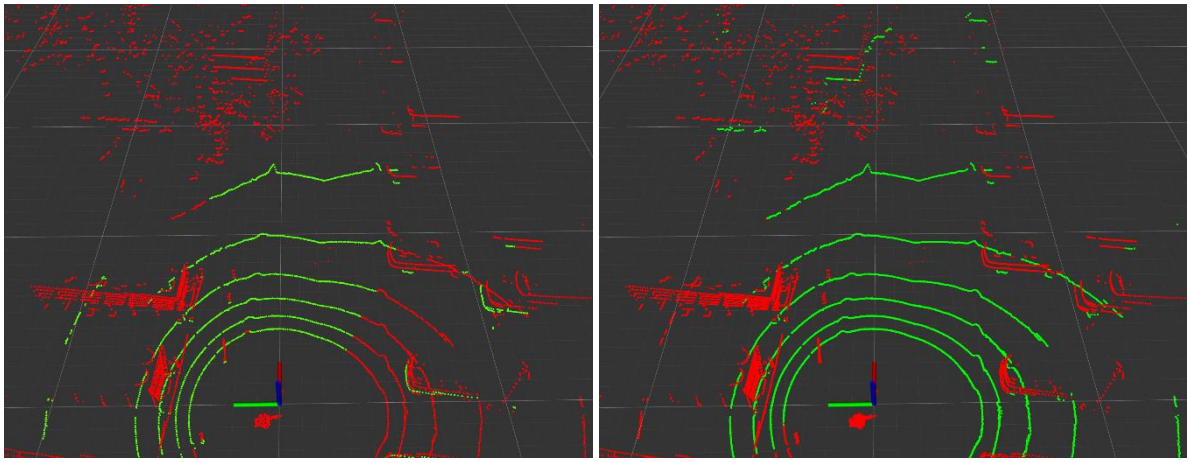


Figure 27 - Comparison of ground detection using RANSAC (left) and PIGD (right) at parking lot. Red points indicate the obstacle classification and green indicates the ground classification.

In Figure 27, the ground is curved and is only partially within the fitted plane. Again, this issue is less common when the position filter is applied, as shown in Figure 47 in appendix 14.1.

In Figure 28 and Figure 29 examples of errors caused by hills or the changing altitude of the road are shown. The test environment of Figure 28 is a very smooth road with a soft curvature (shown in Appendix C - Figure 51). The example in Figure 29 is a very uneven environment compared to the expected conditions at the AUC track, but shows the robustness and capabilities of the PIGD.

Errors occurring when using the PIGD, are mainly consisting of misclassifying small segments of ground as obstacles, when appearing right after a turn seen from afar.

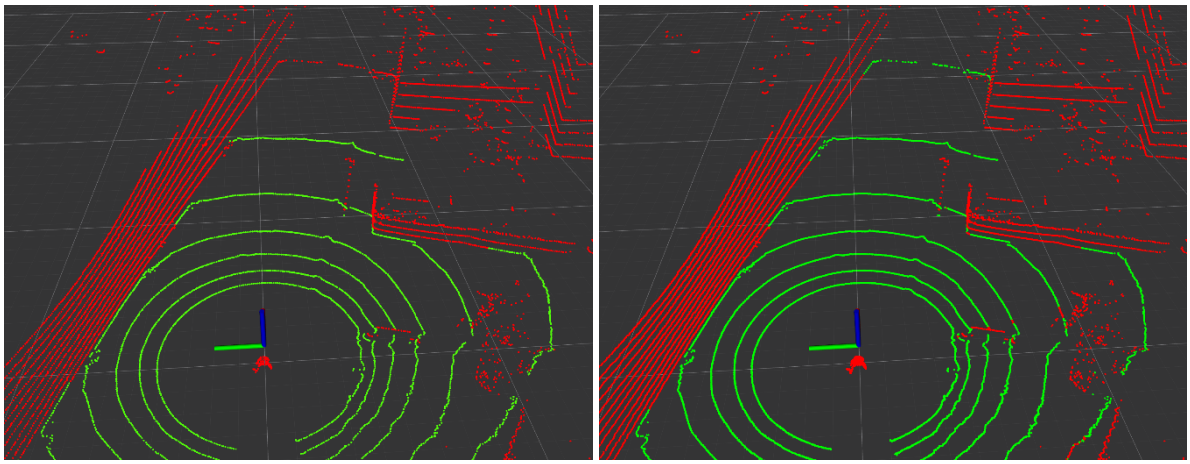


Figure 28 - Comparison of ground detection using RANSAC (left) and PIGD (right) on an uphill road. Red points indicate the obstacle classification and green points indicate the ground classification.

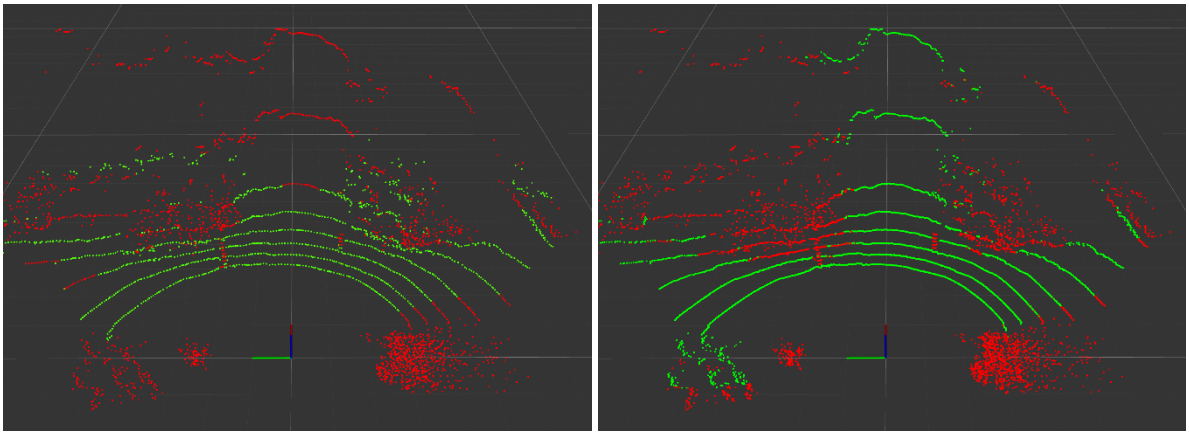


Figure 29 - Comparison of ground detection using RANSAC (left) and PIGD (right) on a running track. Red points indicate the obstacle classification and green points indicate the ground classification.

5.3.2 Driving tests

In order to make a comparison for the performance of the two algorithms, they are both tested on two test drives along the test track, including a left and right turn. Parameters of both algorithms are tuned to perform best for barrier and ground separation in this test environment. The only exception being the number of iterations performed by the RANSAC, since raising this number generally increases the performance to a certain limit. The number of iterations is set to 90, so the computational execution time of the RANSAC and PIGD are similar. The position filter is applied to both algorithms. The RANSAC's inlier distance limit is set to 0.2 m, which works best in the tested environment. Setting this parameter higher results in the barriers to be classified as ground. Setting the parameter lower, results in excessively high sensibility to curvatures of the ground. The PIGD is set with 0.1 m immediate altitude change limit, maximum slope to 15 degrees, and maximal slope change to 3 degrees. The tests are performed on the snow-covered test track, making the ground rough.

Table 5 shows the similarity in execution time between the two ground detection algorithms performed on the test track, when the position filter is applied.

Method of ground detection	**RANSAC**	**PIGD**	**RANSAC** (no position filter)	**PIGD** (no position filter)
Computation time (incl. data import) [ms]	11.1	11.8	35.65	24.51
Standard deviation [ms]	1.02	1.11	1.41	1.78

Table 5 - Execution times of ground detection methods, performed mainly for time comparison. The execution times were performed on an Intel Core i5-5200U CPU with significantly slower single core performance than the CPU in DTU Dynamo.

During the two test drives (58.9 seconds of driving), errors where counted. The error-types and counting criteria are generally defined to facilitate manual counting, avoid questionable errors, and focus on errors influencing autonomous driving.

The errors are defined by 4 categories:

- Type A: Errors consisting of 4 or less clustered points of the track ground, categorized as obstacle. Points neighboring the barriers are ignored to facilitate counting, and because it will not heavily influence the obstacle perception.
- Type B: Point clusters of the track ground no wider than 1 m, categorized as obstacle.
- Type C: Point clusters of the track ground wider than 1 m, categorized as obstacle.

- Type D: Point clusters of a barrier categorized as ground, rendering at least 3m of the barrier length misclassified. It is required that the barrier points are visible to the LiDAR, and more than 0.2 m higher than neighboring ground.

Only errors of the track ground between the barriers are counted, and only the largest error of type A to C, of each scan is counted. Errors more than 25 m down the track are also ignored to facilitate counting.

Table 6 shows the counted error during the test drives. Here it should be noted that among the PIGD counted errors, all 25 type A errors and 2 of the type B errors where cause by a puddle of water on the track (shown in Figure 30). The errors of type D, where mainly end sections of the visible barriers, similar to the barrier in Figure 26 of PIGD. These errors are generally not of big influence when obstacle tracking is later applied.

