Opportunistic Localization and Reactive Navigation in Confined Environments

Development of automated driving system (ADS) for underground mines

Master's Thesis in Complex Adaptive Systems

ALEX GORSKIY
HENRIK GRÖNBÄCK

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Opportunistic Localization and Reactive Navigation in Confined Environments

Development of automated driving system (ADS) for underground mines

Alex Gorskiy
Henrik Grönbäck

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2023

Opportunistic Localization and Reactive Navigation in Confined Environments
Development of automated driving system (ADS) for underground mines

Alex Gorskiy
Henrik Grönbäck

Supervisor: Michael Årnevall, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2023
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Modified RC vehicle used as an autonomous development platform.

Typeset in LaTeX
Gothenburg, Sweden 2023

Opportunistic Localization and Reactive Navigation in Confined Environments
Development of automated driving system (ADS) for underground mines
Alex Gorskiy
Henrik Grönbäck
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

As there is no global navigation satellite system (GNSS) coverage underground, it is of great value to the mining industry to design an automated driving system (ADS) capable of navigation and localization without relying on GNSS, infrastructure or prior sensor mapping of the entire environment. An ADS was developed to navigate such confined environments and was verified to be highly accurate in a confined office environment. The proposed ADS utilizes a topological map of the environment represented as a bidirectional graph with nodes and edges corresponding to intersections and corridors respectively. The ADS navigates reactively when travelling along corridors by identifying the road edges and staying within the bounds of the road. When an intersection is detected, the ADS makes a turn or continues along the path depending on its global goal in the topological map. The proposed ADS was also tested on an unconfined environment to evaluate its ability to generalize. While the proposed ADS was unable to navigate in an unconfined environment, it is likely that an ADS designed for confined environments would be able to generalize to unconfined environments if the environment can be transformed into a confined representation.

Keywords: opportunistic localization, reactive navigation, confined environments localization, underground localization.

Acknowledgements

We would like to thank CPAC Systems AB for providing us with the resources and working space to conduct this thesis, Peter Forsberg, our examiner and academic supervisor, for help and feedback along the way, as well as Michael Årnevall, our industrial supervisor, for his guidance and valuable insights.

Alex Gorskiy and Henrik Grönbäck, Gothenburg, June 2023

Thesis advisor: Michael Årnevall, CPAC Systems AB
Thesis examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADS	Automated Driving System
GNSS	Global Navigation Satellite System
GUI	Graphical User Interface
ICP	Iterative Closest Point
LiDAR	Light Detection And Ranging
RFID	Radio Frequency Identification
ROS	Robot Operating System

Contents

1
Introduction

The world is becoming increasingly more autonomous and the mining industry is no exception. However, while vehicles above ground can safely rely on a global navigation satellite system (GNSS) for navigation and localization, that is unfortunately not the case for vehicles operating underground as there is no GNSS coverage. Designing an automated driving system (ADS) capable of tackling this challenge can bring numerous benefits to the mining industry as autonomous vehicles have the potential to be both safer and more productive than vehicles driven by humans.

1.1 Background

Numerous different approaches exist to tackling underground navigation and localization. Zhuli Ren and Liguan Wang proposed constructing a 3D LiDAR map of the mine using simultaneous localization and mapping (SLAM), and then point cloud matching against that map to localize the vehicle in the mine [1]. Point cloud matching against a map is however computationally expensive and prone to errors as when the observed environment changes it will no longer match the map. To tackle that, Kristin Nielsen proposed to instead compare high-level features extracted from a 2D LiDAR map with features extracted from the current data to localize the vehicle [2].

Radacina Rusu et al. proposed using radio frequency identification (RFID) to localize the vehicle by having the vehicle detect RFID tags placed around the mine [3]. Robert Jones et al. proposed a hybrid approach where RFID tags are used to reset the drift errors accumulated by inertial measurement unit (IMU) dead reckoning [4].

When driving through tunnels, accurate localization is unnecessary as the vehicle only needs to navigate safely through the tunnel and then make a decision when it has arrived at an intersection. With that in mind, Mauricio Mascaro et al. proposed using a topological map represented as a bidirectional graph with tunnel and intersection nodes [5]. The map is built by visiting all the nodes in the mine and then storing LiDAR landmarks associated with each node. Afterwards, localization can be done by scan matching the current LiDAR data with the stored LiDAR landmarks to find which node the vehicle is currently in. If the nodes instead only represent intersections while edges represent tunnels, and the initial

vehicle position as well as heading are known, then it is possible to localize the vehicle in the topological map by detecting transitions to and from intersections. Gustaf Johansson and Mattias Wasteby thus proposed using artificial neural networks (ANN) and support vector machines (SVM) trained on 2D LiDAR data of intersections and tunnels to classify the current LiDAR data as either an intersection or a tunnel [6]. Johan Larsson et al. instead proposed detecting and classifying intersections by evaluating the range difference between two consecutive points in a 2D LiDAR scan to identify discontinuities in the tunnel walls [7]. While mainly aimed towards intersection detection in urban environments, Tongtong Chen et al. proposed an intersection detection and classification approach that uses 3D LiDAR data [8]. The approach entails first extracting points belonging to the road surface, plotting the distance values of these points versus the angle of their respective beams, and then identifying the distance peaks in that plot. These peaks correspond to "discontinuities" in the main road similar to how the aforementioned approach of Johan Larsson et al. identifies discontinuities in the tunnel walls. Aaron Morris et al. proposed yet another intersection detection approach that computes the Delaunay triangulation of a 2D LiDAR range scan to find triangles whose sides are longer than a set threshold [9]. The interpretation of these triangles is that a vehicle with a width less than this threshold can traverse those edges and it thus constitutes an intersection.

Critical systems often require redundancies to avoid failure. Johan Larsson et al. therefore proposed using a combination of different localization methods to improve the robustness of intersection detection for topological navigation [10]. The methods are:

1. dead reckoning of odometry data obtained from scan matching and wheel encoders to estimate when the vehicle is close to or has passed an intersection (some of the edges in the topological map can be labeled with an approximate distance to help the localization in difficult parts of the mine),
2. recognition of the laser signature of the intersection,
3. recognition of the topological structure of an intersection such as the number of side tunnels (similar to finding discontinuities in the tunnel walls as previously mentioned),
4. detection of RFID-tags placed in the intersection.

When it comes to designing an ADS, there are two main approaches: modular approaches where the various challenges are solved as sub-problems, and end-to-end driving approaches where the perceived environment is directly transformed into vehicle actuator inputs [11]. Modular approaches are the standard within industrial research as their intermediate representations makes them more interpretable and more streamlined to develop in parallel given a large team [12]. With a modular approach the ADS can be divided into sub-modules that may include, but are not limited to:

- **Sensors raw data:** the task of collecting environment data to be used in others tasks in the system.
- **Perception:** the task of interpreting sensor data to extract meaningful

features.

- **Mapping and planning:** mapping is the task of combining collected environment data to better understand the surroundings. Planning is the task of finding a valid path from the vehicle's current position to a navigation goal in the aforementioned map.
- **Localization:** the task of localizing and keeping track of the vehicle's global and local positions for the purpose of path following and obstacle avoidance. Global localization usually entails finding and keeping track of the position within the world such as the current road, city or intersection. Local localization, on the other hand, can encompass determining and monitoring the location relative to other vehicles and pedestrians.
- **Decision-Making:** the task of making correct decisions based on the traffic rules.
- **Control:** the task of generating lateral and longitudinal commands in order to have the vehicle follow a planned trajectory. These commands are directly transmitted to the vehicle's low-level hardware actuators making it move in the intended direction.

The end-to-end driving approach instead maps the sensor data directly to steering commands with the help of artificial neural networks. Mariusz Bojarski et al. proposed a convolutional neural network that directly maps raw camera data from a single front-facing camera to steering commands [13]. The system was quite successful with a minimum amount of training data required from humans. As the internal components optimize themselves to maximize the overall system performance, the end-to-end driving approach has the potential to surpass modular systems as these rely on human-optimized intermediate steps. These intermediate steps make the system more interpretable for humans, but do not necessarily guarantee optimal system performance.

1.2 Motivation and aim

As there is no GNSS coverage underground, many of the existing solutions for underground localization and navigation thus rely on infrastructure, e.g. RFID-tags, or sensor data matching in a constructed map of the environment, e.g. laser scan matching in a 2D LiDAR map. Both of these approaches are inflexible as changes to the environment, such as tunnels being closed off or new tunnels being made, require further infrastructure deployment or remapping of the environment to update the map. A navigation and localization system utilizing a topological map is however much simpler to maintain as changes to the environment can be easily added to the map in the form of new nodes and edges. The aim of this thesis is to therefore design, implement and verify an automated driving system (ADS) for confined environments, such as underground mines, that incorporates intersection detection and classification to navigate and localize based on a topological map of the environment.

1.2.1 Research questions

With the aforementioned aim in mind, the following research questions can be formulated:

- What are the aspects and challenges to consider when developing an ADS for an autonomous vehicle operating within a confined environment without GNSS coverage?
- What other intersection detection methods can be utilized for the navigation and localization task in a topological map and what are their limitations?
- Is it possible to apply an ADS designed for confined environments without GNSS coverage on an unconfined environment?

1.3 Scope and limitations

The automated driving system (ADS) development and testing will mainly be done within an office environment as there will not be enough time to verify the functionality in a real underground mine as well as tune any required parameters. An office is a confined environment similar to an underground mine as corridors represent tunnels and intersections are where these corridors meet, so it is considered an acceptable substitute. Furthermore, as the utilized office only has 90 degrees turns in intersections, and also once again for the purpose of saving time, the ADS will be designed to handle only 90 degrees turns. Unlike an underground mine, however, the utilized office has numerous glass walls which a LiDAR, for example, will not identify as obstacles. The ADS will therefore not be designed to handle transparent obstacles such as glass structures. The distance between intersections in the office is also much smaller than in an underground mine with some intersections being located only about 50 centimeters apart. This complicates

the identification of node and edge transitions so therefore such intersections will not be considered. Hence, the ADS will not be designed to handle intersections that are less than 50 centimeters apart. Additionally, to further simplify the problem the ADS will not be capable of reverse maneuvers, which would be required if a navigation goal would be set behind the vehicle. The longitudinal control will also be simplified to either stop or forward movement. Forward movement will be at a low constant speed to give the ADS enough time to process data and make decisions as well as minimize the risk of destructive collisions.

2
Theoretical background

In this section the fundamental theoretical prerequisites are given in order to get a comprehensive understanding of the concepts and technical terms brought up in this thesis.

2.1 Topological map

In contrast to displaying the exact geometric locations of an object in a space, a topological map only depicts the location based on the relationship between environmental features in an area. This can be particularly useful when precise geometric information is not necessary or when it is not available such as in an underground environment [14]. The topological map is also easier to maintain in such an environment that is continuously subject to change compared to other geometric approaches brought up in Section 1.1. A topological map is usually constructed using a graph where nodes represent distinguishable places while edges represent transitions or connections via them [15]. This means that localization in a topological map is limited to nodes so it is only possible to check if an object is at a specific node or not. A common example of a topological map is a metro map where nodes represent stations and edges represent the way to get to a certain station. In the context of robotics and autonomous vehicles, a topological map is commonly used for specific applications such as navigation and exploration in an area. In order to get a more precise depiction of the environment, a topological map is often used in conjunction with other localization and mapping techniques such as occupancy grids [16].

2.2 LiDAR

LiDAR, short for light detection and ranging, is a technology used for accurately representing the surrounding environment as a high resolution map consisting of points known as a point cloud. Each point in the point cloud contains the 3D coordinates and can be calculated with the help of a laser. The laser in a LiDAR is able to emit pulses of light to accurately measure the distance to a point by determining the time it took for the pulse to be reflected on an object and be returned back to the LiDAR's receiver [17]. Equation 2.1 can be used to calculate the distance to a point in the point cloud using a LiDAR where c is the speed of light and t is the total time for the pulse to be emitted and reflected back to the

receiver. Due to t being the total time, in order to only get the time to the object in one direction, the expression must be divided by two.

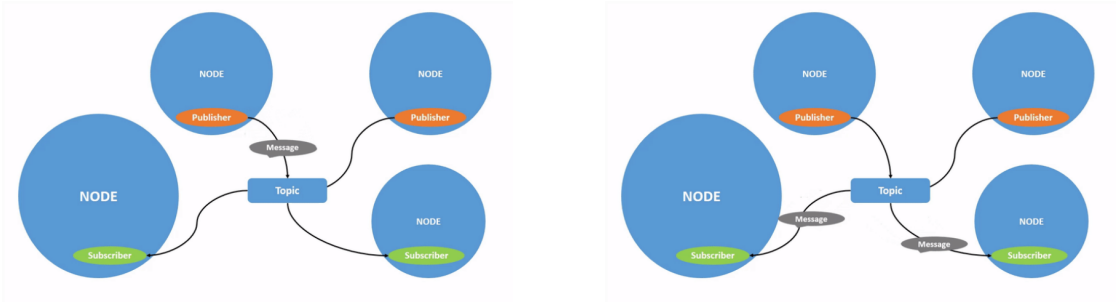
$$d = \frac{c \cdot t}{2} \tag{2.1}$$

The LiDAR's ability to map and scan both man-made and natural environments with a high degree of precision, flexibility and accuracy [18] makes it especially useful for autonomous vehicles. In order to develop a reliable navigation and localization system, an accurate representation of the surroundings is essential for an autonomous system. Even though modern LiDAR systems features a high degree of precision for objects up to 200 meters away [19], there still exists a *blind spot* for objects that are too close to the LiDAR [20]. This means that it cannot detect objects that are in close proximity which can cause inaccurate representation of the environment and therefore faulty behavior of the system. The performance of the LiDAR can also significantly be altered by the presence of glass in the current surroundings due to the transparent and reflective characteristics of surfaces such as glass. When laser pulses are emitted towards a piece of glass, most of them are passed through the glass or reflected away which means that the LiDAR is not able to accurately measure the distance to the glass itself but rather to the object directly behind [21]. Consequently, this leads to an incorrect representation of the environment that negatively impacts the autonomous system's ability to accurately visualize the surroundings.

2.3 ROS2

Robot Operating System (ROS) is an open source collection of tools and software for building robot applications [22]. It promotes modularity by separating code into nodes that communicate based on a publish-subscribe pattern. Nodes can publish data onto topics and other nodes can subscribe to these topics to receive all data published to those topics, which is illustrated in Figure 2.1. ROS also supports other communication methods such as services and actions. Services use a call-and-response pattern with client nodes that request services and server nodes that respond with responses based on the outcome of a service. Actions are similar to services but are intended for long running tasks as they additionally provide continuous feedback while a service executes and also the ability to cancel a service. ROS nodes can be written in either Python or C++ and if one node is written in Python and the other in C++, they can still communicate with each other through topics even though they are written in different languages. ROS also includes a number of useful tools for development, such as rosbag and RViz. Rosbag is a tool that allows the recording and playback of data published to topics. With rosbag, live data can be recorded during physical testing and then played back and carefully analysed as if running it live. This speeds up development and simplifies debugging as the same recorded data can be reused to verify code. RViz, on the other hand, is a 3D visualization tool that has built-in visualization support for most common data types. It helps interpret sensor data and confirm the behaviour of the code.

An example of RViz visualizing a camera feed and a LiDAR point cloud are shown in Figure 2.2.



(a) A message being published by a node to a topic.

(b) A message being received by nodes subscribed to a topic.

Figure 2.1: Publish-subscribe relationship in ROS2. Multiple nodes can publish to the same topic and multiple nodes can subscribe to the same topic. [22]

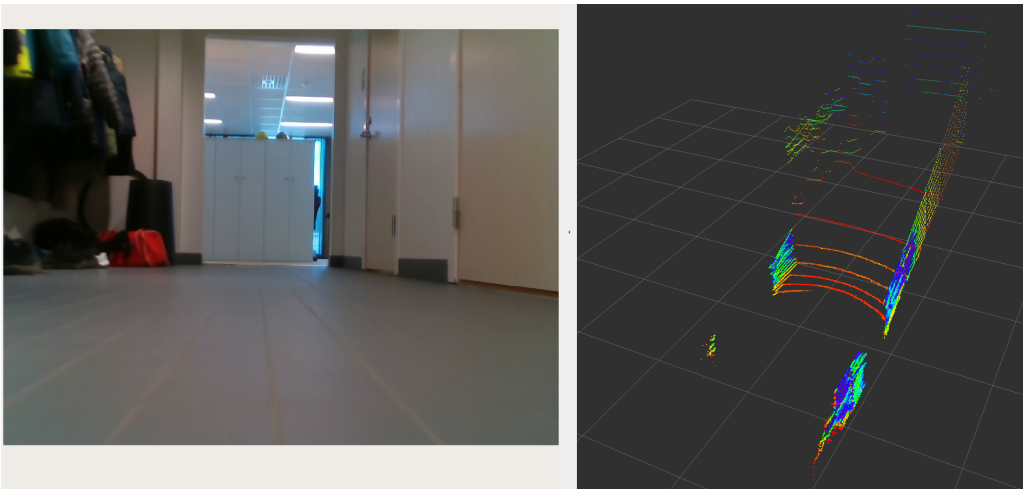


Figure 2.2: RViz visualizing a camera feed on the **left** and a LiDAR point cloud on the **right**.

2.4 Ground segmentation of LiDAR data

Segmenting LiDAR data into ground and non-ground points is essential for tasks such as traversable area detection and object recognition [23]. Furthermore, ground segmentation can be used as a pre-processing step for other segmentation stages such as classification of the points into static and dynamic points. Since ground points can be assumed to be static, this segmentation step therefore only needs to be performed on the non-ground points. With the ground points identified, it also becomes possible for an autonomous vehicle to plan and follow a drivable path while avoiding collisions with non-drivable points such as walls and

other objects in the way. In addition, for corridor-like environments, such as an underground mine, ground segmentation can help with the task of identifying and classifying intersections. Lee et al. proposed a ground segmentation method called Patchwork++ [23], an extension of their earlier Patchwork method. The flow chart of the Patchwork++ method can be seen in Figure 2.3.

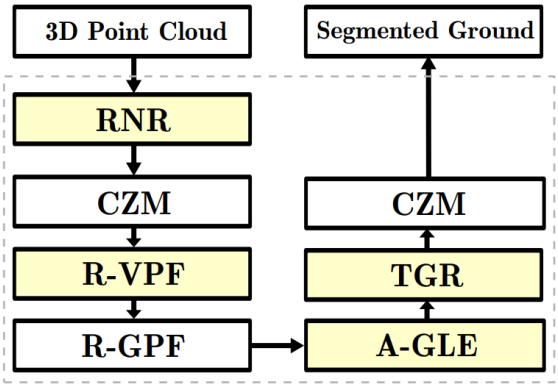


Figure 2.3: Flow chart of Patchwork++ [23]. © 2022 IEEE

The first step in Patchwork++ is **RNR: Reflected Noise Removal** [23]. Noisy reflected points can occur in LiDAR data if the laser beam is reflected before returning. It is assumed that the laser beam travelled in a straight line, so the registered point appears to be underground since it took longer for the beam to come back than the distance to the ground. Furthermore, this additional reflection also decreases the intensity of the noisy point. This phenomenon is illustrated in Figure 2.4. The most critical of these noisy points tend to occur near the LiDAR due to the smaller incident angle of the rays, which leads to lower height values. Therefore, some points near the LiDAR that have lower heights and intensities than set thresholds are removed.

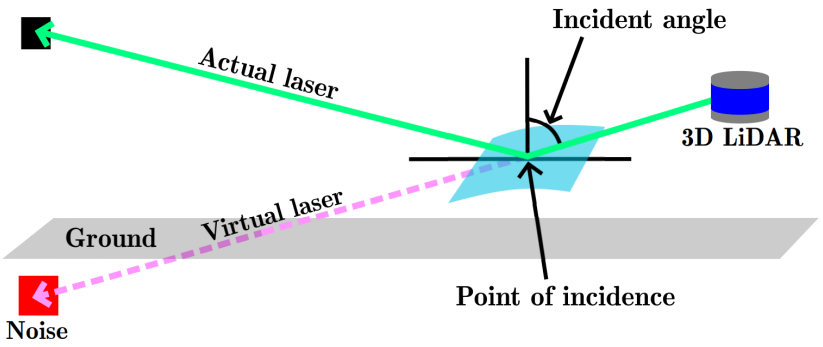


Figure 2.4: A noisy point that appears to be underground due to reflection. The laser beam is assumed to have travelled in a straight line, so it appears as if the registered point is underground due to the distance to the actual point being further than that to the ground [23]. © 2022 IEEE

The second stop in Patchwork++ is **CZM: Concentric Zone Model** [24]. The

observable world has varying degrees of flatness, so plane fitting is done region wise by dividing the data into different zones which are further subdivided into bins of varying sizes according to the proposed Concentric Zone Model, which is illustrated in Figure 2.5. The point cloud in the fourth zone is too sparse to accurately find the ground plane, so for that reason the bins in the fourth zone are made larger to make it easier to find the correct ground plane. Furthermore, when the bins become too small, the estimation of the correct normal vector to the ground plane can sometimes fail, so to address that the bins in the first zone are also made larger.

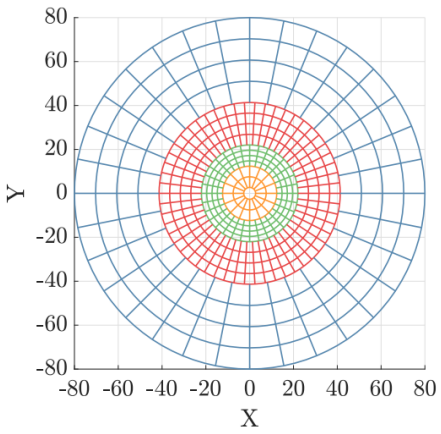


Figure 2.5: The proposed Concentric Zone Model of the Patchwork method that divides the data into 4 zones and then further subdivides the data within those zones into bins of varying sizes [24]. © 2022 IEEE

The third step in Patchwork++ is **R-VPF: Region-wise Vertical Plane Fitting** [23] which aims to find points that belong to vertical planes and reject them. Not rejecting them would lead to **R-GPF: Region-wise Ground Plane Fitting** choosing too low initial points for the ground plane fitting which would lead to incorrect ground segmentation results when the ground is on an elevated plane, as for example is in the case with stairs. Potential vertical plane points are estimated by selecting a number of the lowest points as initial points and fitting a plane to them. Additional points that lie within a certain distance margin of the plane are also added to the set of these potential vertical plane points. The potential vertical plane points are then finally classified as vertical plane points if the angle between the unit vector in the z-axis and the unit normal vector of the plane is within a set margin of $\frac{\pi}{2}$, i.e., the normal vector is almost perpendicular to the z-axis. This is done for a few iterations with points that haven't been classified as vertical plane points before. The final estimation of which points belong to a vertical plane is the union of all points that have previously been classified as vertical plane points during the multiple iterations.

The fourth step in Patchwork++ is **R-GPF: Region-wise Ground Plane Fitting** [24]. Given points in a bin, a number of points with the lowest z-values are chosen as initial ground points. Any points whose z value is less than the mean z value of those initial ground plus a height margin are also considered as ground

points. The normal vector for these ground points is then calculated by calculating the covariance matrix of these points and finding the eigenvector corresponding to the smallest eigenvalue. It is assumed that the variance should be the smallest in the direction of the normal vector to the ground plane, so therefore the eigenvector corresponding to the smallest eigenvalue is chosen, as illustrated in Figure 2.6. The general equation of a plane can be written as $a \cdot x + b \cdot y + c \cdot z = d$ where d is known as the plane coefficient. Using the previously found normal vector $\mathbf{n}$, the plane coefficient can be calculated as $d = -\mathbf{n}^T \bar{\mathbf{p}}$ where $\bar{\mathbf{p}}$ is the average point of the ground points. More points are then added to the estimated set of ground points if they lie within a certain distance margin of the estimated plane. The process is then repeated for the updated set of points: a new normal vector is calculated, a new plane coefficient is calculated and all points that lie within a certain distance margin of the newly estimated plane are added to the next iteration of ground points estimation. Because all points are considered, points can be added and removed depending on the distance to the newly estimated plane, so even the points that were initially estimated to be ground points can be removed later if they are too far from the estimated plane.

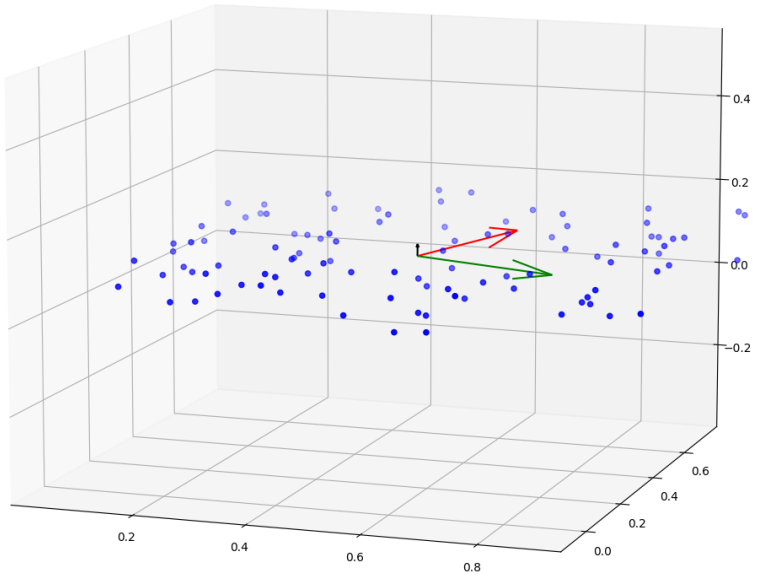


Figure 2.6: Randomly generated data points with a smaller variance in z. The three vectors are the eigenvectors of the covariance matrix scaled with their respective eigenvalues. As can be seen, the vector drawn in black has the smallest eigenvalue and is thus the most likely to represent the normal vector to the ground plane.

The fifth step in Patchwork++ is **A-GLE: Adaptive Ground Likelihood Estimation** which determines whether the region-wise estimated ground planes are ground or non-ground based on **uprightness**, **elevation** and **flatness** indicator functions [23]. **Uprightness** is defined as how orthogonal the normal vector of the estimated plane is to the X-Y plane. Any estimated plane whose angle between the normal vector and the X-Y plane is less than 45° is discarded.

Elevation assigns a lower probability of being ground to bins that are close to the sensor and whose average height value is above a certain threshold. This improves ground classification when the ground is not completely visible due to occlusion or partial observation which can occur when objects near the sensor prevent it from seeing the ground, like for example if there are other nearby cars. Without this, roofs of nearby cars could be classified as ground. An example of where elevation helps filter out false positives is illustrated in Figure 2.7. **Flatness** aims to revert some false negatives filtered out by **elevation** if they are definitely an even plane, like for example is the case with a very steep uphill. This is done by reverting any plane whose earlier calculated eigenvalue, that corresponds to the ground plane normal, is below a certain threshold. The optimal threshold values for **elevation** and **flatness**, as well as the height threshold used in **RNR: Reflected Noise Removal**, are however dependent on the environment and it would be time-consuming to set them every time for a new environment. To tackle this problem, the threshold values are updated based on the values of the previously estimated ground planes in each ring of the concentric zone model. The update is done according to $e_{\tau,m} \leftarrow \text{mean}(E_m) + a_m \cdot \text{stdev}(E_m)$ where $e_{\tau,m}$ is the threshold value for elevation in ring m, E_m is the set of all elevation values for the estimated ground planes in ring m and a_m is a constant gain of the standard deviation term in ring m. Flatness is updated in a similar manner, but with a separate standard deviation constant gain term. The height threshold for reflected noise removal is updated according to $h_{\text{noise}} \leftarrow \text{mean}(E_1) + \delta$ where h_{noise} is the height threshold, E_1 is the set of all elevation values for the estimated ground planes in the **first ring** of the concentric zone model and δ is the distance margin of the noise removal. This update flow is illustrated in Figure 2.8.

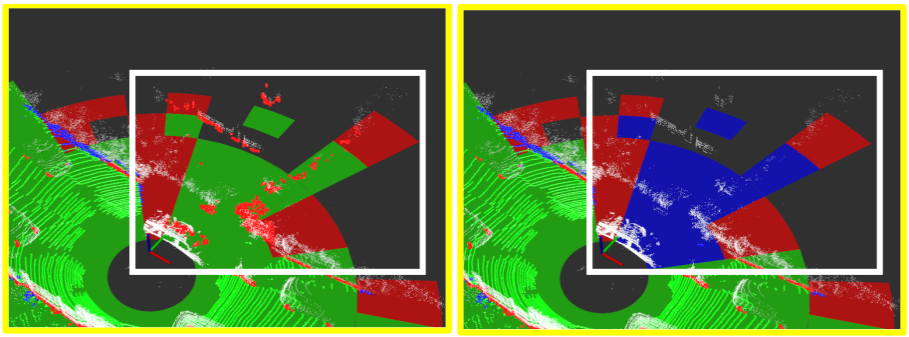


Figure 2.7: Left image: false positive red points above the car, as well as behind the car, that have been falsely classified as ground points since they meet the **uprightness** criteria. **Right image:** The false positive red points have been correctly filtered by the **elevation** criteria as they lie close to the sensor and have heights higher than the threshold [24]. © 2022 IEEE

The sixth and the last step in Patchwork++ is **TGR: Temporal Ground Revert** [23]. While A-GLE has the advantage of generalizing better for new environments, it also has the side effect of acting as a low pass filter due to the values being updated over time. This makes it difficult to correctly estimate certain bins as

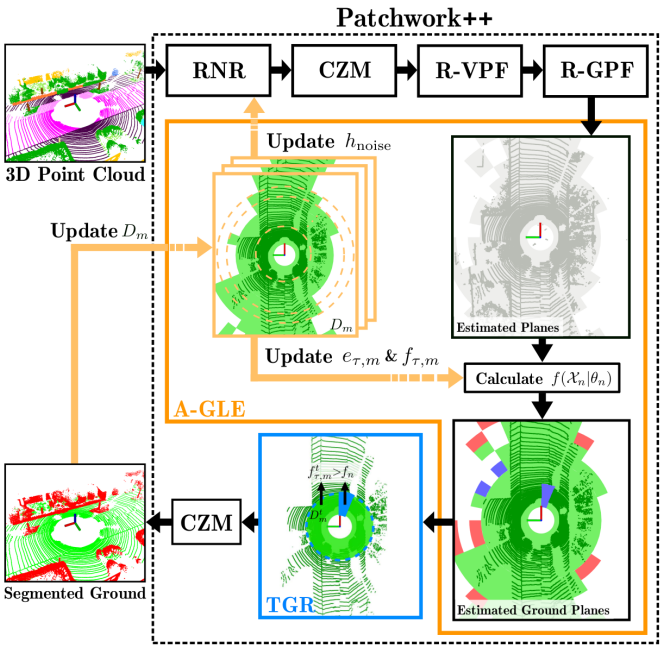


Figure 2.8: Update flow of A-GLE: Adaptive Ground Likelihood Estimation uses to adaptively adjust the threshold parameters for elevation, flatness and the height threshold for RNR: Reflected Noise Removal [23]. The variables $e_{\tau,m}$, $f_{\tau,m}$ and h_{noise} correspond to the elevation threshold for ring m of the concentric zone model, the flatness threshold for ring m and the height threshold for RNR: Reflected Noise Removal respectively. $f(X_n|\theta_n)$ is the density function for ground likehood estimation, X_n is a random variable in bin n and θ_n is the uprightness, elevation and flatness parameters in bin n. D_m is the previously estimated ground planes in ring m. © 2022 IEEE

ground if the flatness value temporarily becomes large, which could be the case in rough terrain when there are for example sudden patches of grass. To address this issue the current flatness values are also compared to earlier flatness values and the corresponding bins are classified as ground if the values don't differ too much.