Method of ground detection	**RANSAC**		**PIGD**	
	[count]	*[%]*	*[count]*	*[%]*
Type A	39	6.62	25	4.24
Type B	93	15.8	3	0.51
Type C	125	21.2	0	0
Total [A+B+C]	257	43.6	28	4.75
Type D	10	1.70	3	0.51

Table 6 - Counted errors during both left and right test drives on the test track

Figure 30 - Image from test drives on test track, with puddle indicated by red rectangle

The roughness calculation did indicate usable information as shown in Figure 25. When the required computation time were inspected, it revealed an increase of approximately 140 %, using a sliding window of just 7 points. The roughness might be useful to analyze the ground plane, but for the current goals of the AUC, its application remains absent.

5.4 Partial Conclusion

It is clear that the PIGD is better suited for ground detection for the AUC. Though the tests are not based on precise quantitative results, an estimation of the improved performance in both ground detection and computational efficiency, has been established. It is of course possible to further handle and filter the errors appearing in the tests with additional algorithms. The amount and gravity of the errors from the PIGD are minuscule compared to RANSAC based ground detection, for the intended purposes and environments. These tests can be considered as worst-case scenarios as the track at AUC is expected to be smoother and with little curvature, but PIGD appears to be a great solution to obtain great robustness and stability.

6 Downsampling & Segmentation

Previous chapters clearly emphasize the heavy influence of the number of 3D points on execution time. To reduce the number of data points without discarding vital information, it is important to use a suitable procedure.

Downsampling by a voxel grid consists of applying a 3D grid with a specified cell size, to divide the 3D space into equally sized volumetric boxed, also known as voxels. Then the number of points within each voxel is reduced to a single data point. This procedure allows for retaining feature such as lines or surfaces represented by the data points (shown in Figure 31). Isolated points will remain and downsampled points of feature such as lines can remain on the same lines. This procedure also equalizes the data density in the point cloud. It is important to choose an appropriate size of the voxels, which must be smaller than the features of interest within the data. In our case, the scan lines are well separated, making this procedure very usable for this project.

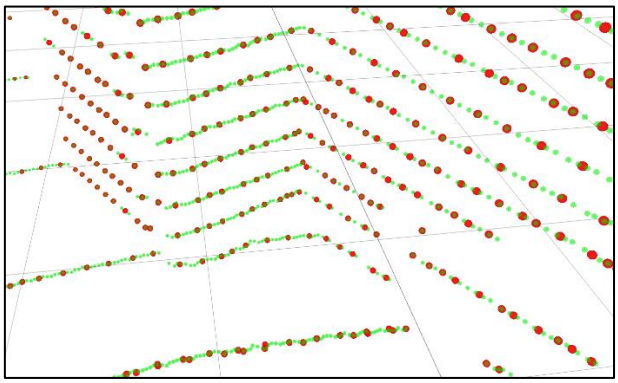


Figure 31 - Visualization of a voxel grid filter applied on 3D points. Green points are the original points, and red points are the downsampled points using a voxel grid filter. The grid cells in the plot are 1 m by 1 m.

The second important procedure to inspect obstacles in the LiDAR data is segmentation. Segmentation between ground and everything else (considered obstacles) has already been discussed in chapter 5 - Ground Detection. The remaining cloud after the removal of ground points needs to be divided into smaller point clouds, each representing a detected obstacle, such as a car, tree or barrier.

In this project, only well-separated obstacles are of interest to segmentation, since the DTU Dynamo only needs to know of the traversability between two obstacles. Another considered aspect of segmentation is to enable later classification or tracking of obstacles.

Many types of segmentation algorithms exist (i.e., by density, model fitting, etc.), but for this project, only separation by Euclidian distance is needed. A visualization of a segmentation performed on the parking lot is shown in Figure 32.

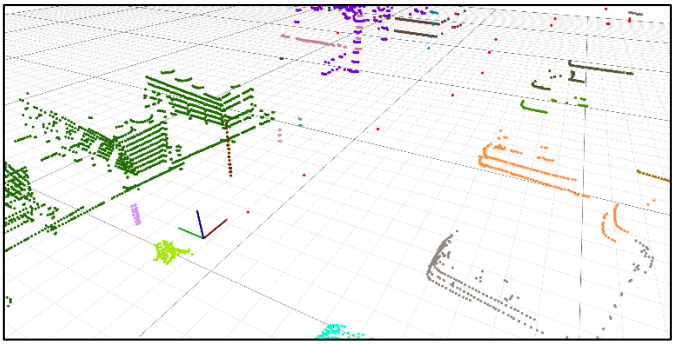


Figure 32 - Visualization of segmented clouds, differentiated by color. On the z-plane, large grid cells are 10 m by 10 m, and small grid cells are 1 m by 1 m.

6.1 Implementation

The Point Cloud Library (PCL) offers functionality for downsampling using a voxel grid filter. The voxel size is set to *0.1 m* (all sides), since this is enough to heavily reduce the point cloud, especially for points near the LiDAR, where the distance between horizontal points is unnecessarily high.

Segmentation by Euclidian distance is also a build in functionality in PCL. The algorithm relies on generating a k-D tree index (k-dimensional tree), to perform fast segmentation. This is a good solution to keep time complexity low with large datasets. Unfortunately, generating a k-D tree is quite time-consuming, and is not worth the extra time, when dealing with smaller datasets.

A custom segmentation algorithm has been implemented, since it is possible to take advantage of the sequence of which data arrives from the LiDAR. The raw data points directly from the LiDAR are sorted by angle, and this is often also true for the obstacles within the point cloud. The custom implementation for segmentation exploits this order, combined with bounding boxes of each segmented cloud, to perform the segmentation. The bounding boxes are also usable for fast proximity evaluation using bounding conditions. The custom implementation is not explained further, but the implementation can be found in Appendix D: Source files.

6.2 Tests & Results

To test the performance of downsampling, tests are performed on both full LiDAR scans and scans after position filtering, to give an estimate of the execution time and point reduction. The measurements are performed on the same data from the from the test track with a voxel size of 0.1 m. The results are shown in Table 7.

	Full LiDAR scan	**Filtered LiDAR scan**
Duration [ms]	1.80	0.430
Variance [ms]	0.199	0.0932
Point reduction [%]	59.9	46.0
Standard deviation of point reduction [%]	4.61	3.96

Table 7 - Performance results of downsampling, showing execution times for both full point clouds, and for filtered point clouds (using the position filter). The execution times were performed on an Intel Core i5-5200U CPU with significantly slower single core performance than the CPU in the DTU Dynamo, and are mainly intended for time comparisons.

The segmentation is performed on both the test track and the parking lot, to test two distinct environments, using a maximal separation distance of *1.5 m*. The test track contains fewer large solid obstacles and more largely distributed point clouds due to the fields surrounding it. The parking lot has many large solid obstacles that are well separated. The computation times are shown in Table 8.

	Custom implementation Test track	k-D tree segmentation Test track	Custom implementation Parking lot	k-D tree segmentation Parking lot
Full LiDAR scan				
Duration [ms]	69.2	49.5	51.8	73.0
Variance [ms]	15.1	3.40	4.61	3.91
Filtered LiDAR scan				
Duration [ms]	2.13	8.43	1.33	9.68
Variance [ms]	1.33	3.61	0.86	2.84

Table 8 - Performance results of segmentation algorithms, showing execution times for both full point clouds, and for filtered point clouds (using the position filter). The execution times were performed on an Intel Core i5-5200U CPU with significantly slower single core performance than the CPU in the DTU Dynamo, and are mainly intended for time comparisons.

6.3 Partial Conclusion

Using the voxel grid filter in PCL allows for an efficient reduction in data, and conserves vital features for perception. For segmentation, it appears that using k-D tree indexing is not particularly advantageous because of the size of the dataset. Because we expect to use position filtering to remove non-essential data points, the custom segmentation algorithm is a more suitable choice.

7 Obstacle Generation

With obstacles reduced to separated 3D point clouds, interpretation and simplification can be performed to each obstacle individually. Since the car's movement is mainly in two dimensions, it makes sense to interpret the local environment in two dimensions as well. Such a 2D map representation is considerably simpler and more convenient to work with for further processing. Because all 3D points from the LiDAR are within the height of the car (because of the position filter), all points represent a possible point of collision. Just by projecting all 3D points to a 2D surface representing the ground, such a map representation is obtainable. These scattered 2D points are not ideal for later processing, such as tracking, and planning and navigation purposes, but more simplification can be accomplished. Transforming the point clusters into regions, defining the outline of the obstacles, will neglect irrelevant points within these regions. These enclosing regions of point clusters are often referred to as hulls.

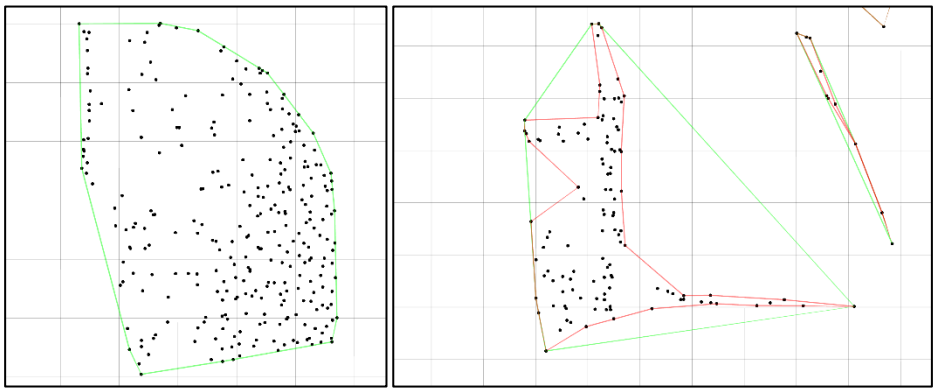


Figure 33 - Point clusters enclosed by hulls. Green hulls represent convex hulls, red hulls represent concave hulls.