2.5 Occupancy grid

An occupancy grid is a data structure commonly used in autonomous vehicles and robotic applications [25] to represent the surrounding environment as a discrete grid consisting of cells that depicts the state of a small area of the environment. Each cell is often binary where 0 represents that the area is not occupied while 1 represents an occupied area [26]. The characteristics of an occupancy grid is fundamental when trying to implement an algorithm to find the shortest-path such as Dijkstra's algorithm or the A* algorithm. This is due to the ability to represent the occupancy grid as a weighted graph where the cells can be represented as nodes and the neighbouring cells can be represented as adjacent nodes that edges connect to. The weights of each edge in the case of the occupancy grid is determined by the

state of the two connecting cells such that the weight is high if the connected cell is occupied and lower if the cell is unoccupied [27].

2.6 Dijkstra's algorithm

Dijkstra's algorithm is used to efficiently generate the shortest path given a starting node and a goal node in a weighted graph. Each edge in the graph is assigned a weight which is often based on the distance between the two connecting nodes and once every edge is weighted, the algorithm's main purpose is to find the minimum-cost path between the two given nodes [28]. The algorithm works by retaining a list of unvisited nodes and their approximate distances from the source node while iteratively increasing a set of visited nodes. The first step in the algorithm is to set the distance to all unvisited nodes in the graph to infinity while setting the start node to zero. The algorithm then iteratively chooses the node from the set of unvisited nodes with the shortest tentative distance with respect to the start node and adds it to the collection of visited nodes [29].

For each visited node, the Dijkstra's algorithm calculates the total distance from the starting point to each of its unvisited neighbors. It does this by adding the distance from the start to the visited node and the distance from the visited node to each of its unvisited neighbors. If the new calculated distance is shorter than the current distance to a neighbour, the neighbour is updated with the new distance so that only the smallest distance is kept. This process continues until the distance to all the neighbours of the current visited node is calculated and updated accordingly [30]. The algorithm then proceeds with the next unvisited node that has the shortest tentative distance and explores the neighbours as previously explained. When every unvisited node is explored or the destination node is reached, the algorithm terminates and outputs the set of visited nodes with its corresponding distance from the starting node. The minimum-cost path can then be constructed based on the output which represents the shortest path between two nodes in the weighted graph [31]. The shortest path generated by Dijkstra's algorithm can be used in a variety of application such as in finding the most optimal way between two nodes in a topological map.

2.6.1 A* search algorithm

The A* algorithm builds upon Dijkstra's algorithm but in order to make the search more effective, A* also includes a heuristic function to explore the most probabilistic paths first. The underlying mechanics of A* is similar to those in Dijkstra's algorithm, however, A* implements a real time estimation of the cost to reach the goal node given the current node [32]. With the help of a priority queue, which is sorted by the predicted cost to the goal, the next node to explore is selected based on the lowest estimated cost which is calculated based on the heuristic function. The heuristic function used in A* can vary but the main underlying principle is that it is admissible and consistent meaning that it never overestimates the distance to the end node which always returns the optimal path.

The cost estimate to minimize each iterative loop can be represented as follows [33]:

$$f(n) = g(n) + h(n)$$

Where n is the current node, $f(n)$ is the estimated additive cost for the current node based on $g(n)$, which is the cost to reach the current node from the start node, and $h(n)$, which is the heuristic function that predicts the remaining cost to reach the goal node.

2.7 Odometry

Odometry is the process of estimating the change in a robot's or vehicle's rotation and translation relative to some known initial position using sensors [34]. Some common forms of odometry include:

- **Wheel odometry** involves counting the number of wheel rotations of wheels in contact with the ground with the help of encoders to estimate the linear displacement.
- **Inertial Navigation Systems (INS)** involve integrating data from accelerometers and gyroscopes to estimate the change in velocity, position and orientation.
- **Global Navigation Satellite System (GNSS)** involves using the man-made satellites in orbit around Earth to determine the absolute position using trilateration and the travel time of the satellites' signals.
- **Sonar or ultrasonic sensors** send out ultrasonic signals and then measure the time it takes for the signals to come back after they are reflected off an object. Knowing the time from transmission to reception, the distance to the object the signal reflected off can be calculated. By creating an array of the aforementioned sensors, the change in rotation and translation can be estimated through model matching and triangulation by comparing the sensor data at different times.
- **Visual odometry** involves using a stream of images from one or multiple cameras to estimate the change in position and orientation by integrating the pixel displacements between image frames.
- **LiDAR odometry** involves estimating the change in translation and orientation by calculating the transformation that aligns two point clouds generated by consecutive LiDAR scans, commonly known as scan matching [35]. A common scan matching algorithm is the iterative closest point (ICP) algorithm.

2.7.1 Iterative Closest Point (ICP) algorithm

The iterative closest point algorithm (ICP) is an iterative method of finding the optimal translation and rotation between two sets of points. If used continuously in a vehicle equipped with a LiDAR, the LiDAR scans can be continuously compared to estimate the vehicle's translation, orientation and velocity as

illustrated in Figure 2.9.

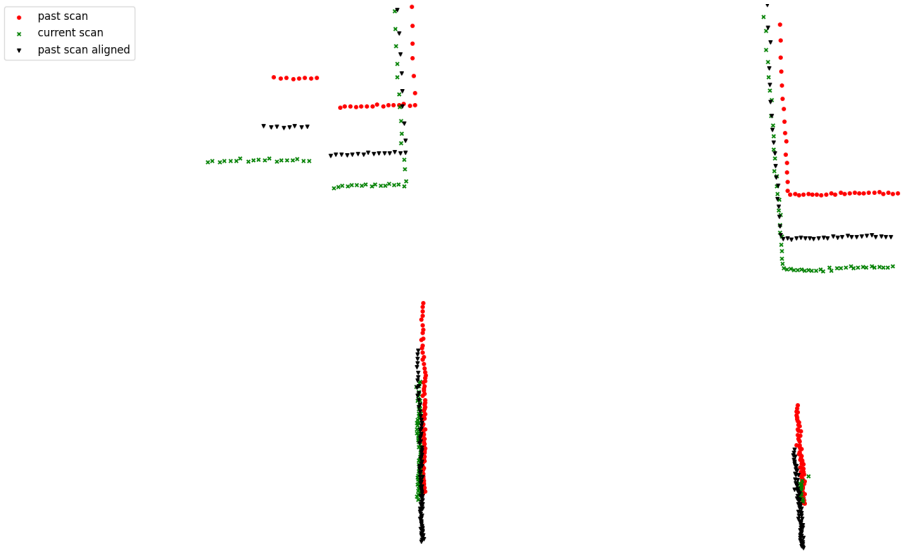


Figure 2.9: Top view of a vehicle driving straight through a hallway. Red circles are the past scan and green crosses are the current scan. The black triangles are the past scan aligned to the current scan using the transformation obtained from ICP. A perfect transformation would align the previous scan on top of the current scan. In other words, the black triangles should be directly on top of the green crosses. This is however not the case, but it still manages to capture the approximate displacement.

The ICP algorithm works as follows [36]: given a source set of points P consisting of n points and a target set of points X, initialize the iteration by setting $P_0 = P$ and iteration index $k = 0$. Thereafter, iterate until the difference in mean-squared error between iterations is less than a set threshold of τ:

1. For every point in P_k, find the closest point in X to create Y_k such that the point $\vec{y}_{k,i} \in Y_k$ minimizes the Euclidean distance to point $\vec{p}_{k,i} \in P_k$, i.e. $\vec{y}_{k,i} = \min_{\vec{x} \in X} \|\vec{x} - \vec{p}_{k,i}\| \ \forall i \in 1, \ldots, n$.
2. Calculate the optimal transformation between P_0 and Y_k. The optimal transformation is found by minimizing the mean square function $f(\vec{q}_k) = \frac{1}{n} \sum_{i=1}^{n} \|\vec{y}_{k,i} - \mathbf{R}(\vec{q}_{k,R})\vec{p}_{0,i} - \vec{q}_{k,T}\|^2$ where:

   - $\vec{q}_k = [\vec{q}_{k,R} | \vec{q}_{k,T}]^T$ is the complete transformation vector for iteration k,
   - $\vec{q}_{k,R} = [q_{k,0} q_{k,1} q_{k,2} q_{k,3}]^T$ is the unit quaternion vector representing rotation where $q_{k,0} \geq 0$ and $q_{k,0}^2 + q_{k,1}^2 + q_{k,2}^2 + q_{k,3}^2 = 1$ for iteration k,
   - $\mathbf{R}(\vec{q}_{k,R})$ is the 3×3 rotation matrix generated by unit rotation quaternion $\vec{q}_{k,R}$ for iteration k. Without taking the iteration number into account, it is generally defined as:

$$\mathbf{R}(\vec{q}_R) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix}$$

   - $\vec{q}_{k,T} = [q_{k,4} q_{k,5} q_{k,6}]^T$ is the translation vector for iteration k,

- $\vec{q}$ is initialized to $\vec{q}_0 = [1, 0, 0, 0, 0, 0, 0]^T$ for iteration $k = 0$.

The optimal rotation is found by first constructing the cross-variance matrix Σ_{py} of P_0 and Y_k. Without taking the iteration number into account, it is generally defined as $\Sigma_{py} = \frac{1}{n}\sum_{i=1}^{n}[(\vec{p}_i - \vec{\mu}_p)(\vec{y}_i - \vec{\mu}_y)^T]$ where $\vec{\mu}_p$ and $\vec{\mu}_y$ are the centroids of P and Y respectively. The column vector $\Delta = [A_{23}A_{31}A_{12}]^T$ is then formed from the matrix $A = (\Sigma_{py} - \Sigma_{py}^T)$. This vector, together with the trace of Σ_{py} and the 3×3 identity matrix $\mathbf{I_3}$ form the matrix $Q(\Sigma_{py})$:

$$Q(\Sigma_{py}) = \begin{bmatrix} \mathrm{tr}(\Sigma_{py}) & \Delta^T \\ \Delta & \Sigma_{py} + \Sigma_{py}^T - \mathrm{tr}(\Sigma_{py})\mathbf{I_3} \end{bmatrix}$$

The unit eigenvector $\vec{q}_R = [q_0 q_1 q_2 q_3]$ corresponding to the largest eigenvalue of the matrix $Q(\Sigma_{py})$ is chosen as the optimal rotation, whereas the optimal translation is calculated as $\vec{q}_T = \vec{\mu}_y - \mathbf{R}(\vec{q}_R)\vec{\mu}_p$.

3. Apply the transformation to create the set of source points for the next iteration: $P_{k+1} = \vec{q}_k(P_0)$.
4. Terminate the iteration if the change in mean-squared error is less than τ, i.e. $d_k - d_{k+1} < \tau$ where $d_k = \frac{1}{n}\sum_{i=1}^{n}\|\vec{y}_{k,i} - \vec{p}_{k+1,i}\|$

2.8 Road edge detection

Autonomous vehicles need to stay on the road and as such the road edges need to be identified and extracted from the sensor data. In 2008, Johan Larsson et al. proposed to use Hough line transform on 2D LiDAR range scan data to identify lines and then filter line pairs to find which ones correspond to the road edges [37]. The line pairs were filtered based on their lengths, how parallel they were to each other, whether they were far enough apart from each other (lines closer together than the expected corridor width would not be favoured), and how much their directions differed from the vehicle's travelling direction (lines in the travelling direction of the vehicle would be favoured). In 2021, Saglam and Papelis used a similar approach to identify corridor walls in indoor environments using a single depth camera [38]. More recently in January of 2023, Yuanjian Jiang et al. proposed a path planning method based on 2D LiDAR range scan data that does not rely on identifying the road edges [39]. Instead, the LiDAR data is converted into a binary image, skeletonized and then a path is extracted from the skeleton. While comparable to the Hough line transform approach when travelling straight, this method is also able to handle intersection turns. Hough line transform approach is unable to handle turns as usually the LiDAR cannot see far enough into the intersecting corridor to identify that corridor's road edges.

2.8.1 Hough line transform

A line in the Cartesian coordinate system can be represented as $y = kx + m$ [40]. In the polar coordinate system, a line can be rewritten on the form $y = (-\frac{\cos\theta}{\sin\theta})x + (\frac{\rho}{\sin\theta})$

where ρ is the normal distance from $(0, 0)$ to the line and θ is the angle between the x-axis and the normal vector from $(0, 0)$ to the line, as illustrated in Figure 2.10a. Solving for ρ, we get $\rho = x \cos\theta + y \sin\theta$ meaning that for each point (x_0, y_0) we can represent the family of lines going through the point as $\rho_\theta = x_0 \cos\theta + y_0 \sin\theta$. Plotting the family of lines for a given point with θ on the horizontal axis and ρ on the vertical axis we get a sinusoid as illustrated in Figure 2.10c. Plotting the family of lines for multiple points, a line that fits multiple points can be found by identifying where several sinusoids intersect. Implementations of this transform often allow choosing the resolution of θ and ρ to create a 2D array where each cell represents a different line. Each cell in the array then accumulates a number of votes based on how many points lie on that line. The cell with the most votes is the line that goes through the most points. Often, a threshold parameter is also set to return all lines that get more votes than that threshold.

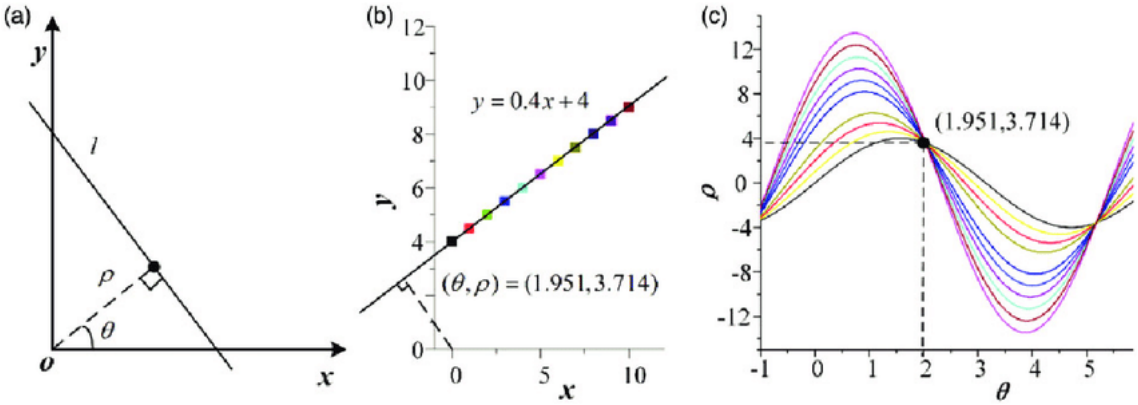


Figure 2.10: Using Hough line transform to find a best fit line. **Figure (a):** polar representation of a line using the ρ and θ parameters where ρ is the normal distance from $(0, 0)$ to the line l and θ is the angle between the x-axis and the normal vector from $(0, 0)$ to the line l. **Figure (b):** linearly distributed points on line $y = 0.4x + 4$ which in the polar coordinate system is represented as $(\theta, \rho) = (1.951, 3.714)$. **Figure (c):** different lines going through each point with the colours corresponding to the colour of the points in Figure (b). The sinusoids intersect at $(\theta, \rho) = (1.951, 3.714)$, which is correctly identified as the line $y = 0.4x + 4$ [41].

2.8.2 Progressive probabilistic Hough line transform

The progressive probabilistic Hough line transform aims to speed up the algorithm by randomly sampling points without replacement, voting on the lines that go through a sampled point and then as soon as a line exceeds a voting threshold, removing all other points that lie on that line [42]. All votes from points on that line are then subsequently removed meaning that each point can only vote for one line. The algorithm outline for finding lines in a binary image where points are represented as pixels is as follows:

1. Check the input image. If it is empty, then the algorithm is done.
2. Update the vote accumulator with a randomly sampled pixel from the image.

3. Remove the selected pixel from the image so that it cannot be sampled again.
4. Check if any of the lines voted by the pixel have exceeded the set vote threshold. If not, go to 1.
5. Remove any other pixels from the image that lie on those lines so that they cannot be sampled.
6. Remove all votes from the accumulator cast by pixels belonging to those lines.
7. If the lines are longer than a set minimum length, add them to the output list.