Generally, there are two types of hulls, convex hulls, and concave hulls. A convex hull is the smallest possible polygon enclosing a set of points when using only the least possible number of enclosed points as vertices. This means that a convex hull cannot contain any interior reflex angles. Examples are shown in Figure 33.
Concave hulls are any non-convex hull and may contain any angles, and may use all enclosed points as vertices. Any cluster of points have only one convex hull, but many possible concave hulls. Examples of concave hulls are also shown in Figure 33.

7.1 Implementation

There are several algorithms for finding the convex hulls, which are much faster to perform than for concave hull generation. Sadly, convex hulls often expand over larger empty regions, as exemplified in the right image of Figure 33. This imposes a major problem for our track interpretation when barriers curve, because a large part of the track will be unnecessarily interpreted as occupied regions. This is demonstrated in Figure 34, using a simulated track section representing a left turn.

This issue is the reason why convex hulls will not be sufficient for this project, as the convex hull can encapsulate important parts of the traversable region. The PCL includes a fast and efficient convex hull generation algorithm, which is only used for execution time comparison.

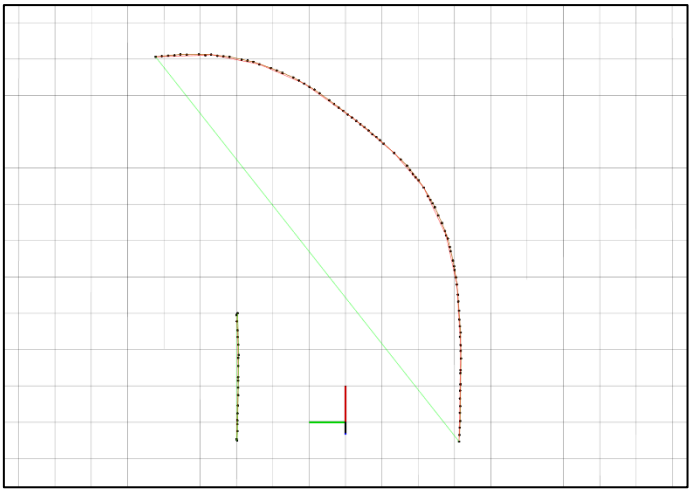


Figure 34 - Point clusters enclosed by hulls. Conceived by simulation of track turn in Gazebo. Green hulls represent convex hulls, red hulls represent concave hulls.

Many algorithms for concave hull generation exists and have different computational complexities and principles for finding solutions. One possibility is to rely on relative distances between neighboring points. Such processes like a k-neighbors approach are very effective when absolute distances and densities are unknown and can be useful even when noise and outliers are present, as exhibited in [12].

Another approach is using absolute distances to regulate the hull concavity, but this requires geometric knowledge about the examined point cloud. PCL includes an alpha shape approach, where a radius defines the grade of hull concavity. The alpha shape approach can be briefly explained as rotating a circle of specified radius, by its circumference from point to point, to mark outlying points, used as vertices for the hull. This approach would be sufficient for this project, but the implementation of the algorithm has several issues when handling the expected environments of this project. In Figure 35, the limitations of the PCL implementation are noticeable.

Most errors occur because the implementation does not allow using the same point twice to define the hull. When a single hull solution is not possible due to this limitation, the PCL divides the cloud into multiple hulls, which is undesirable when describing entire point clouds by a single connected polygon. Connecting all the resulting vertices from this implementation results in the visualized shape in Figure 35. This scrambled order of the vertices is not desired when describing the outline of the obstacle. Furthermore, the implementation can completely ignore single points, most likely because it declares them outliers. This is also visible in Figure 35, where the upper right point, is not part of the resulting hull. Sadly, the implementation does not offer adjustable parameters to handle these issues, nor the documentation to allow informed code adjustments.

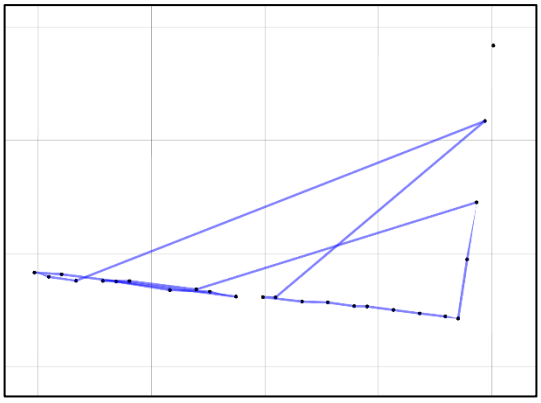


Figure 35 - Lines between resulting vertices from the PCL implementation for concave hull generation by alpha shape based approach. All points from the projected cloud are shown as black spots.

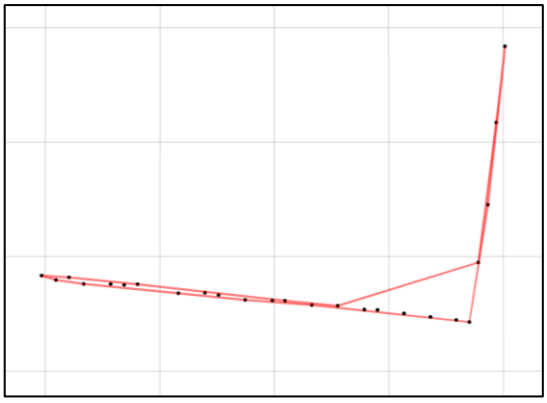


Figure 36 - Lines between resulting vertices from the OSA algorithm implementation for concave hull generation. All points from the projected cloud are shown as black spots.

The approach used in this project is similar to the alpha shape approach. The Swinging Arm algorithm (referred to as SA) described in [13], can be described as using a line segment (the arm) anchored in an initial outermost point, rotating till it encounters another point. This point is then used as the next point for the hull, and the arm is reset with the new point as anchor. This is visualized in Figure 37. The process is repeated untill the entire hull is finished.

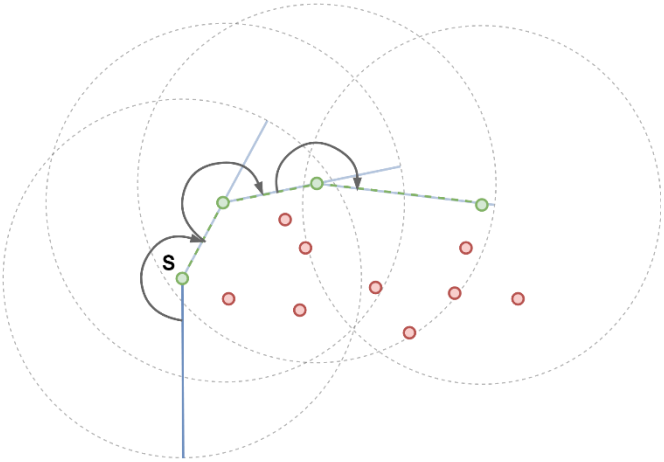


Figure 37 - General visualization of swinging arm algorithm. Green points and lines represent the hull vertices and edges, and red points are the remaining points of the set. S marks the start point and blue lines represent the swinging arm.

This algorithm is well suited for the obstacle description in this project, as the arm length can be set in regard to the car width and the known maximal distance between points in an obstacle cluster, as it is defined by the segmentation process.

The implementation used for this project uses the very same principles as the SA algorithm but with some modification to optimize it for the project use. This custom implantation using the SA principles is referred to as the Optimized Swinging Arm algorithm (OSA). The following alteration compared to the original Swinging Arm Algorithm are implemented in OSA.

- The first alteration of the OSA, is not checking if all points are enclosed by the hull. This is not necessary if the arm length is no shorter than the maximal distance used for segmentation.

- The second alternation is checking for edge crossings, every time a point is considered as an addition to the hull. This is necessary because it is possible for the arm to encounter a point behind the previous point when the arm is reset.
- The last alteration is an optimized indexation. Using k-D tree indexation, the algorithm can be optimized. This is particularly advantageous because this algorithm needs to check distances between points multiple times for each point.

The OSA allows generation of hulls such as the one shown in Figure 36. The outline is ideal for checking for path collisions and comparing hulls for later obstacle tracking. A hull data structure is implemented, which includes the ordered vertices, cartesian extremums, and the timestamp of the LiDAR data used to create the hull. These hull objects are the main elements for the implemented data structure of obstacles. The obstacle data structure is expanded with additional information, such as a unique ID and can include multiple hulls. Most of these additional elements in the obstacle data structure are expansions implemented to enable features for combining sequential LiDAR scans and perform obstacle tracking. These features are further explained in chapter 8 - Obstacle Tracking.

7.2 Tests & Results

When testing the OSA for concave hull generation, and arm length of 1.5 m was used. The algorithm worked as expected, and can generate concave hulls such as the one presented in the right-most plot of Figure 33. The computation time was on average 3.47 ms (standard deviation of 1.25 ms)[4] when performed on the segmented clouds generated from driving on the test track with the position filter. The execution time is very similar to the concave hull algorithm included in PCL, but avoids the discussed issues of the PCL alpha shape algorithm.

7.3 Partial Conclusion

Reducing 3D point clouds for each detected obstacle to a simplified 2D map representation has been achieved using a custom concave hull algorithm. Not only does this reduce the data dimensionality to 2D, but also removes non-essential data points, maintaining the essential obstacle outlines. The achieved computation times can be performed in a sufficiently fast manner, even with real-time performances in mind.

[4] The execution times were performed on a CPU (Intel Core i5-5200U) with significantly slower single core performance than the CPU in the DTU Dynamo, and are mainly intended as a rough time estimate and for time comparisons.

8 Obstacle Tracking

Obstacle tracking can be used to monitor individual obstacles over time. The main reason to implement tracking for this project is merging data over time. Combining LiDAR data from multiple scans can both improve the understanding of individual objects, and provide a more fault tolerant perception. Since the separation of horizontal LiDAR scan lines increases by distance, single distant obstacles can be perceived as multiple due to gaps between scan lines. Combining multiple scans while moving enables a scan like behavior, which remembers previously observed obstacles in these blind regions.

An important downside of combining multiple measurements over time and movement is the accumulation of errors. Noise from every scan will be accumulated along with odometry errors, which over longer distances can be significant as shown in chapter 4 - Sensor Fusion.

8.1 Implementation

To perform maintenance of the tracked obstacles, three steps have been implemented:

1. Trim
2. Match
3. Merge

Because knowledge of previously detected obstacles is needed, the obstacle data structure contains an ancestry list of hulls, referred to as parents. Every time new scans are performed, the first step of the obstacle tracking is to trim the ancestry, removing parents of expired importance. Secondly, newly detected obstacles (hulls at this point) are matched to corresponding parents and merged, while parentless hulls (orphans) are created as new obstacles. The third step is to merge the matched obstacles hulls. For each obstacle element, the parents are combined with their newly matched obstacle hull, into a "*current hull*". The resulting behavior of these steps is shown in Figure 38.

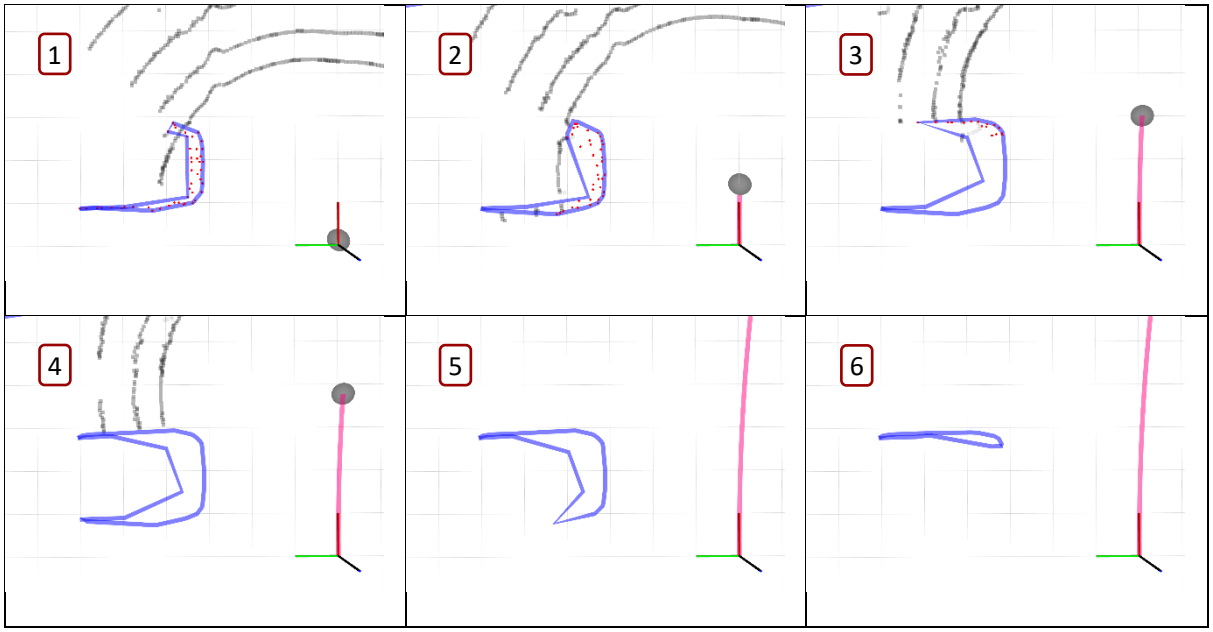


Figure 38 - Obstacle tracking over time, at six sequential positions. Top view (2D) of LiDAR (gray sphere) moving along a path (red line), obtaining obstacle hull (blue line) using points classified as obstacle (red points), and also showing points classified as ground (gray squares).

8.1.1 Obstacle Trimming

Because each parent contains its own timestamp, it is possible to calculate the age and LiDAR position of its creation. These parameters can be used to decide when a parent can be considered irrelevant. Because of the accumulation of error, it is necessary to remove irrelevant parents entirely from the environmental perception. A second reason to remove some parents is to keep processing time short. Both obstacle matching and merging require more computational work when more obstacles and larger ancestry are present. The effect of the trimming can be seen in image 4 to 6 in Figure 38.

For this implementation, only time is used for trimming. Current distance from the parents' creation point could also be added to the trimming criteria, but time trimming appears sufficient for purposes of the AUC. The code implementation does however allow for easy expansion of trimming criteria.

8.1.2 Obstacle Matching

Comparing and matching new obstacle detections with previous obstacle detections is essential for tracking. Matching two obstacles observed from different locations requires identifying common features and characteristics. With the 2D hull simplification of obstacles, one option is simply matching by the proximity of the obstacle outlines. This will enable matching two obstacles seen from two different points of view, only requiring that segments of their shape are proximate. Such situations are expected when observing obstacles at the AUC, because the positional difference between point of views is determined by the movement of the DTU Dynamo between LiDAR scans.

When correlating object outlines defined by vertices and edges, a proximity function introduced in [14]. This proximity function uses a to evaluate the minimum distances from one hull's vertices to another hull's edges and vice versa. Adapting this approach allows for the desired obstacle matching. The odometry of the DTU Dynamo is already accounted for in the point cloud, and consequently, the minimum distance between matching obstacles should be low and related to the error of the odometry between scans.

The implemented obstacle matching uses the proximity function to check for distances shorter than a specified parameter, to evaluate if two hulls are matching. For use at the AUC, this matching requirement is set to a single proximate distance. Since all obstacles at the AUC are stationary, this should be sufficient. The implemented matching function does however allow for requiring multiple proximate edges and vertices, for later use cases.

This matching approach does allow for obstacles within other obstacle hulls, but this will not affect the autonomous driving.

8.1.3 Obstacle Merging

When obstacles have been matched, they can be merged into a single obstacle. Merging is performed by age, and the younger obstacle is merged into the older obstacle. The relative age is obtained from the unique IDs, as all new obstacles are given an ID based on an accumulating number. All parents of the younger obstacle are added to the older obstacles ancestry. To merge all the hulls into the *current hull*, all vertices are added to a single 2D point cloud, and the implemented hull generation is performed (OSA). The results of this merger are noticeable in image 1 to 3 of Figure 38.

8.2 Tests & Results

Using the explained implementation, it was possible to achieve the desired tracking functionality. The following behavior could be seen when driving on the test track (shown in Figure 39), which includes a left turn with barriers on each side.

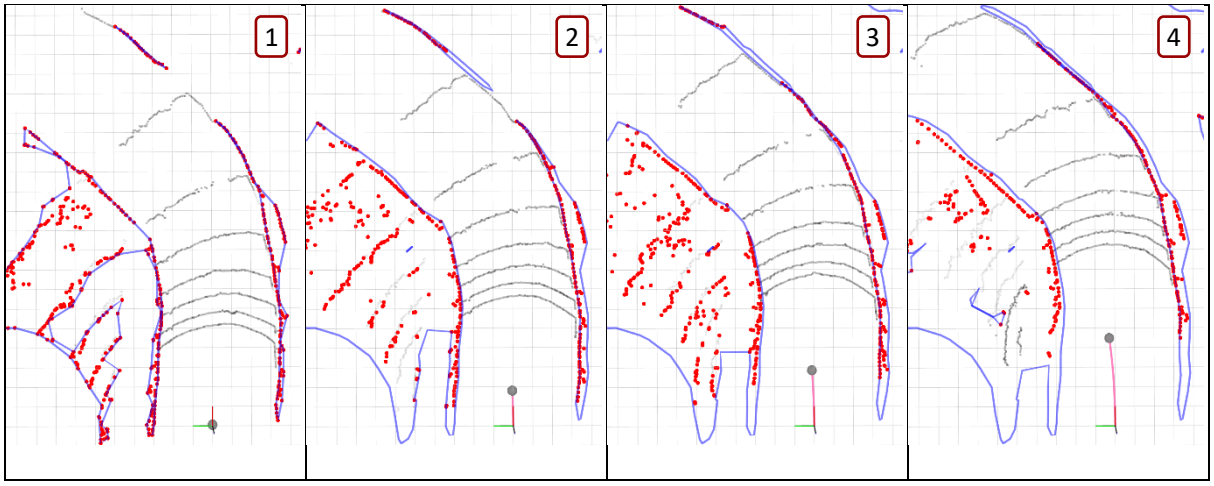


Figure 39 - Obstacle tracking over time performed on the test track, at four sequential positions. Top view (2D) of LiDAR (gray sphere) moving along a path (red line), obtaining obstacle hull (blue line) using points classified as obstacle (red points), and showing points classified as ground (gray squares). Grid cell size is 1 m by 1 m.