3

Methods

The following section gives a detailed explanation of the technical approaches used to achieve the results in this thesis where both hardware and software solutions are presented.

3.1 Hardware

In order to collect data and evaluate solutions, a development platform was needed. A Traxxas RC car was therefore equipped with an Intel RealSense D435 RGB-D camera, a Velodyne VLP-16 LiDAR, an MPU9250/6500 IMU and an Arduino. The vehicle before and after can be seen in Figure 3.1, and a close up of the vehicle's circuitry can be seen in Figure 3.2. The intention was to use the camera for image processing, such as road segmentation, and the IMU for odometry, but in the end they did not end up getting used. The Arduino listens for throttle and steering commands and actuates the vehicle based on those.

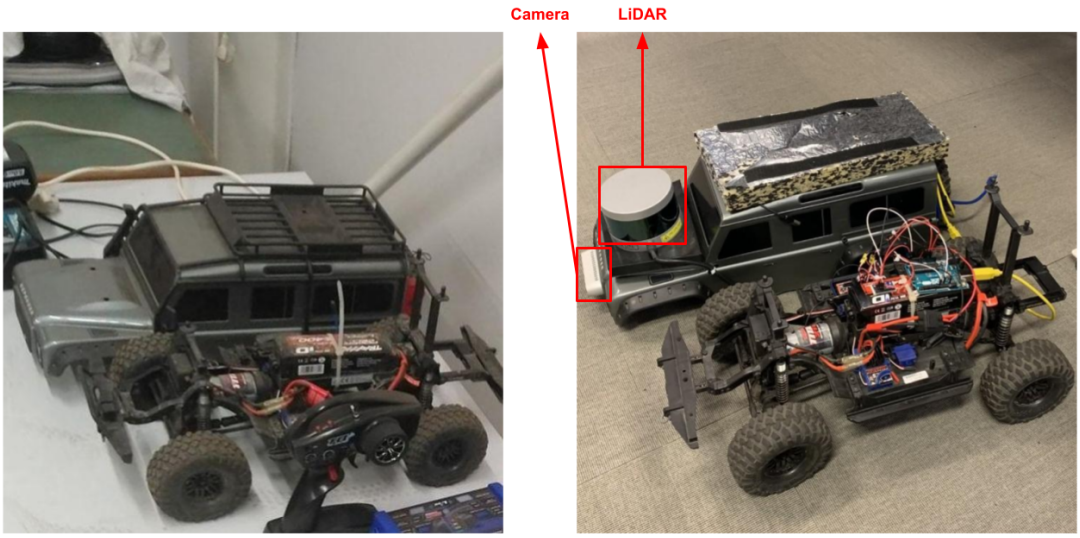


Figure 3.1: Traxxas RC car before and after being equipped with sensors and Arduino on the left and right respectively.

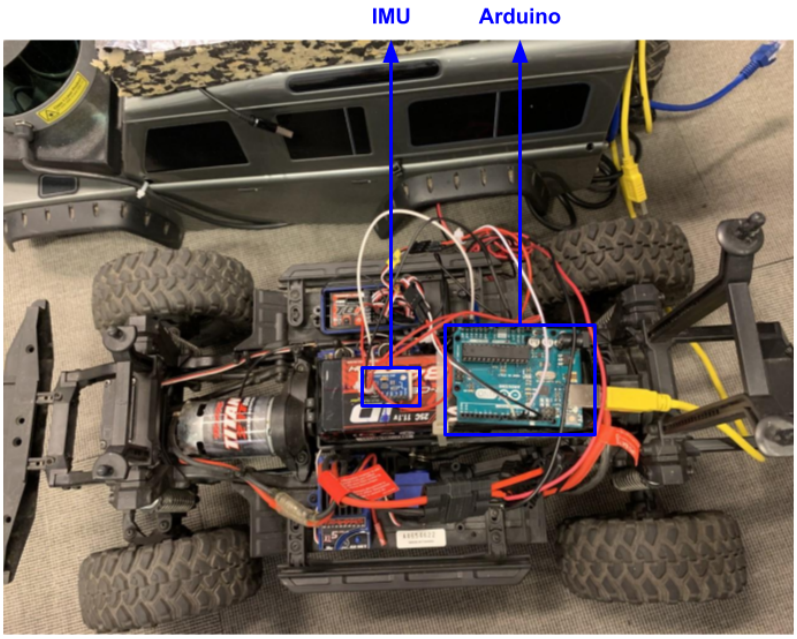


Figure 3.2: A close up of the circuity with IMU and Arduino marked.

3.2 Automated Driving System

The Automated Driving System (ADS) is constructed using several sub-modules called *nodes* where each *node* is responsible for a certain part of the complex autonomous driving task. The ADS serves as the backbone of the entire system and represents how different nodes communicate with each other through the publish-subscribe pattern in ROS2. In order to make the ADS as clear and interpretable as possible, the system is divided into three main areas that constitute the fundamental components of the ADS. These are the perception block, global navigation block and local navigation block that each contain nodes associated with the corresponding block. The perception block is responsible for extracting high level features from raw LiDAR data, the global navigation block handles tasks on a broader and wider scale while the local navigation block is confined to only handling tasks in its current surroundings. An overview of how the different components of the ADS interact with each other can be seen in Figure 3.3 where the ellipses represent the topics that have been published to by the previous block and that are available for the next block to use. There is also a stand alone GUI node that operates in parallel to the other blocks and acts as the human-machine interface, enabling the user to control the system and receive real time updates of the vehicles status.

The following sections will go into more detail on the inner workings of each block and their respective nodes.

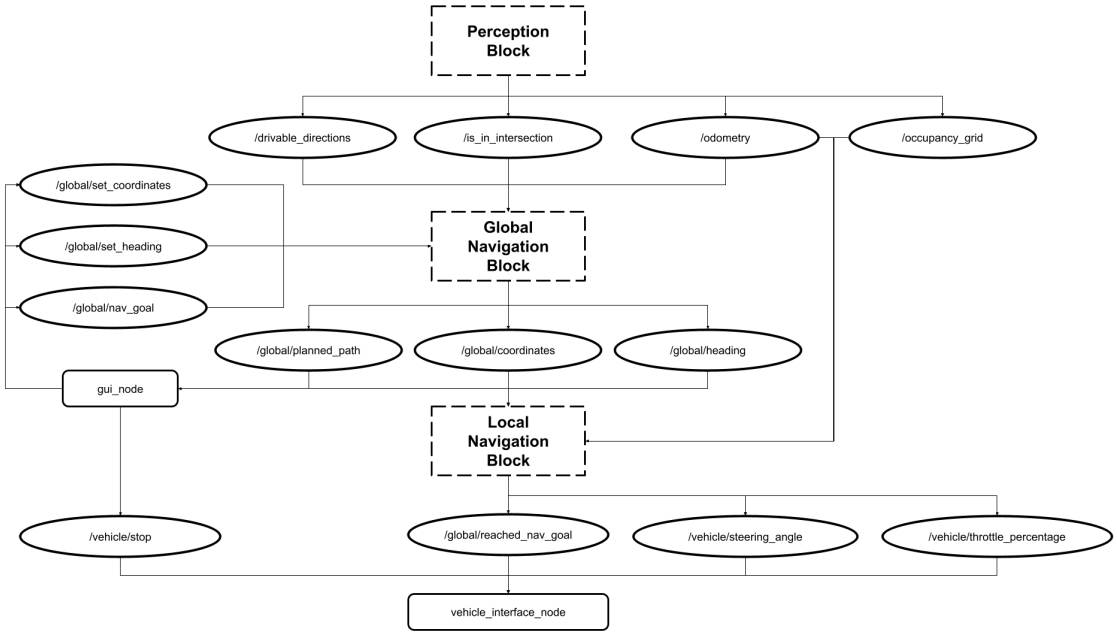


Figure 3.3: A data flow diagram of how different components interact with each other in the ADS. Ellipses correspond to topics and rectangles correspond to nodes.

3.2.1 Perception block

The perception block is responsible for extracting high level features from raw LiDAR data. The other parts of the stack then utilize these features to keep track of the vehicle's global and local position, plan navigation on a global and local scale, detect and classify intersections, as well as follow the globally and locally planned paths. This is done by extracting the following four features from the LiDAR data:

1. **Ground and non-ground points:** in order to avoid collisions, plan a path locally and detect intersections, the point cloud given by the LiDAR needs to be segmented into ground and non-ground points.
2. **Occupancy grid:** after the data has been segmented it needs to be discretized into a grid representation as the majority of path search algorithms, such as Dijkstra's and A*, require a discrete graph to work.
3. **Drivable directions:** in order to detect and classify intersections it is required to know which directions are traversable by the vehicle.
4. **Odometry:** in order to be able to follow a locally planned path, odometry data consisting of the vehicle's current position, heading, as well as linear and angular velocities is required. This data is extracted from the LiDAR data as the IMU data was too inaccurate.
5. **Is in intersection:** due to many nodes and processes being dependant on knowing when the vehicle is currently in an intersection, there needs to be a reliable node that can detect and publish this.

A schematic overview of the interaction between nodes and topics in the perception block can be seen in Figure 3.4 where the arrows indicate the direction of the data flow.

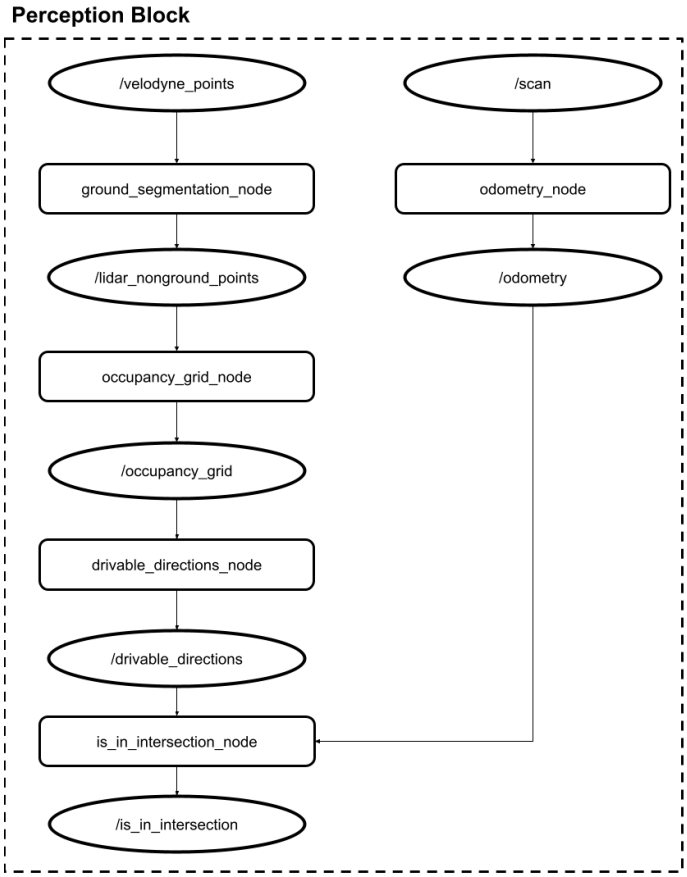


Figure 3.4: A visual representation of the interactions between nodes and topics in the perception block where ellipses correspond to topics and rectangles correspond to nodes.

3.2.1.1 Ground segmentation

The ground segmentation algorithm used is Patchwork++ [24] [23]. The open source Patchwork++ code [43] was used to incorporate the ground segmentation into the ADS. Example segmentation results of a hallway are illustrated in Figure 3.5.

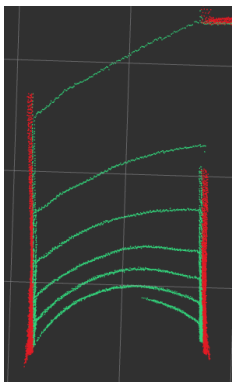


Figure 3.5: Ground segmentation of a hallway viewed from above with green and red points representing ground and non-ground points respectively.

3.2.1.2 Occupancy grid

In order to use the A* search algorithm, a discretized occupancy grid was needed to be generated based on the surrounding environment. The grid has the dimensions of 10x6 meters and is divided into cells with size of 0.2x0.2 meters meaning that the grid consists of 1500 individual cells. With the help of the segmented non-ground points from the LiDAR data, the exact location of obstacles such as walls could be extracted and placed in the corresponding cell of the grid. A padding around the obstacles was added in order to prevent the vehicle from driving too close to the obstacles and colliding with them. To simplify the adding of padding, the resolution of the grid was set so that one cell would be enough padding, which means that each obstacle has an extra 0.2 m of extra space around it. This padding size was calculated based on the width of the car which ensured that it kept a safe distance to the walls and that it avoided paths that were too narrow for it to pass through. A visualization of the complete occupancy grid can be seen in Figure 3.6.

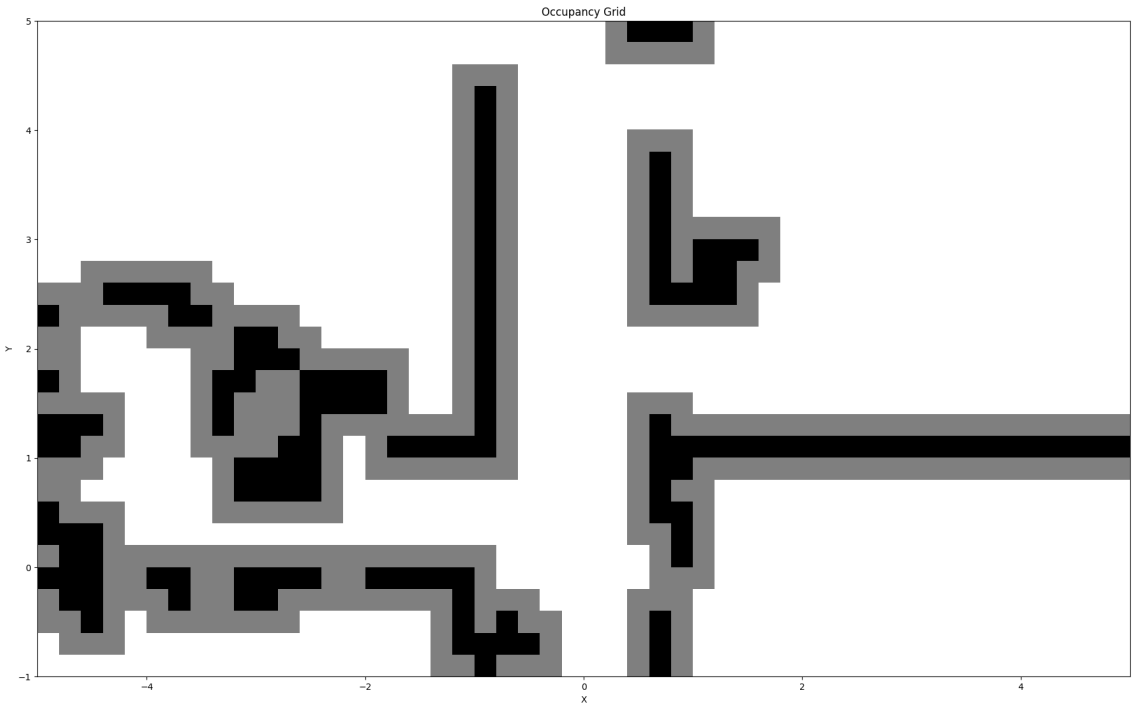
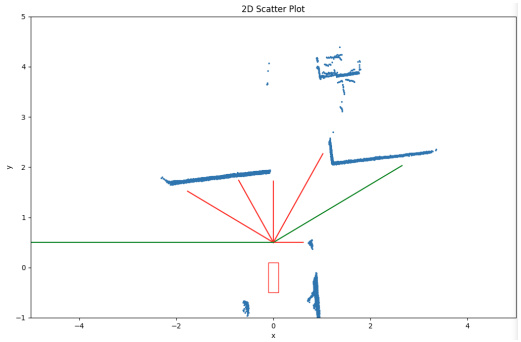


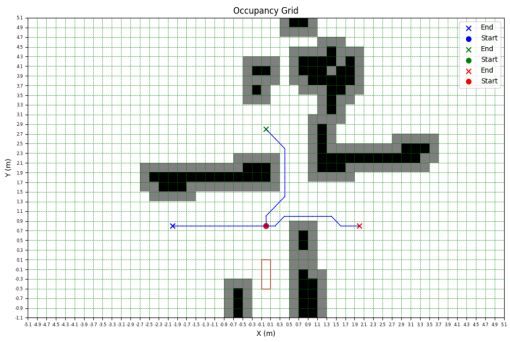
Figure 3.6: A top-down view of the occupancy grid where white cells indicate drivable space, black cells represents non-drivable space and gray cells is the padding corresponding to the non-drivable space.