Figure 39 presents four images which contain visual examples of the tracking behavior observed through testing. The tracking history is noticeable when inspecting image 1 and 2. In image 1, the uppermost detected points of the right-side barrier, are created as an obstacle. Then in image 2, the obstacle region is using the previous detections to keep the region marked as occupied. This happens even though the current scan does not provide any new detected obstacle points in this region.

Merging two previously separated obstacles is also noticeable in image 2 and 3. The gap in the right-side barrier is closed and the two previous barrier segments are merged into one obstacle.

Finally, image 4 again exemplifies the achieved detection tolerance when combining sequential scans. It shows that when the furthest visible part of the right-side barrier is not detected, but is maintained using the observations of previous scans.

Another important observation during tests is the growth of obstacles due to the accumulated error. Such growth was observable but not very prominent and is investigated further in chapter 10 - System Tests & Results. Fortunately, the growth can be limited by the trim time, because it limits the amount of accumulated error and inconsistencies between the series of scans, used to define the resulting obstacle hulls. The accuracy of the LiDAR is very high in comparison to the odometry, which is why the odometry is the main source of error. Limiting the trim time will also limit the accumulated odometry error.

The best results seemed to be present with a trim time of 1 - 2 seconds, and a proximity distance of 1 meter. This setup was tested on both the test track and the parking lot, to ensure representation of two very distinct environments, and resulted in the execution times in Table 9.

Environment	Trim time [s]	Avg. execution time [ms]	Std. dev. execution time [ms]
Parking Lot	1	6.58	1.81
Parking Lot	2	10.6	3.00
Test Track	1	8.45	2.79
Test Track	2	13.2	4.44

Table 9 - Execution times of obstacle tracking including position filtering. Performed for relative comparison, although on an Intel Core i5-5200U CPU significantly slower than the CPU in the DTU Dynamo.

From Table 9, it is as noticeable that the execution time significantly increases the overall execution time, although it is deemed acceptable. The trim time of 2 seconds at 25 km/h should be sufficient to cover the entire blind horizontal gap, discussed in section 2.2.3.1 - LiDAR Selection.

8.3 Partial Conclusion

From the results achieved using the implemented obstacle tracking, it appears combining data from multiple LiDAR scans can provide an improved and more robust perception for the AUC. Merging multiple LiDAR scans does introduce accumulation of errors, but this can be kept at a sufficiently limited level using data trimming. The implementation allows for further expansions, but the simple object tracking principles are sufficient for the expected use.

9 System Implementation Overview

9.1 Final Data Preparation

With all the LiDAR data processed along with odometry, it is important to consider what information is essential to further processes. The obstacle perception will be the foundation for navigation and planning and must be described as simply as possible, without discarding significant information. The reduction from raw data to obstacle hulls already achieves much of this goal, and approximately 288.000 3D points are on average converted into less than 2000 2D points, every second.

These resulting 2D obstacle hulls are not the only information of interest and more can be added to this 2D occupancy map.

9.1.1 Visibility

Because of the 3D properties of the LiDAR, some obstacles can be detected behind other obstacles, in respect to the line of sight in the 2D map representation. In other words, the LiDAR is able to look over obstacles. These obstacles are mostly completely irrelevant for the autonomous driving and can be unreachable for the car. These obstacles could be removed from the data sent to navigation and planning, but are simply labelled as non-visible. This allows the planning and navigation to ignore these obstacles if need be.

The implementation for this visibility check is based on algorithms for generation of visibility polygons using an angular sweep, often used in computer games. (An interactive description of such an algorithm can be found in [15]). Instead of generating this visibility region, the implemented algorithm simply marks the visible obstacles if a single edge of the entire obstacle is visible. This preserves partly visible barriers such as barriers in turning track sections, as they are still vital for path planning. Obstacles behind or even inside other obstacles are marked invisible, since they are irrelevant to path planning.

9.1.2 Data Structure

The final data structure of the obstacle perception forwarded to the navigation and planning processes includes:

- **A timestamp** of the LiDAR data used to generate obstacles hulls.
- **Position and orientation** of the car at the time of the timestamp.
- **A list of detected obstacles**, each containing:
    - **Obstacle ID**
    - ***Current hull***
    - **Number of parents**
    - **Visibility boolean**

9.2 Data Flow

While Figure 13 (in 2.2.3.3 - New Platform Overview) showed a simplified overview of the processing steps throughout the LiDAR perception, Figure 40 shows a much more detailed overview. The diagram illustrates the data flow and processing throughout the entire LiDAR perception, described in all the previous chapters. The yellow processing steps are implemented as optional, depending on the desired performance of the LiDAR perception as a whole. When implemented, some of the processes explained in previous chapters are intertwined to optimize execution time. Moreover, Figure 40 shows red processing steps, indicating where it is possible to implement classifiers for individual obstacles in both 2D and 3D. This could

be used to classify the gate poles by LiDAR but is not included in this thesis, as this task has been delegated to other students in DTU Roadrunners.

In the final performance setup, both the RANSAC addition and the visibility check is used. The additional RANSAC can be used to avoid problems of undetected obstacle shadowing as previously shown in Figure 24, but mainly to filter ground points within obstacles. In environments such as the test track, the surrounding tall grass fields are detected as non-traversable regions, later handled as obstacles. These large field obstacles contain many ground points and removing them using the additional RANSAC is much faster than leaving the points to be processed in the subsequent processes. This could also apply to crowds of people, but this will not be known till data from the AUC has been captured using the *new platform*.

Finally, the roughness estimate is not used in the final setup, due to its slow execution time and lacking necessity in the level 1 of AUC.

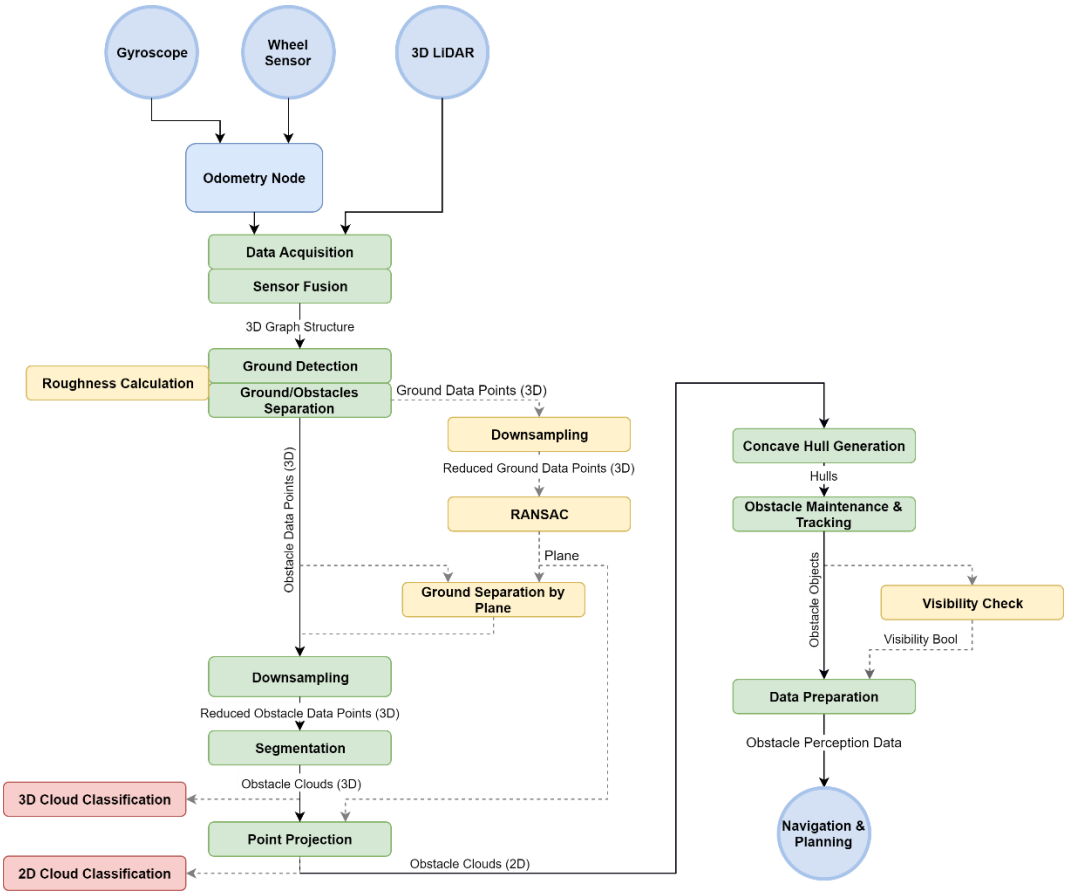


Figure 40 - Detailed functional overview of LiDAR perception implementation in steps. Green indicates core processes, yellow indicate optional steps, red indicate possible future extensions to the perception, and blue indicate non-perception sources or processes.

10 System Tests & Results

In previous chapters, several test sites were used to test individual functionality of implemented processing steps in a variety of distinct environments. These test locations were mainly used to test and achieve robustness. With high robustness already established, system tests are performed on the test track, which resembles the conditions at the AUC track the most. The tests are evaluated with regards to the final output of the perception implementation.

The parameters used for ground detection on the test track is set as following: 0.06 m immediate altitude change limit, maximum slope to 5 degrees, and maximal slope change to 3 degrees. The trim time is set to 2 seconds unless otherwise specified.

All tests are performed by pushing the DTU Dynamo manually at various speeds, and with manual steering. Testing the system with navigation and planning in a closed loop setup will occur in the near future, but the navigation and planning were not ready for testing during the work of this report and mechanical issue with the steering control further disallowed such tests.

10.1 Detection Ranges

To test the detection ranges of barriers, a straight path towards a barrier segment was tested, as shown in Figure 41. Firstly, the test showed a maximum detection range of 33.4 m from the LiDAR position, not as much as calculated to be ideally possible in Table 4, but well beyond the required detection range. Inaccuracies of the LiDAR mounting and curvature of the track can affect these results, but the performance is mainly dependent on the sensitivity of the ground detection algorithm. The points on the barriers were visible to the LiDAR from a distance of approximately 40 m.

Secondly, the test showed the horizontal gap between the LiDAR layers, hiding the barrier in the interval 16.6 m to 28.2 m. This is again affected by the environmental conditions and sensibility parameters of the ground detection algorithm. The trimming time of 2 seconds, will be able to minimize the gap's influence on the perception, especially at a higher speed when the detection range is of higher importance. Finally, the test also shows the minimum detection range to the barrier to be 0.71 m from the front of DTU Dynamo.

Figure 41 - Test setup for testing barrier detection range.

10.2 Obstacle Growth & Movements

Effects of the accumulated error on the tracked obstacles appear minimal using 2 seconds of trimming time. To visualize the small growth due to LiDAR noise, all obstacle hulls generated throughout 30 seconds

of stationary measurements are shown in Figure 42. Here, only small variations (apart from the upper left corner, caused by human movement) are detected, especially on the inner sides of the barriers.

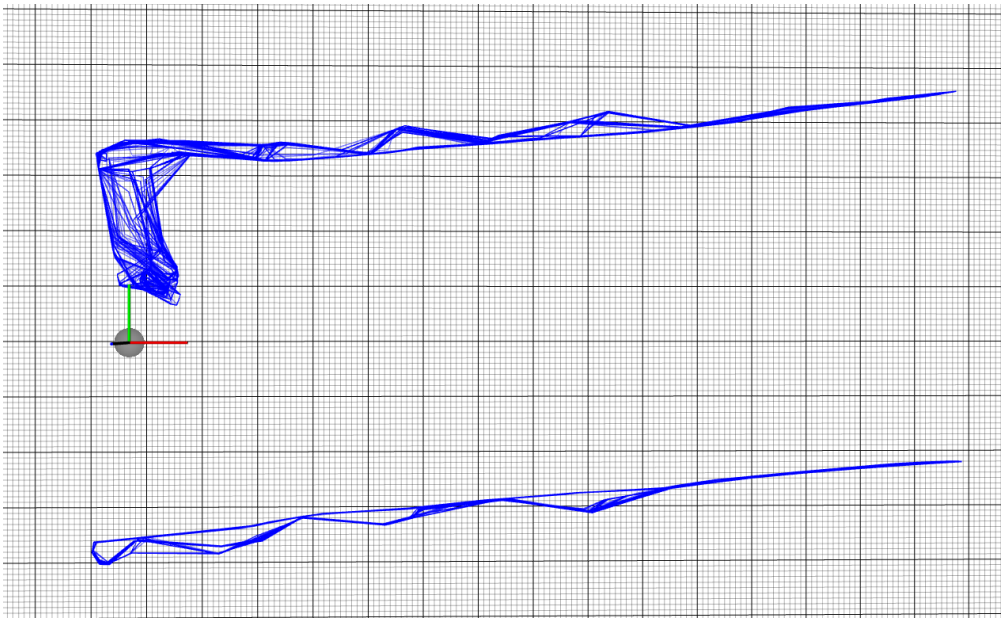


Figure 42 - Stacked obstacles hulls (blue lines), detected during 30 seconds of stationary positioning on a straight track segment. The LiDAR position is marked by the large gray disk. The large variety in hull shapes above the LiDAR upper left corner is caused by human movement next to the car. Large grid cells are 1 m by 1 m, and the smaller grid cells are 0.1 m by 0.1 m.

To inspect the growths and movements of obstacles due to odometry error, the setup shown in Figure 43 was used. With trim time set to infinity, a test run of approximately 40 m towards the wooden plate was performed, shown in Figure 44. The test clearly shows the movement of the wooden plate, since it should only result in a flat hull. The movement of the hull is approximately 2 m, corresponding well with the odometry performance tested in section 4.1 - Odometry.

Figure 43 - Test setup on the test track. The test includes a standing wooden plate at the end of a long straight track section.

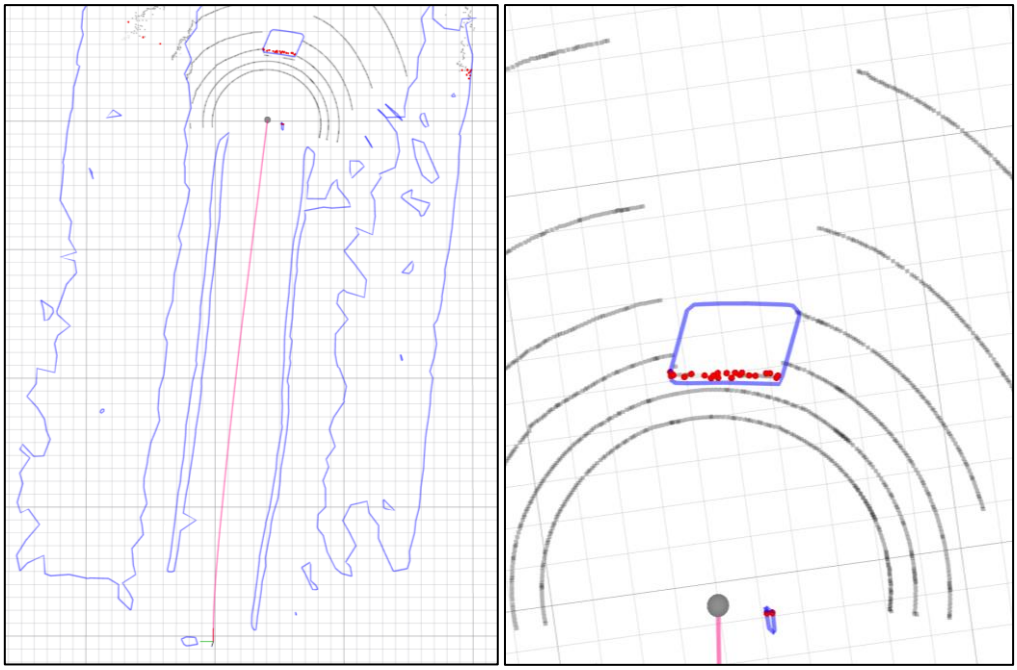
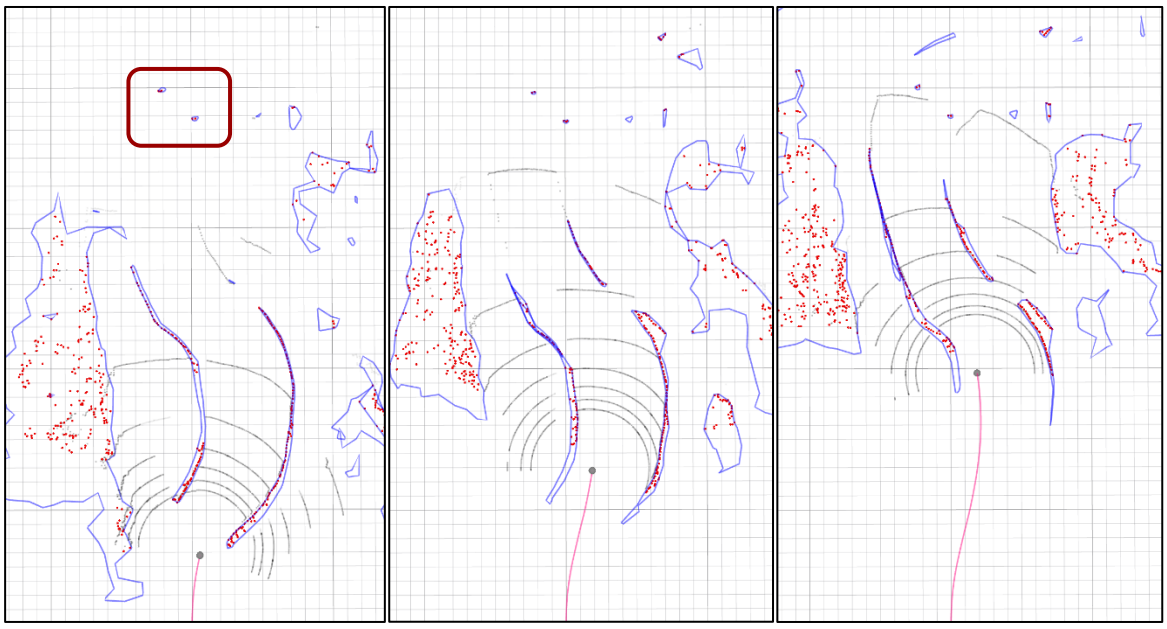


Figure 44 - Obstacle tracking using infinite trim time. Obstacle hulls are shown in blue, driven path is shown by the red line, and gray points are final detected ground points. Both images show the final outcome, while the right image is a magnification on the obstacle hull, corresponding to the wooden plate. The small grid cells are 1 m by 1 m.