3.2.1.3 Drivable directions

When navigating based on a topological map, one fundamental prerequisite for reliable localization is that an accurate intersection detection algorithm is implemented that can detect drivable directions. In order to get a robust solution that can detect 90° intersections (based on the limitations) with high accuracy, it is not sufficient to just check for straight paths round the vehicle (see Figure 3.7a). A more reliable approach is to use A* to generate paths to points with 90° increments in the vehicle's field of view (see Figure 3.7b).



(a) When only checking for straight paths, the system is prone to missclassifying the intersection type that it has encountered. Here the color of the lines represents whether it is a free path or not, given by the length of the line before encountering an obstacle. If the line is $> 2.5m$ it is defined as a free path and if it is $< 2.5m$ the lines becomes red indicating no free path. In the figure above it fails to identify a 4-way intersection.

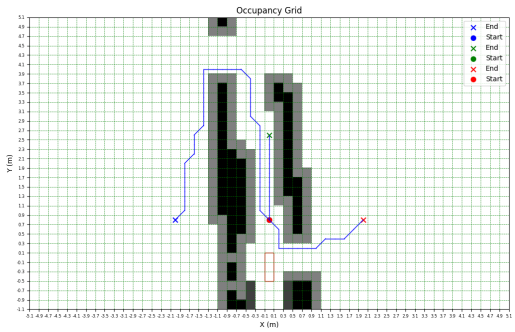


(b) When using A* to identify the same intersection, the paths that the algorithm generates are able to navigate around obstacles which more accurately represent the free paths in the current intersection. A 4-way intersection is thus successfully identified.

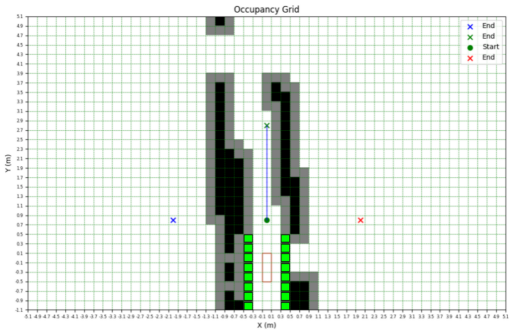
Figure 3.7: Comparison between two drivable directions algorithms in a 4-way intersection where the left subfigure only checks for straight paths while the right subfigure uses A* to find drivable directions.

If a path is successfully generated to a point while using A*, a drivable path is detected in that direction. If more than two successful paths are generated, then an intersection is identified. When doing so with the unmodified occupancy grid, A* sometimes generates an unrealistic path to a point either through the LiDAR's blind spot or around the wall which can be seen in Figure 3.8a. In order to fix this, artificial walls were added to the occupancy grid that prevent impossible paths from being generated. The two vertical lines in Figure 3.8b prevents paths from being generated through the blind spot. To avoid paths from being drawn around the walls, a software solution was implemented. The software solution prevents paths

from being 30% longer than the original Manhattan distance which is measured from start point to end points in Figure 3.7b.



(a) Without the added artificial walls, A* generates unachievable paths around walls and through the blind spot. This leads to multiple paths being generated which wrongfully classifies an intersection.



(b) The green cells represent the added artificial walls which together with the software solution prevent A* from generating improper paths to the end points marked with an X. This contributes to a more reliable intersection detection algorithm which ensures that a real intersection is found when paths are drawn to either of the two points on left and right side.

Figure 3.8: In order to generate proper drivable directions, a software solution and artificial walls were added as shown in the right subfigure. Otherwise the A* algorithm generates unachieveable paths as in the left subfigure.

3.2.1.4 Odometry

Initially, angular velocity and linear acceleration data from an IMU was used to estimate the vehicle's current position, heading as well as linear and angular velocities. While angular velocity was quite accurate, linear acceleration was not accurate enough to reliably estimate the vehicle's velocity and position. Instead, LiDAR odometry was utilized by using the ICP (Iterative Closest Point) algorithm to compare the current point cloud with the previous point cloud and calculate the transform between them. The transform gave the translation in x and y as well as the rotation around the z-axis between the point clouds. These were integrated together with the past values to keep track of the vehicle's x and y coordinates as well as the rotation around the z-axis (heading). Furthermore, velocities were calculated by dividing the acquired values from the transform by the time between the point clouds. Figure 3.9 shows an example of integrating the ICP results to acquire a travelled path. To reduce the computational complexity, the ICP algorithm was only run on the point cloud at $z \approx 0$.

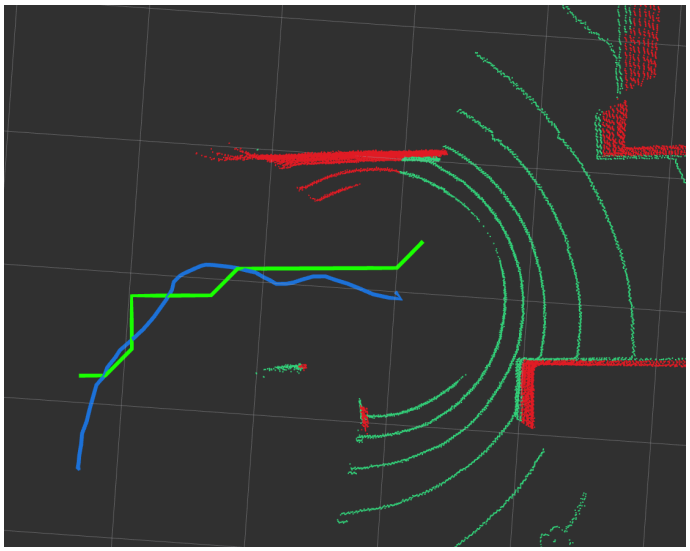


Figure 3.9: Top view of the odometry using the ICP algorithm when making a right turn. The blue path is the travelled path acquired by integrating the displacement from the ICP algorithm. Green path is the planned path. As with many odometry methods, this method is prone to cumulative errors.

3.2.1.5 Is in intersection

Since several nodes are dependant on knowing when the vehicle is in an intersection, there needs to be an accurate and reliable algorithm that can detect when an intersection starts and ends. In order to achieve this, the *is in intersection* node was implemented which receives real time data from the drivable directions node and publishes a boolean describing whether the vehicle is currently in an intersection or not. Once two or more paths are free according to the drivable directions node, an intersection has been identified and the *is in intersection* node publishes **True**. For the boolean to be set to **False**, the drivable directions node needs to detect no traversable paths to the right and to the left of the vehicle for at least 0.5 meters, meaning that the vehicle has exited the intersection and is currently in an edge.

3.2.2 Global navigation block

The global navigation block is responsible for planning a global path on the topological map, keeping track of the vehicle's position and heading, taking input from the user and providing feedback back to the user. This is done as five separate sub-tasks:

1. **Graphical user interface** allows the user to enter the vehicle's initial position and heading as well as set a global navigation goal. While the vehicle travels to its goal, the GUI also displays the vehicle's current position and heading.
2. **Global path planning** calculates the shortest path from the vehicle's starting position to the global navigation goal provided by the user.
3. **Global coordinates** updates the vehicle's global position when an intersection is detected. The new position is dependent on the previous

position as well as the global heading.

4. **Global heading** keeps track of the vehicle's global heading by integrating the heading change while the vehicle is in an intersection.

A schematic visualization of how different nodes and topics interact with each other as part of the global navigation block can be seen in Figure 3.10. Keep in mind the graphical user interface is not officially part of the global navigation block but it is essential either way to explain it in order to fully understand the global path planning.

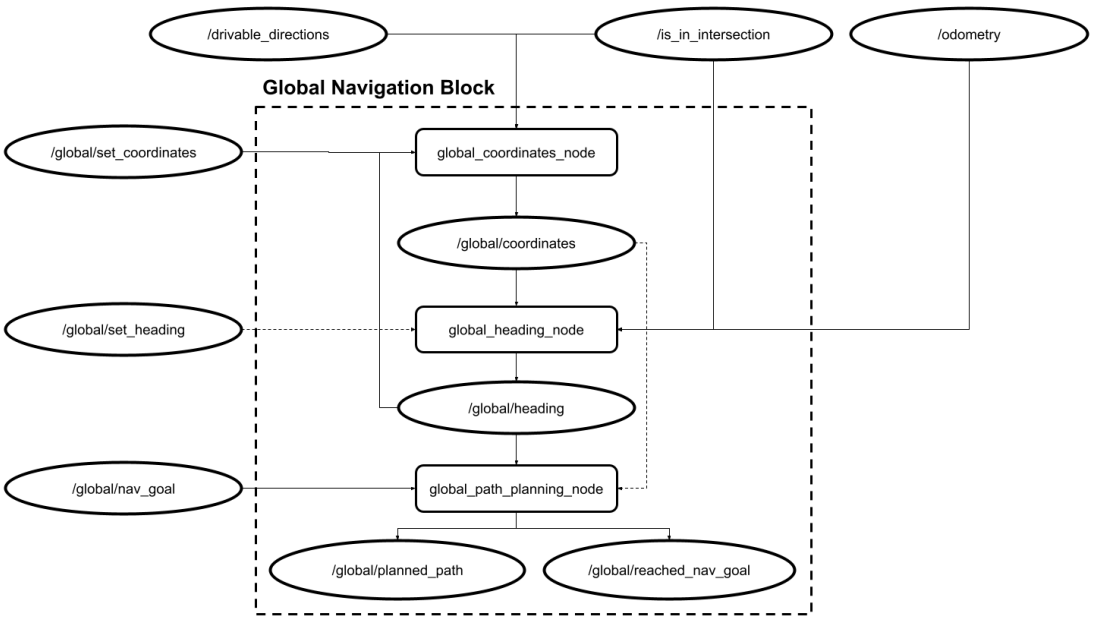


Figure 3.10: A data flow diagram that represents the interactions between nodes and topics in the global navigation block where ellipses correspond to topics and rectangles correspond to nodes.

3.2.2.1 Graphical User Interface

The Graphical User Interface (GUI) is the link between the user and the system and is responsible for handling input and providing the user with system related information. The GUI consists of an undirected graph that is displayed on top of an image of the office environment at CPAC Systems. The GUI can bee seen in Figure 3.11 where nodes represent intersections or start/end points, and edges represent hallways. Together this makes up the topological map which is the backbone of the system and is used when localizing and navigating the environment. At the top left in the GUI, there are buttons used for inputting initial arguments to the automated driving system. When clicking on the *Set current node* button, the user can input the initial starting position by clicking on the corresponding node in the topological map. Once a valid node is selected, the *Set current node* button turns green and it now displays the selected node instead. The user is then prompted to select which node the vehicle is currently facing towards and once this is done the button turns green and displays the selected heading node. Finally, the user is able to click on the *Set goal node* to select the node to navigate to in the topological map. When all necessary input has been given through the GUI, all buttons turn green indicating that the system has all the required arguments needed to execute the topological navigation. The user can then press the *Start* button and the car will autonomously navigate to the selected goal node. Once the start button is clicked it will transform into a *Stop* button which the user is able to press in case of emergency. There also exists a *Reset* button which enables the user to reset the current input.

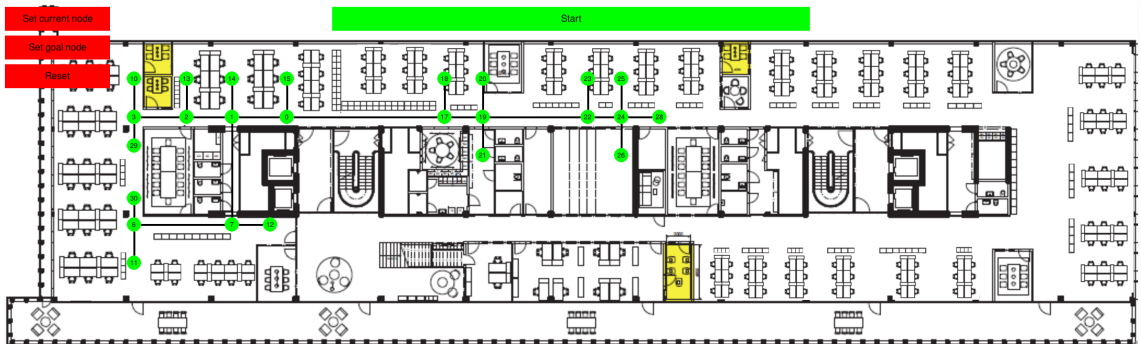


Figure 3.11: An overview of the GUI representing the 4th floor at CPAC systems with a graph added on top of it where nodes correspond to intersections and edges correspond to hallways. The graph and the office map make up the topological map that enables the user to intuitively select the current node, heading node and goal node. There are also a set of buttons that the user is able to interact with in order to make changes and manipulate the system.

3.2.2.2 Global path planning

The global path planning node, which utilizes Dijkstra's algorithm to find the shortest path between two nodes in the topological map, uses the current node, heading node, and goal node from the GUI as arguments. Since the vehicle is not able to do U-turns or reverse, the node directly behind the current node with

respect to the heading node is temporarily removed from the graph (if there is no node behind the current node, nothing is removed). This prevents impossible global paths from being generated and once a valid path is found, the shortest path is published back to the GUI node which displays the path in a cyan color. This can be seen in Figure 3.12 where a path is planned from node 18 to node 6.

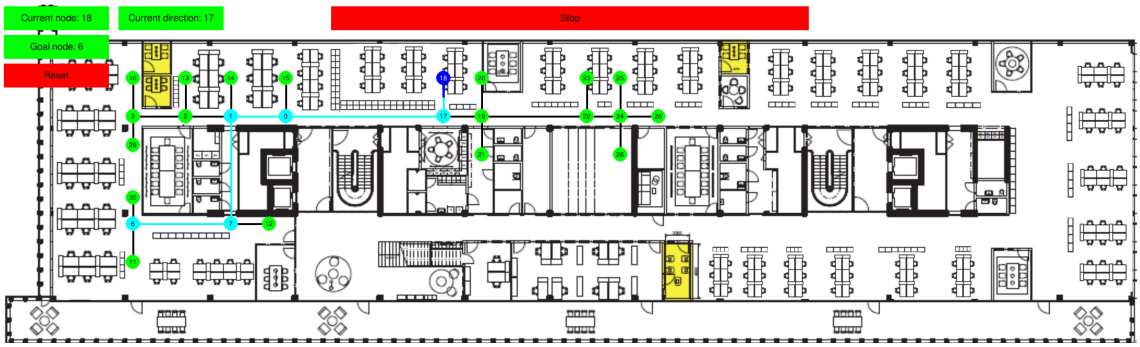


Figure 3.12: A graphical representation of the GUI with a drawn global path plotted based on the selected current node, current direction and goal node. The start button has also been pressed, transforming it into a stop button and enabling the autonomous driving mode for the vehicle.

3.2.2.3 Global coordinates

The global coordinates node is responsible for updating the vehicle's node position, which is represented as x,y coordinates, based on node transitions in the topological map. An update of global coordinates is only registered once the observed intersection matches the expected intersection. The observed intersection is received in real time from the drivable directions node and the expected intersection is calculated based on the number of edges of the heading node. This means that the global coordinates are discrete and can only be updated to the coordinates of the nodes in the topological map. An update of coordinates can be seen in Figure 3.14 where the vehicle encounters the intersection in Figure 3.13. The global coordinates node is also software limited to only publish one set of coordinates per intersection which means that the vehicle must see an edge for 0.5 meters before publishing a new set of coordinates. This prevents multiple coordinates from being published at the same intersection which would otherwise create an inaccurate localization. Once the global coordinates are set, they are then published and displayed by the GUI as a blue node on the topological map.