10.3 Track Detection

To test the general performance of track detection, the implemented system has been tested on various track segments, built on the test track. This included turning track segments, straight track segments, chicanes, and a track split. Various combinations of these test setups were performed using on-track obstacles, gates (simulated using plastic cylinders shown in Figure 49 in Appendix C), and gaps in the barriers. The performance is generally very good and no major errors were detected affecting the generated map representation. To show the common behavior of the system implementation, a test of driving through a chicane with a large gap and a gate is visualized in Figure 45. This example also shows how the gate cylinders are detected from afar due to their height.



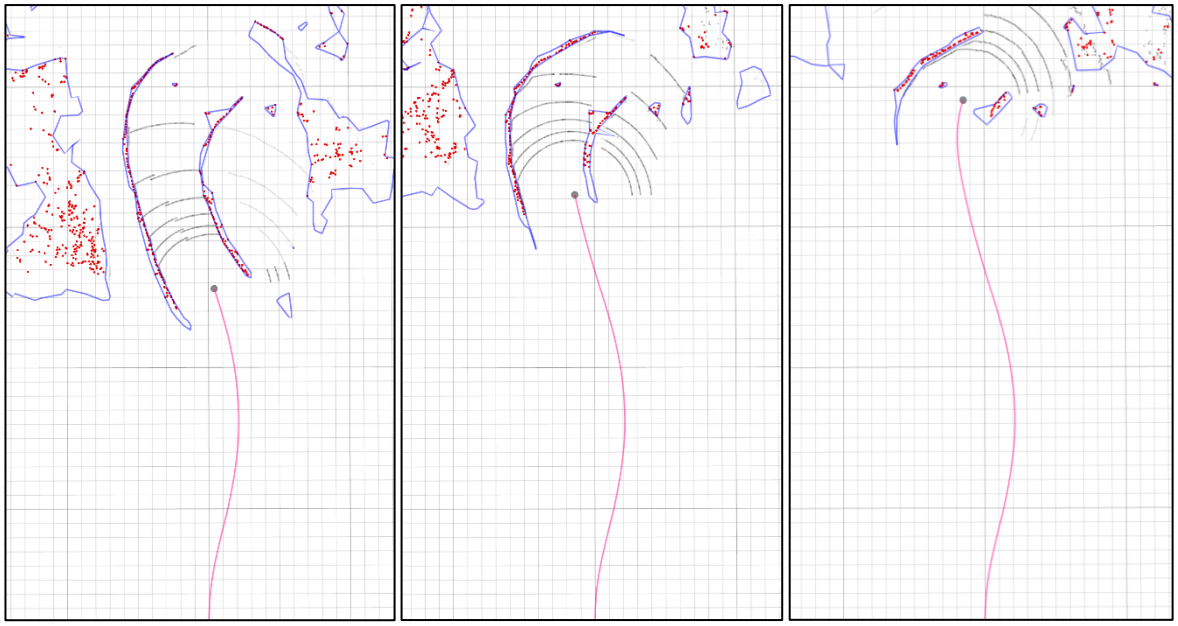


Figure 45 - Test drive through chicane with a large gap and a gate, visualized in six steps. Obstacle hulls are shown in blue, driven path is shown by a red line, and gray points are detected ground points. A red rectangle I inserted on the upper left image to show the initial position of the gate placed in the track segment. The small grid cells are 1 m by 1 m.

10.4 Execution Times

The execution time of the entire processing of the whole implemented system was tested on the DTU Dynamo[5]. Tests were performed over entire track sections at both 10 Hz and 20 Hz, resulting in the execution times shown in Table 10.

LiDAR scan rate	**10 Hz**	**20 Hz**
Avg. execution time per scan [ms]	23.0	15.5
Std. deviation of execution time per scan [ms]	1.81	5.01

Table 10 - Execution times of system implementation.

The lower execution times for the 20 Hz scan rate is caused by the lower horizontal point resolution at this scan rate. Both execution times are more than sufficient to achieve real-time execution.

[5] Has an Intel Core i7-8650U CPU

11 Conclusion

Throughout this project, the previous autonomous platform of DTU Dynamo was initially analyzed and then augmented to meet the expected perception requirements of the challenges in the Autonomous UrbanConcept Competition at the Shell Eco-Marathon 2018.

LiDAR perception using a Velodyne VLP-16 LiDAR has been implemented into the existing system to inspect the local environment and tracking obstacles in real-time. Detection of elements essential to the challenges in the competition has been tested in various environments, to achieve a robust and fault-tolerant solution. These tests have shown sufficient levels of performance to consider to the expected requirements satisfied.

Data processing has been divided into the following multiple steps, allowing for easy system adjustments, future system expansions, and testing of each step's individual performance:

- **Data Acquisition** retrieves LiDAR data from the hardware and filters non-essential data points.
- **Sensor Fusion** combines LiDAR data with odometry, adding information to the measurements.
- **Ground Detection** provides a robust segmentation of ground and obstacles.
- **Downsampling and Segmentation (3D)** allows for fast data reduction and separation of individual obstacles.
- **Obstacle Generation (2D)** simplifies obstacle point clouds into 2D hull representations.
- **Obstacle Tracking** tracks obstacles between scans while combining measurements into more detailed obstacle interpretations.
- **Data Preparation** prepares perception information for navigation and planning purposes.

Overall, the high amount of raw data provided by the LiDAR has been successfully fused with odometry and reduced into only essential information for situation awareness. The resulting interpretations from these processes will be the foundation for navigation and planning, along with computer vision. This will enable DTU Roadrunners to meet the expected perception requirements for the Autonomous UrbanConcept Competition at the Shell Eco-Marathon in July 2018, and thus hopefully complete all of the five competition challenges.

12 Future Work

In the following, proposals for future work, based on current and projected future perception requirements of the AUC will be described.

The most interesting test which was not possible to perform during this thesis, is to test the performance of the perception in a closed loop system, where the perception is used for navigation and planning to control the autonomous driving. Many unexpected issues might occur when testing the performance of the autonomous system as a whole, as it is impossible to predict the real-world behavior entirely.

12.1 Code Optimization

Many of the code implementations can be optimized to achieve faster execution time. As shown through testing, the computer system in the DTU Dynamo is more than capable of executing the implemented perception processes in real-time. But in future iterations of the autonomous system, more raw data might need to be processed. Some process implementations can merely be optimized by using faster code implementations, and many of the processing steps can be optimized for parallel computing.

12.2 Computer Vision Collaboration

As errors are inevitable in sensory systems, combining the environmental perception of image sensors with the LiDAR perception could provide a better evaluation of data, especially when errors occur.

12.3 Roughness

As mentioned in 5 - Ground Detection, a simple roughness estimate has already been implemented. This could be used to classify ground regions such as different types of pavement or grass areas. Subsequently, this functionality could enable the car to identify the track without depending on barriers.

12.4 Moving Obstacles

As mentioned in chapter 8 - Obstacle Tracking, tracking of moving obstacles has already been considered. At the current level of AUC, this is not a necessity, but can be expected in future competitions.

12.5 Obstacle Classification

As mentioned in section 9.2 - Data Flow, classifiers can be applied to the individual obstacles. This would allow classification of elements such as gates, barriers, cars, traffic signs, humans, etc. Such information could both be used for path planning or used to indicate regions of interest for other perception sensors, such as the camera in the DTU Dynamo.

13 References

[1] Shell, "About Shell Eco-Marathon," [Online]. Available: https://www.shell.com/energy-and-innovation/shell-ecomarathon/about.html. [Accessed April 2018].

[2] Shell, *2018 Autonomous UrbanConcept Competition Rules - Version 13,* 2017.

[3] Shell, "Shell Eco-marathon 2018 Global Rules, Chapter I," 2017. [Online]. Available: https://www.shell.com/energy-and-innovation/shell-ecomarathon/europe/for-europe-participants.html. [Accessed April 2018].

[4] H. S. Høj, "Platform Integration for Autonomous Self-Driving Vehicles," 2017.

[5] E. Ackerman, "Lidar That Will Make Self-Driving Cars Affordable," *IEEE Spectrum,* vol. 53, no. 10, p. 14, 2016.

[6] A. Filgueira, H. González-Jorge, S. Lagüela, L. Díaz-Vilariño and P. Arias, "Quantifying the influence of rain in LiDAR performance," *Measurement,* vol. 95, pp. 143-148, 2016.