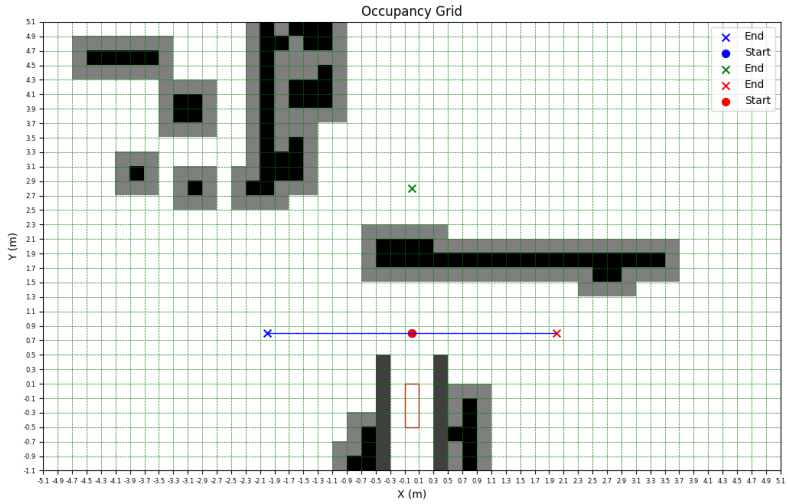


Figure 3.13: A plot of the drivable directions algorithm applied upon an occupancy grid where it is able to identify a T-intersection.

3.2.2.4 Global heading

When an intersection is entered, the global heading node starts integrating the change in heading. When the vehicle leaves the intersection, the global heading node finds which neighbouring node is the most likely one to be in the heading direction, and sets global heading to the angle between that node and the current node. This is illustrated in Figure 3.14.

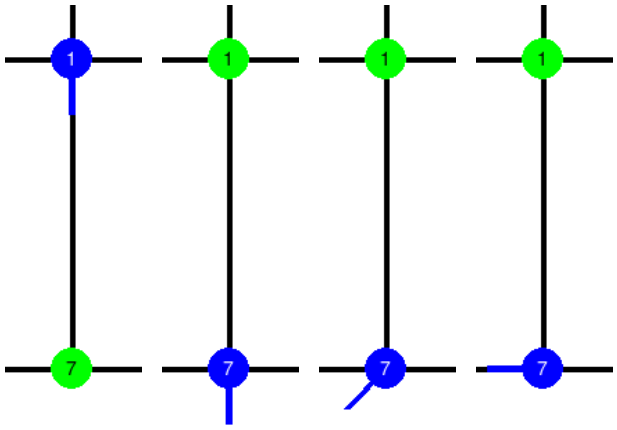


Figure 3.14: From left to right: as soon as intersection 7 is entered, the change in global heading is integrated until the vehicle leaves the intersection. The blue line represents the current global heading and the blue node is the current position of the vehicle. When the vehicle leaves the intersection, the global heading is set to the angle of the neighbouring node most likely to be in the heading direction. In this figure, that node is the node to the left of node 7, so therefore the global heading is set to 180 degrees (the unit circle convention is used with 0 degrees being straight to the right, 90 degrees straight up and so on).

3.2.3 Local navigation block

The local navigation block is responsible for finding a local navigation goal based on the global navigation goal and the current global position, planning local paths as well as ensuring the planned path is followed. This is done as three separate sub-tasks:

1. **Local navigation goal finding** searches for a target point relative to the current surroundings to navigate to. The position of this point depends on the vehicle's current position, globally planned path as well as whether the vehicle is in an intersection or not.
2. **Local path planning** finds a collision-free path from the vehicle's current position to the local navigation goal.
3. **Local path following** ensures that the planned local path is followed and requests a new path when it starts getting close to the end of the locally planned path.

A data flow diagram of the local navigation block can be seen in Figure 3.15 where nodes and topics interact with each other.

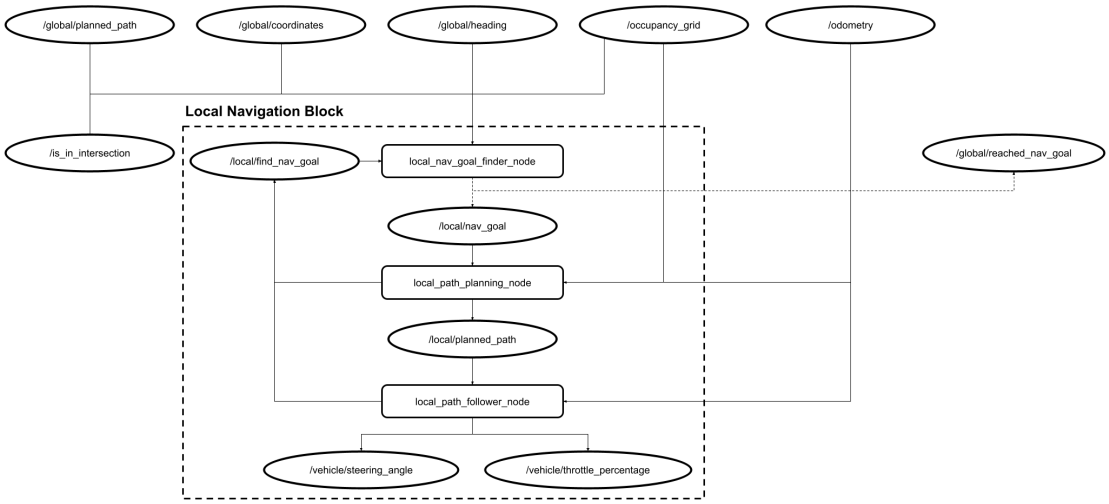


Figure 3.15: A visual representation of the interactions between the components of the local navigation block.

3.2.3.1 Local navigation goal finding

When a local navigation goal is requested, the local navigation goal finder node checks whether the vehicle is in an intersection. If the vehicle is in an intersection and needs to turn left or right to reach the next global node, then the navigation goal is set 1.2 meters ahead and 2.75 meters to the side. If the vehicle is in an intersection but should not turn in the intersection, then the navigation goal is set 2 meters ahead. When not in an intersection, Hough line transform is used on the non-ground LiDAR points to identify the road edges as two almost parallel lines. The line pair representing the road edges is picked based on the following criteria:

1. The line pair has to be almost parallel with a maximum allowed difference of 30 degrees.
2. The average angle of the lines has to be within 45 degrees of 90 degrees to ensure the lines are along the travelling direction.
3. The parallel distance between the lines has to be at least 0.8 meters. This is to avoid choosing lines that are part of the same wall.

The line pair with the longest average line length that fulfills the aforementioned criteria is picked as the line pair representing the road edges. The navigation goal is then picked in the middle between those two lines so that it lies 3 meters away from the vehicle. This process of filtering out lines and picking a navigation goal is illustrated in Figure 3.16.

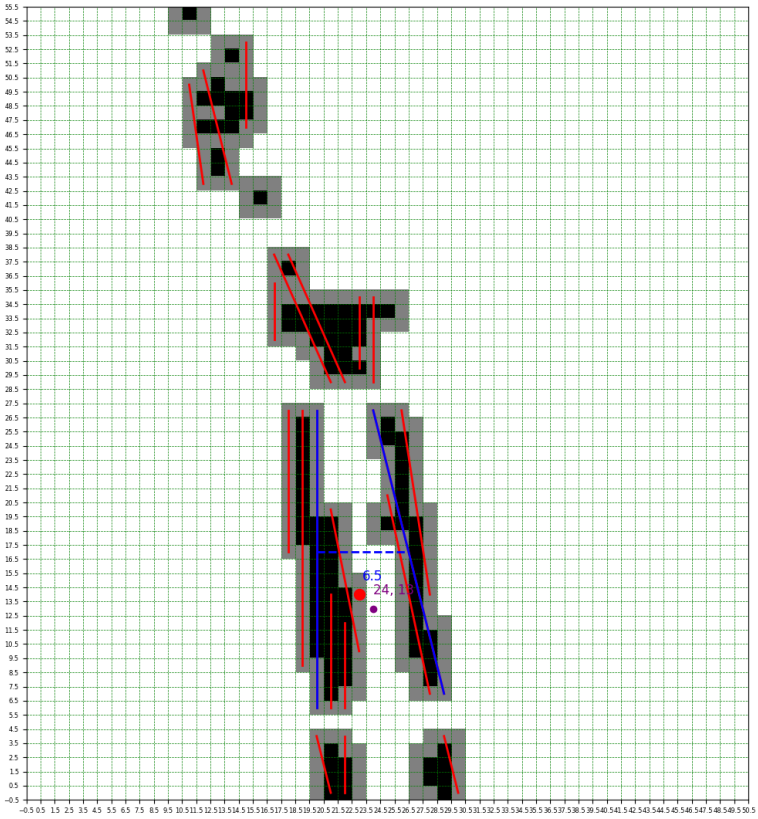


Figure 3.16: Vehicle driving along a corridor towards a T-intersection. Each cell is 0.2×0.2 meters. The black cells contain non-ground points as identified by ground segmentation and the grey cells are an extra layer of padding around the non-ground points so that the vehicle stays at least 0.2 meters away from them. The drawn lines are the lines found by Hough line transform. The two blue lines are the lines identified as being the road edges. The parallel distance between them is indicated by the dotted blue line whose distance is equal to 6.5 cells, which is $6.5 \cdot 0.2 = 1.3$ meters. The red dot is the local navigation goal that is between the two blue lines and approximately 3 meters away from the vehicle. Unfortunately, it is within a grey cell, so it is moved to the closest unoccupied cell as indicated by the purple dot.

3.2.3.2 Local path planning

Given a local navigation goal, the local path planning node uses the A* algorithm with the octile distance heuristic to find the shortest path to the goal, as illustrated in Figure 3.17. The octile distance heuristic is chosen because the planning is done on a grid where diagonal movement is allowed (if it wasn't on a grid, then the Euclidian distance heuristic would have been chosen instead and if diagonal movement wasn't allowed, then the Manhattan distance heuristic would have been chosen instead). The octile distance heuristic is defined as [44]:

$$D \cdot (\Delta x + \Delta y) + (D^2 - 2D) \cdot \min(\Delta x, \Delta y)$$

Where D is the cost of non-diagonal steps, D^2 is the cost of diagonal steps and Δx and Δy is the number of steps in x and y respectively from start to goal. In our case $D = 1$ and $D^2 = \sqrt{2}$. The intuitive interpretation of this heuristic is to take the number of steps required if diagonal movement is not allowed, and then subtract the number of steps saved by taking diagonal steps.

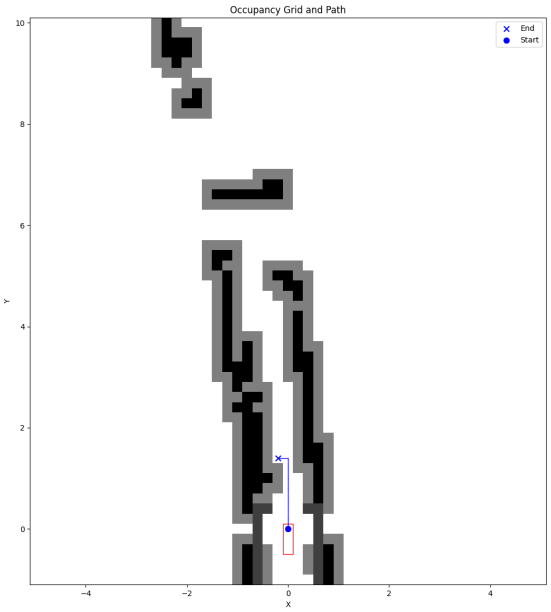
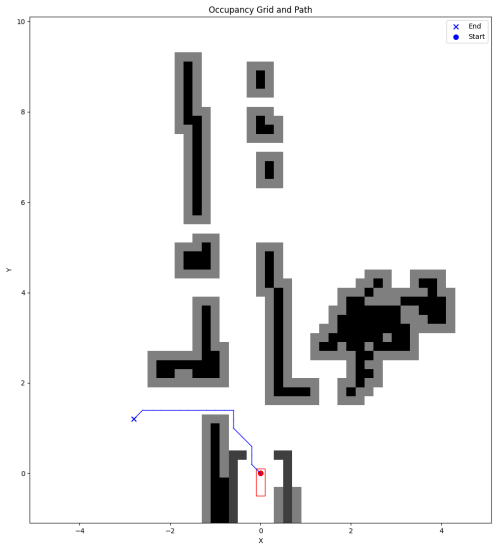
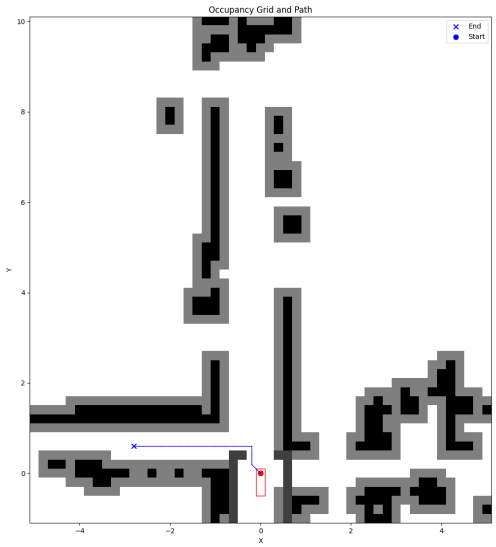


Figure 3.17: Vehicle driving along a corridor towards a T-intersection. The A* algorithm found the shortest path, drawn in blue, from the start point (blue dot) to the local navigation goal (blue cross). The vehicle is drawn as a red rectangle. The dark grey cells represent artificial walls that have been added around the vehicle so that A* does not plan through the LiDAR's blind spot and so that the vehicle does not make sharp turns. The blind spot can be identified as holes in the wall on either side of the vehicle (the vehicle is driving straight through a corridor, so there should be no holes in the corridor walls, but due to the LiDAR's blind spot there are holes in the walls to the left and to the right of the vehicle).

To prevent the algorithm from planning through the LiDAR's blind spot as well as prevent sharp turns, artificial non-traversable cells are added around the vehicle. While the vehicle travels along the path, the local path planning node also keeps continuously checking if the planned path is collision-free, ie. it does not go through any non-traversable cells. If the path does go through non-traversable cells, then local path planning will try to find a new path. If no path is possible, the vehicle will attempt to follow the original path, otherwise it will attempt to follow the new path. This usually happens during turns as the LiDAR cannot see around corners, so the turn maneuver ends up being on a collision course with one of the walls in the corridor the vehicle is turning into. An example of path replanning during a turn is illustrated in Figure 3.18.



(a) Path planning has planned a collision-free left-turn maneuver.



(b) As the vehicle approaches the turn and the LiDAR sees around the corner, it is realized that the previously planned path leads into the wall. A new collision-free route is therefore planned.

Figure 3.18: Vehicle driving along a corridor preparing for a left turn maneuver.

3.2.3.3 Local path following

Most of steering models are based on, or contain elements from, control theory wherein deviations from a desired path are corrected by outputting one of the following quantities: the steering angle, the vehicle steering angle, or the lateral acceleration [45]. Craig Reynolds' path-following algorithm is one such model and can be used to follow the locally planned path [46]. This algorithm estimates the vehicle's future position and steers it towards the planned path if it deviates more than a set threshold from the path by outputting a vehicle steering angle. This algorithm is used for local path following and Figure 3.19 illustrates the intuition behind this algorithm.

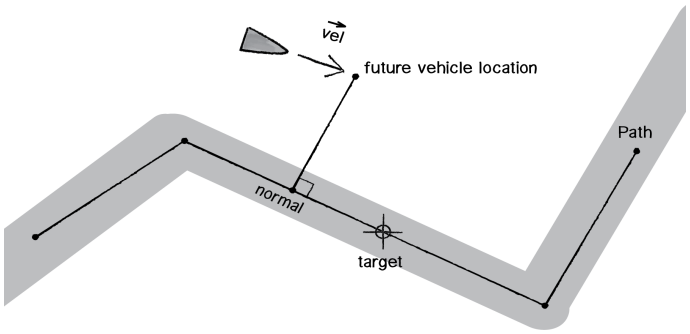


Figure 3.19: The path-following algorithm estimates the vehicle's future location based on the vehicle's velocity. If the future location deviates more than a set threshold from the path, marked by the grey highlight around the path, a steering signal will be applied to steer the vehicle towards the path. The target point to steer towards is chosen by first finding the point on the path that has the shortest distance to the vehicle's future position, and then taking a point on the path that is slightly further ahead [46].

4
Results

In order to assess the performance of the system, several paths were autonomously driven to evaluate the accuracy of localization, navigation, and autonomous driving. For the purpose of answering the research question: *Is it possible to apply an ADS designed for confined environments on an unconfined environment?*, the ADS was tested on an outdoor environment. This was done to asses the generalization ability of the system as well as to identify any potential issues that could surface in a real underground mine. All results were gathered using the ADS structure proposed in Figure 3.3.

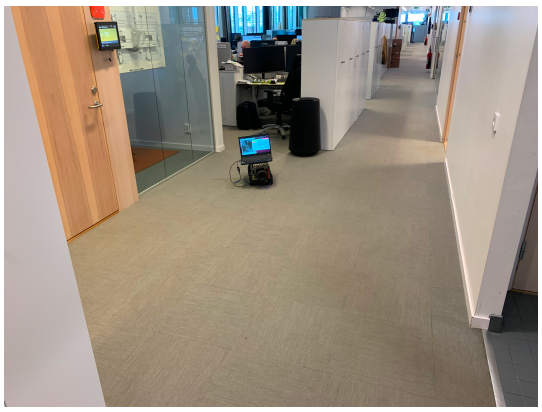
4.1 Autonomous navigation in a confined environment

The results of evaluating the ADS in a confined office environment are summarized in Table 4.1. The ADS was unable to complete routes 1 and 2 for the path 6 - 25 in both directions where it failed to traverse the edge 19 - 22. The corridor gets wider in this edge and there is also a room with a glass wall to the side, which is illustrated in Figure 4.1. The ADS failed due to the found road edges not corresponding to the continuation of the corridor and thus also the wrong local navigation goal being chosen, which is illustrated in Figure 4.2. The reason behind this failure can be one of the following:

- the ADS is unable to handle temporary abrupt increases in corridor width,
- the room with the transparent glass wall was identified as the continuation of the corridor instead of the actual corridor,
- a combination of both of the above.

Nr.	Path	Nodes traversed	Success	Comment
1	$6 \rightarrow 25$	5/8	X	Did not find correct road edges
2	$25 \rightarrow 6$	3/8	X	Did not find correct road edges
3	$21 \rightarrow 3$	6/6	✓	
4	$3 \rightarrow 21$	6/6	✓	
5	$7 \rightarrow 18$	4/4	✓	
6	$18 \rightarrow 7$	4/4	✓	
7	$13 \rightarrow 15$	4/4	✓	
8	$15 \rightarrow 13$	4/4	✓	
9	$14 \rightarrow 19$	4/4	✓	
10	$19 \rightarrow 14$	4/4	✓	
	Total	44/52	8/10	Traversed acc. 84.6%, Success acc. 80%

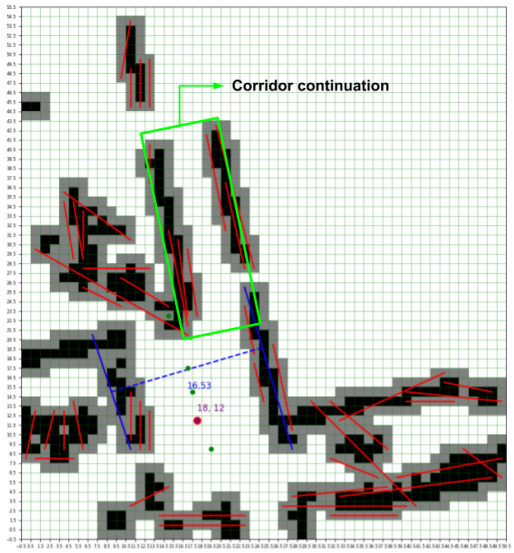
Table 4.1: ADS evaluated by driving 10 routes with 5 distinct paths in the office described by the topological map in Figure 3.11. Each path was driven in both directions. The ADS failed to traverse routes 1 and 2 which was for path 6 - 25. For both directions the ADS failed to traverse the edge 19 - 22. The total route accuracy was therefore 8/10, ie. 80%. In total, 44 out of 52 nodes were traversed correctly, ie. a 84.6% accuracy.



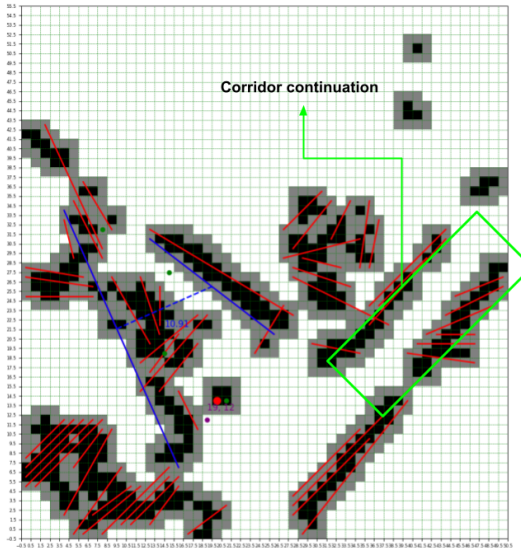
(a) Edge $19 \rightarrow 22$. There is a room with a transparent glass wall on the left that the LiDAR can see through.

(b) Edge $22 \rightarrow 19$. There is a room with a transparent glass wall on the right that the LiDAR can see through.

Figure 4.1: Edge 19 - 22 which the ADS failed to traverse.



(a) As the corridor gets wider, the ADS steers to position the vehicle in the middle of the wider corridor.



(b) As the vehicle positions itself in the middle of the wider corridor, the LiDAR sees through the glass wall into the room ahead of it, the ADS identifies the room walls as the continuation of the corridor and then tries to steer the vehicle into that room.

Figure 4.2: ADS failing to traverse the edge 19 → 22 due to the found road edges not corresponding to the continuation of the corridor. Blue lines are the lines identified as the road edges. Purple point is the point between the road edges chosen as the navigation goal. The ADS fails in a similar manner when traversing this edge in the opposite direction, ie. 22 → 19.

4.2 Gravel road results

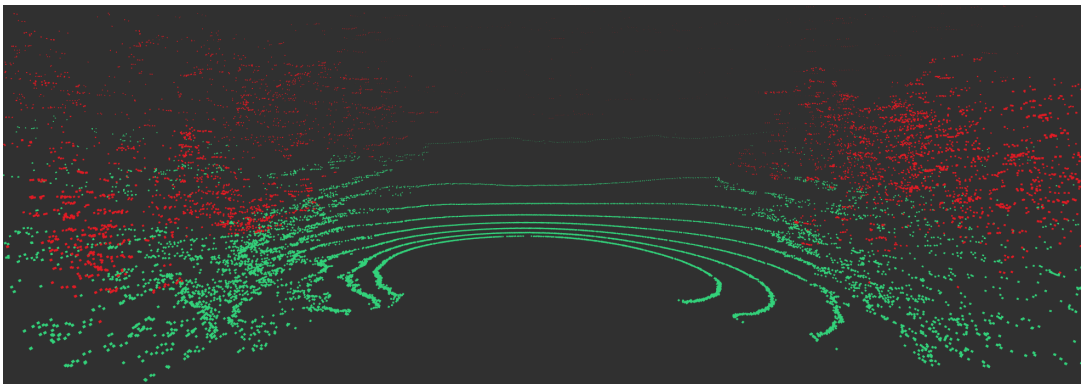
Although the focus of the work was on confined environments with corridor-like connections, such as underground mines, navigation based on a topological map should in theory work for any environment as long as the road can be correctly segmented and intersections can be classified. As the utilized ground segmentation algorithm is not dependent on corridor-like structures, but rather on the flatness of the surface, it should be able to generalize to any environment where the road can be clearly differentiated due to being flatter than its surroundings. Furthermore, access to a mine was not available to evaluate the ADS in the environment it was designed for, so instead it was decided to have the vehicle drive on gravel roads to evaluate the ability of the ADS to generalize to non-corridor-like environments as well as investigate the performance of the ADS in a less perfect environment. Gravel roads were therefore chosen as the road boundaries are not as clearly defined and are not as straight and parallel as the ones in an office environment. The gravel roads chosen were in the park to the north of the Guldheden Södra football field in Gothenburg. The road width there was different compared to the office environment the ADS was previously tested on, so it was hoped that tuning a few parameters related to the road width would suffice. However, that was not the case and the performance of the ADS was too poor to conduct any meaningful tests.

4.2.1 Ground segmentation

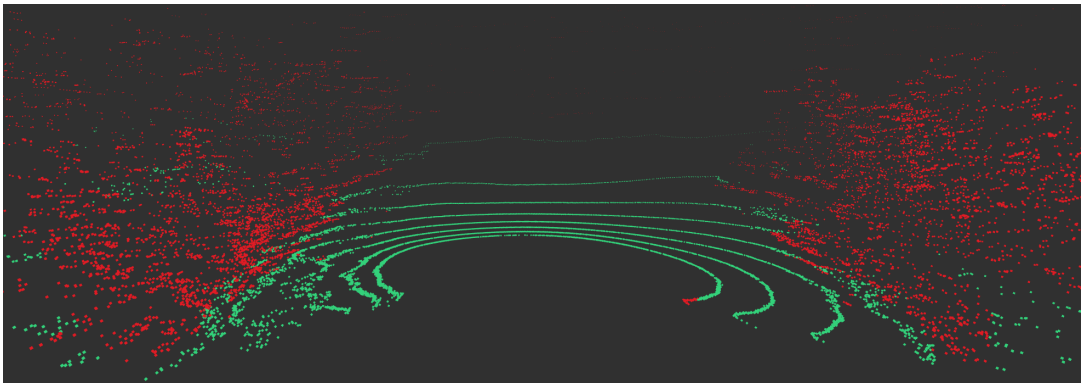
Ground segmentation successfully identifies the gravel road, but it also misclassifies some of the low vegetation near the road edge as also being part of ground, as illustrated in Figure 4.3c. Decreasing the threshold for the ground thickness threshold from 12.5 centimeters to 3 centimeters improves the results as less of the adjacent vegetation is misclassified as part of the road, as illustrated in Figure 4.3d.

(a) Third person view of the gravel road.

(b) First person view of the gravel road from the vehicle camera.



(c) Ground segmentation results where green and red points are ground and non-ground points respectively. As can be seen, while the algorithm successfully identifies the road, it also identifies a lot of the low vegetation near the road edges as ground points as well.



(d) Ground segmentation results of gravel road after decreasing the ground thickness threshold from 12.5 centimeters to 3 centimeters. Green and red points are ground and non-ground points respectively.

Figure 4.3: Ground segmentation results of a gravel road.

4.2.2 Occupancy grid

With a 3 centimeters ground thickness threshold on ground segmentation, the occupancy grid manages to capture the road characteristic quite well as illustrated in Figure 4.4.

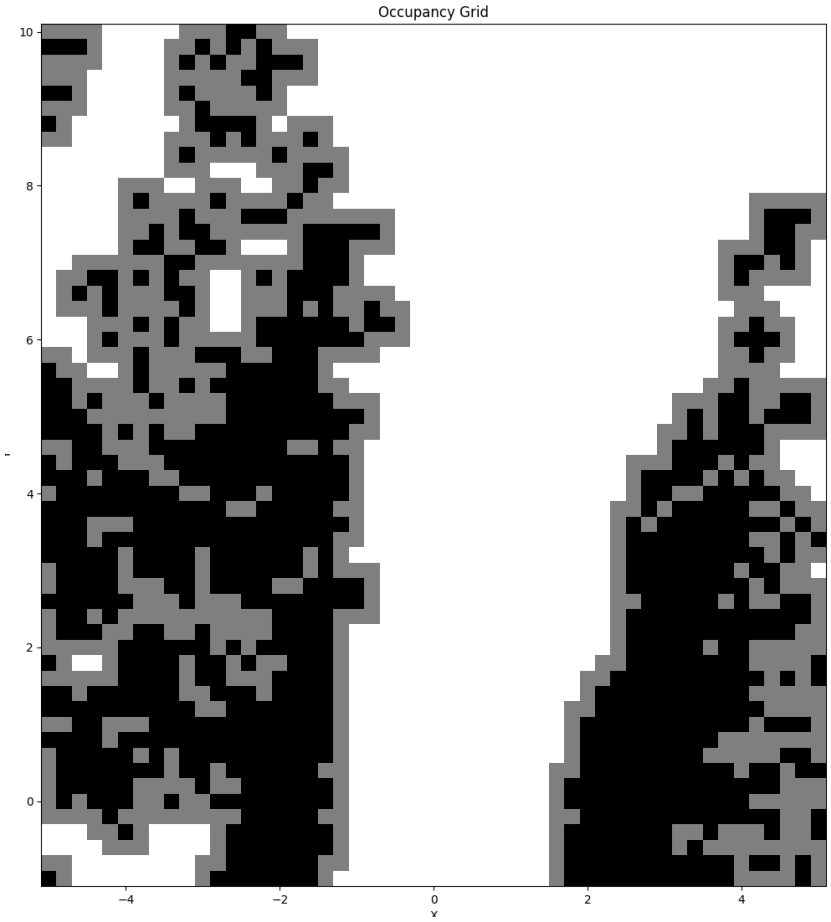


Figure 4.4: Occupancy grid generated from ground segmentation of the environment from Figure 4.3d. The ground thickness threshold in ground segmentation has been decreased to 3 centimeters.

4.2.3 Intersection detection

As the parameters in intersection detection were tuned according to the corridor width of the office, the ADS fails to detect intersections due to the road being much wider, which is illustrated in Figure 4.5.

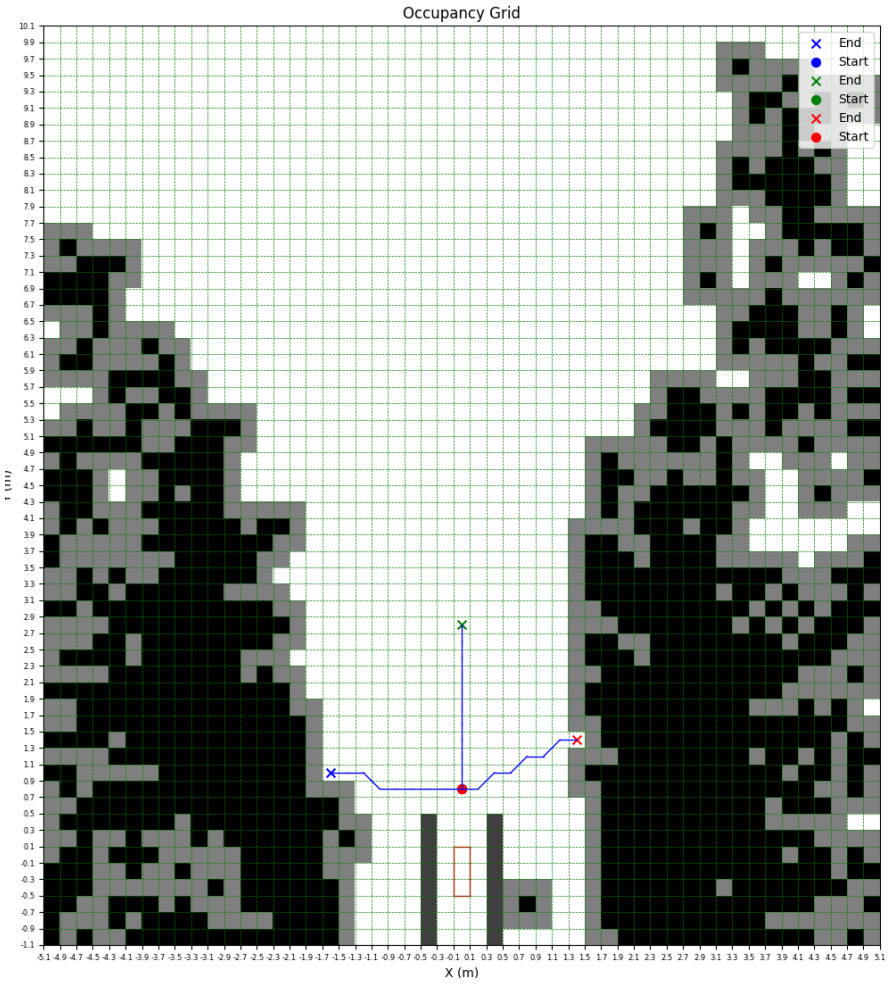


Figure 4.5: The straight path is classified as a 4-way intersection due to the road being so wide in comparison to the corridors in the office environment the intersection detection parameters were tuned for.