[7] Velodyne, "VLP-16 Data Sheet," [Online]. Available: http://velodynelidar.com/vlp-16.html. [Accessed April 2018].

[8] SICK, "MRS1000 Outdoors is our fourth dimension (Datasheet)," [Online]. Available: https://www.sick.com/ag/en/detection-and-ranging-solutions/3d-lidar-sensors/mrs1000/c/g387152. [Accessed April 2018].

[9] D. Scaramuzza , R. Siegwart and I. R. Nourbakhsh, in *Introduction to Autonomous Mobile Robots*, 2nd ed., MIT Press Ltd, 2011, pp. 252-259.

[10] F. Moosmann, O. Pink and C. Stiller, "Segmentation of 3D Lidar Data in non-flat Urban Environments," *2009 IEEE Intelligent Vehicles Symposium,* pp. 215-220, 2009.

[11] Z. Zhu and J. Liu, "Graph-based Ground Segmentation of 3D LIDAR in," *2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA),* pp. 1-5, 2014.

[12] A. Moreira and M. Y. Santos, "Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points.," in *Proceedings of the Second International Conference on Computer Graphics Theory and Applications*, Barcelona, Spain, 2007.

[13] A. Galton and M. Duckham, "What is the region occupied by a set of points," in *GIScience 2006*, pp. 1-9, 2006.

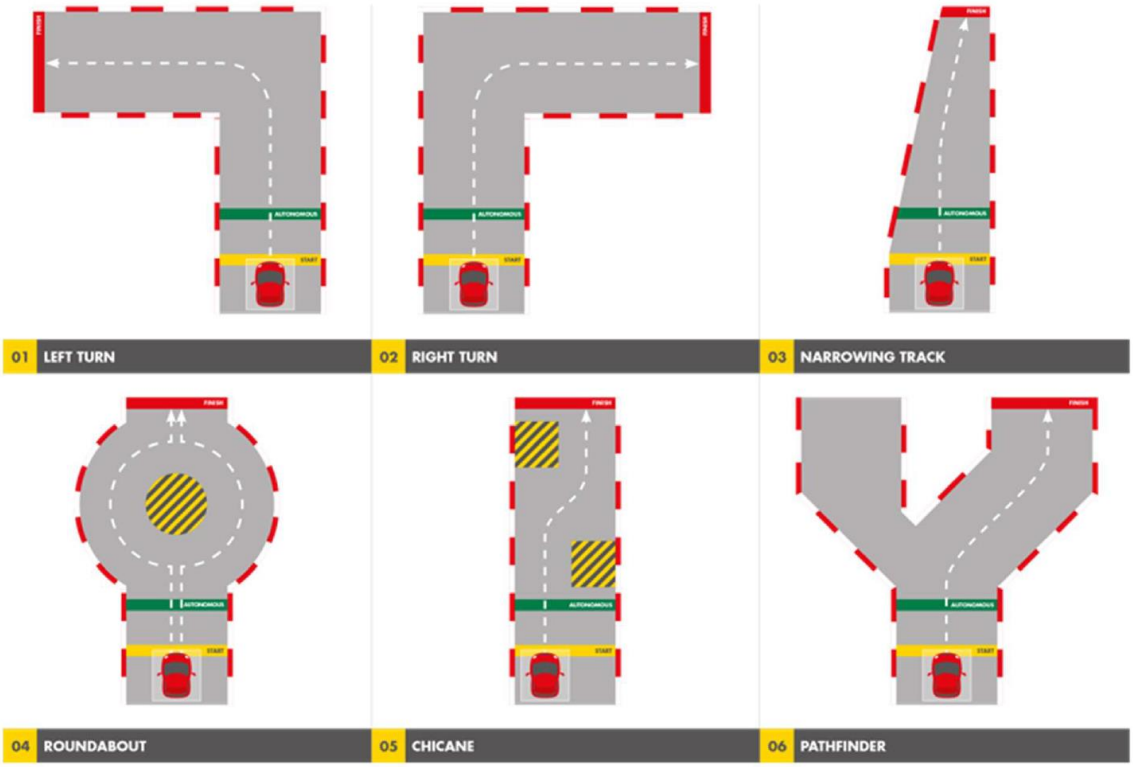
[14] J. C. Andersen, "Mobile Robot Navigation," pp. 114, 2007.

[15] A. Patel, "2d Visibility," 2018. [Online]. Available: https://www.redblobgames.com/articles/visibility/. [Accessed April 2018].

14 Appendix

14.1 Appendix A: Complex Track Sections

The source of the following track layout illustrations is [2].



14.2 Appendix B: Ground Detection Comparisons

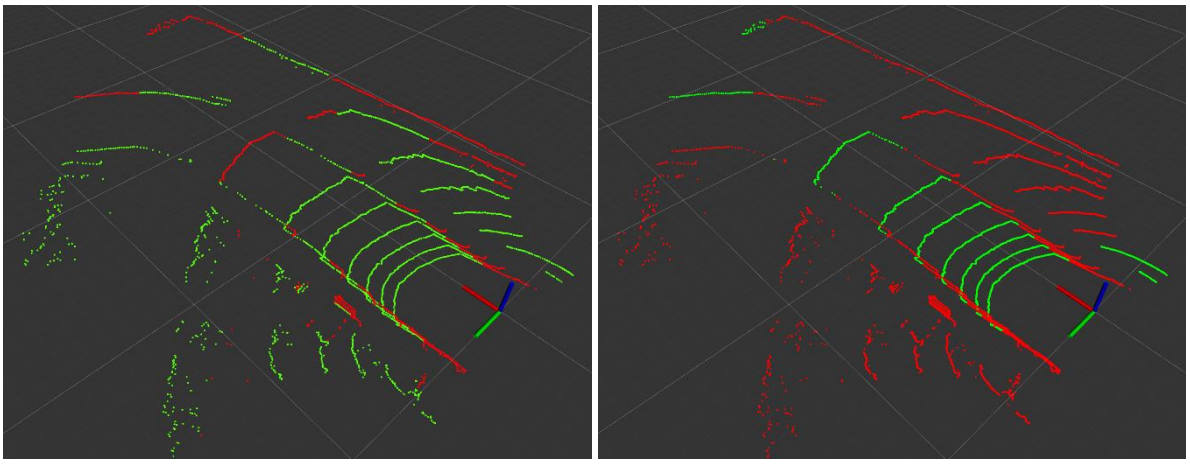


Figure 46 - Comparison of ground detection using RANSAC (left) and PIGD (right) at test track with position filter applied. Red points indicate the obstacle classification and green indicates the ground classification.

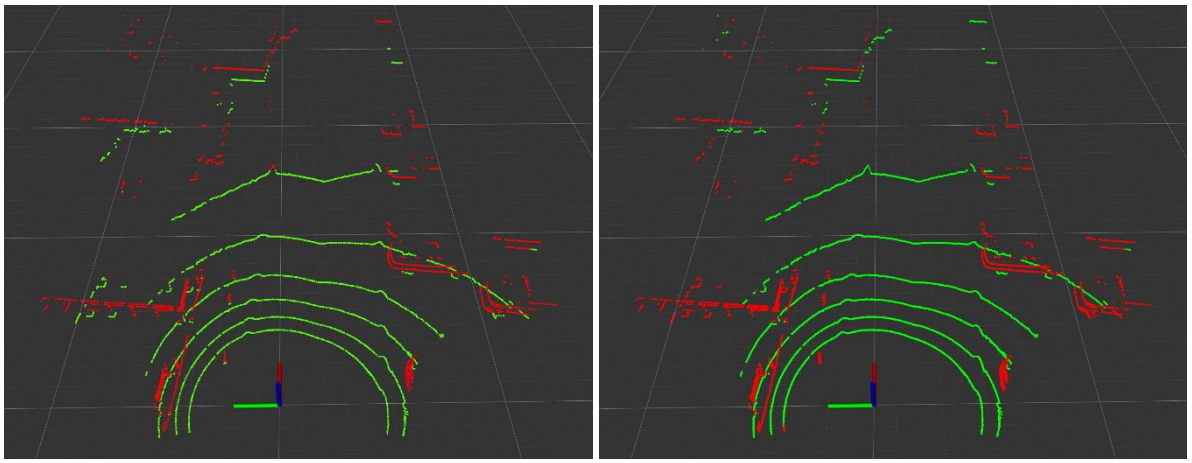


Figure 47 - Comparison of ground detection using RANSAC (left) and PIGD (right) at parking lot with position filter applied. Red points indicate the obstacle classification and green indicates the ground classification.

14.3 Appendix C: Test Locations

Figure 48 - Test track with barriers arranged in a S shape

Figure 49 - Test track and a plastic cylinder used to simulate a gate pole

Figure 50 - Parking lot used as test environment

Figure 51 - Uphill road used as test environment

Figure 52 - Running track used as test environment

14.4 Appendix D: Source files

The source files for the ROS implementations described in this thesis, can be found in the digital appendices, compressed in a zip file named "Appendix_D.zip".

Table of Figures

DTU Electrical Engineering
Department of Electrical Engineering
Technical University of Denmark

Ørsteds Plads
Building 348
DK-2800 Kgs. Lyngby
Tel: (+45) 45 25 38 00

www.elektro.dtu.dk