4.2.4 Local navigation goal finding

One of the criteria for finding the lines which make up the road edges was that the lines had to be at least 0.8 meters apart to avoid choosing lines that were part of the same wall from being chosen. However, this was too short of a distance when driving on wider gravel roads as the detected "walls" were a lot wider, so lines part of the same wall were still chosen, as illustrated in Figure 4.6. Increasing the minimal distance between the lines to 3 meters does improve the road edge detection somewhat, as illustrated in Figure 4.7, but it is still inaccurate as it is not exactly the road edge. This can be further improved if the Canny edge detector is applied first before applying the Hough line transform and then choosing the line pair closest to the vehicle that is also at least **2 meters apart**, as illustrated in Figure 4.8.

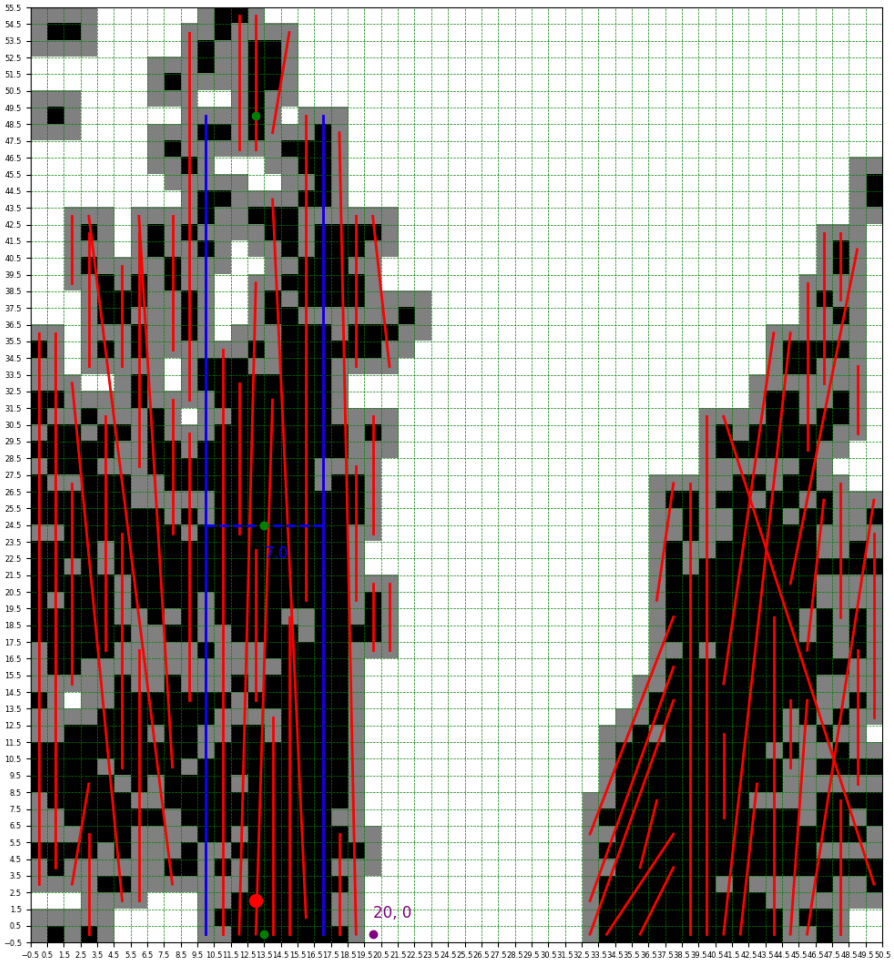


Figure 4.6: Road edges incorrectly identified when driving outside on gravel roads due to the minimal distance between the lines being too short. The blue lines represent the identified road edges.

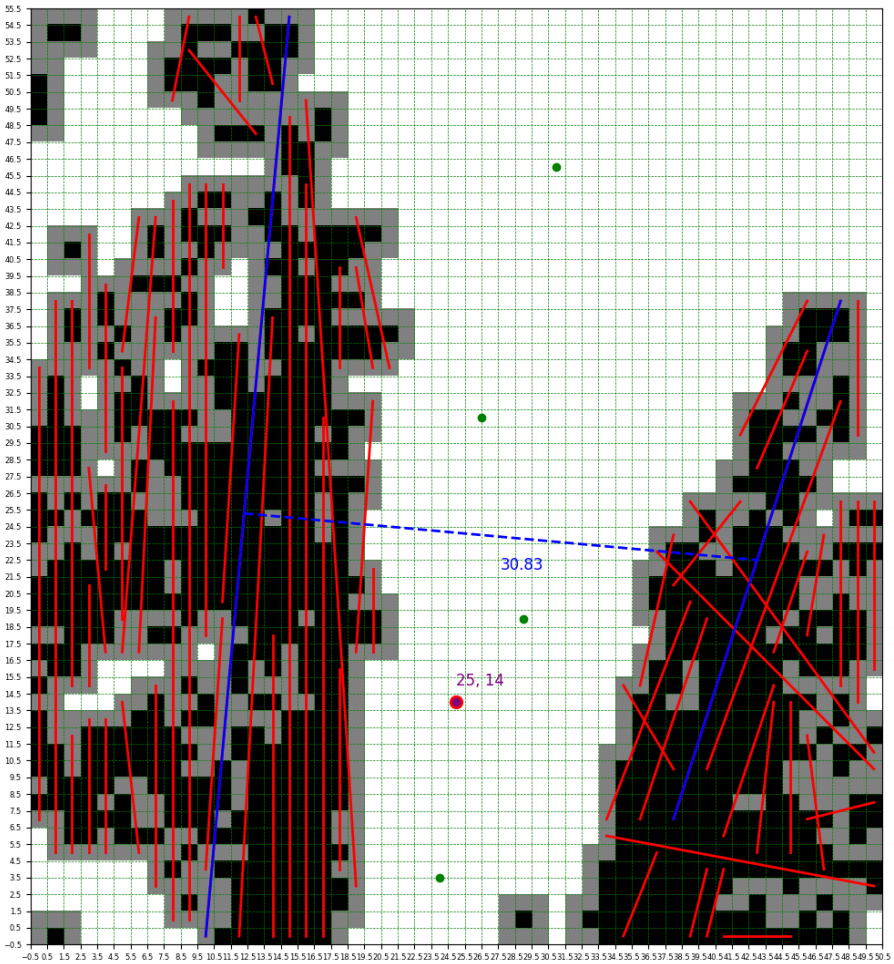


Figure 4.7: Identified road edges in blue when driving outside on gravel roads. The minimal distance between the lines has been increased to 3 meters. The picked local navigation goal is plotted as a purple dot. While the road edges are not entirely correct, the midpoint between the blue lines is still somewhat in the middle of the road.

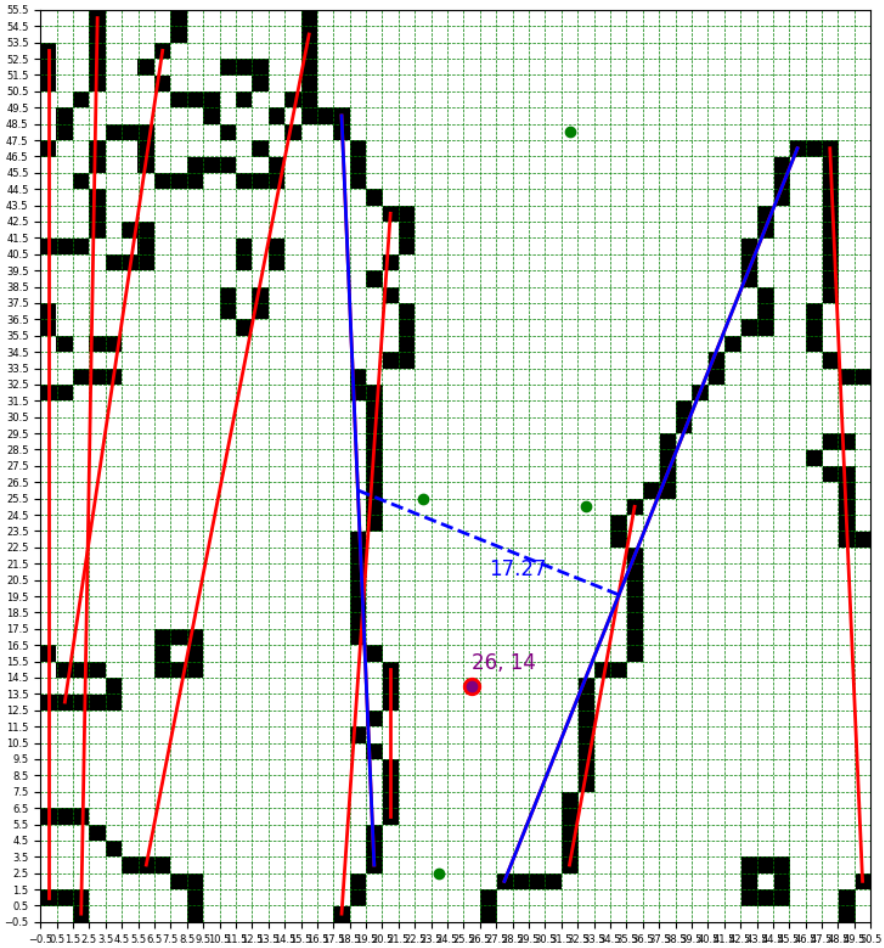


Figure 4.8: Identified road edges in blue when driving outside on gravel roads. The Canny edge detector has been applied before applying the Hough line transform to find the lines representing the road edge. The black squares are the edges found by the Canny edge detector. The line pair that is at least **2 meters apart** and closest to the vehicle has been chosen as the road edge line pair.

5
Discussion

When analyzing the results gathered while autonomously navigating a confined environment in Section 4.1, it is possible to see that the system performed well and managed to traverse 84.6% of the nodes resulting in a 80% success accuracy. Although the outcome seems promising and the ADS performed well under controlled circumstances, it is uncertain if the ADS translates well into a real underground environment. For the topological localization and navigation to function as intended, the intersection detection is essential for a reliable system. However, in the results gathered above, the intersection detection algorithm proposed in Section 3.2.1.3 is limited to and can only detect 90° intersections. Given the complex underground intersections of a mine, the 90° limitation may not generalize that well and give poor performance due to inconsistency of intersection angles in an underground environment. The proposed intersection detection algorithm is also limited to only identifying up to 4-way intersections which means that the topological localization and navigation may fail if an intersection type of higher degree is detected. Due to the entire system being dependent on an accurate and general intersection detection algorithm, it might be of importance to consider some of the algorithms proposed in Section 1.1 instead.

The LiDAR's blind spot that was brought up in Section 2.2 also constituted a major problem for the development and performance of the ADS. The fact that the LiDAR could not detect objects in its immediate vicinity prevented an accurate representation of the surroundings which negatively impacted the local navigation ability. Several work-arounds were needed to be implemented in order to address the blind spot problem which degraded the performance of the system. Due to the severity of the problem, the nodes in the ADS needed to be developed with the blind spot in mind, making the autonomous solution custom fitted to counteract the blind spot rather than purely focusing on the autonomous task at hand. The problem with the blind spot was solved in this thesis by implementing the artificial walls discussed in Section 3.2.1.3 and visualised in Figure 3.8. As a result of this, the artificial walls were always present next to the vehicle even though there were no real walls in its vicinity which leads to numerous problems in the path planning ability. Although this is just a temporary solution, a more permanent one would consist of adding another LiDAR that can fill in the blind spot of the first one. By combining two separate point clouds from two different LiDARs, a more accurate representation of the environment can be achieved and the blind spot problem could be tackled. Another solution would consist of adding a pair of ultrasonic sensors around the vehicle that can assist the LiDAR by detecting close up objects.

Furthermore, when designing an ADS for a confined environment, it is important to consider several key factors to improve the outcome, the development process and the extendability. One of the main reasons why the modular approach was chosen over the end-to-end driving approach was due to the Separation of Concerns (SoC) which is a software design principle used for dividing a complex task into several submodules, each responsible for a specific function of the system. This approach improved the development process of nodes due to the ability to work in parallel without being constrained and dependant on other nodes. The SoC also enabled the ability to verify and test the functionality of each node individually which simplified the troubleshooting and debugging process. Therefore, when encountering an incorrect behaviour in the ADS's functionality during the testing phase, the problem could be quickly pinpointed to a specific node due to the isolated structure of the ADS. The benefits of a modular approach are also apparent when doing local modifications to a node which was especially noticeable when developing the *is in intersection* node. The first version used time to detect when the vehicle had exited an intersection. However, after testing it was concluded that distance was a more reliable measurement. Thanks to the isolated node structure, this was quickly modified in the corresponding node without impacting other functionality in the ADS. Furthermore, end-to-end approaches usually rely on machine learning methods which require labeled training data. As labeled training data of a confined environment was not readily available and creating such a data set would be time consuming, an end-to-end approach was deemed unrealistic given the time constrains.

In order to asses the generalization of our ADS, the system was applied on a non-confined environment. Given that the ground segmentation method used in this thesis is not specifically designed for confined environments, but rather a general solution that segments drivable surface based on leveled planes, it should in theory work on any environment including non-confined environments as well. Based on the results gathered in Section 4.2 it can be seen that the algorithm roughly manages to segment the gravel road presented in Figure 4.3c. When applying the hyper-parameters used for the confined environment navigation, the outdoor segmentation sometimes classifies roadside vegetation as drivable surface, leading to incorrect representation of the surroundings. However, after some hyper-parameter tuning where the ground thickness threshold was decreased from 12.5 to 3 centimeters, it can be seen in Figure 4.3d that the algorithm now manages to more correctly segment the vegetation as non-drivable surface. Even though the performance has improved, the segmentation still missclassifies some of the low vegetation along the road which decreased the performance of the ADS. However, the road characteristics can still be identified as can be seen in Figure 4.4, which means that hopefully with further work the designed ADS should be able to navigate even such an irregular environment as this. The system also struggled with following the road as it had difficulty identifying the road edges due to the "walls" being wider. However, applying a Canny edge detector and then choosing the line pair closest to the vehicle and at least 2 meters apart seems to

improve the road edge identification significantly, as illustrated in Figure 4.8. Although there was no time to test this, the ADS would most likely be able to follow the road with these improvements, but it might still sometimes identify the wrong road edges. A further improvement could be to also take the amount of drivable surface between the line pairs into account when choosing which line pair corresponds to the road edges. In order to adapt the system to the outdoor environment, it was also necessary to do some further hyper-parameter tuning, especially with the *Drivable directions* node given the larger scale of the environment. Therefore, due to time constraints, extensive hyper-tuning of the system's parameters was not possible and the assessment of the designed ADS's ability to autonomously navigate in non-confined environments remains inconclusive. However, under the assumption that a non-confined environment can be transformed into a confined environment representation, an ADS designed for confined environments should be able to function even in non-confined environments.

As previously mentioned, many parameters depend on the road width and need to be tuned for every new environment, which is time-consuming. However, the road width can actually be calculated based on the distance between the identified road edges. Therefore, with some further work the ADS should be able to set its own parameters adaptively based on past road widths calculated from the road edge identification. The local path planning also does not take the vehicle dynamics, such as the turning radius, into account when planning a path, so sometimes the turns are too sharp causing the vehicle to crash into the wall when turning at an intersection. Furthermore, when turning at an intersection the turn navigation goal is not set dynamically: it is always set 1.2 meters ahead and 2.75 meters to the side. This will not generalize well as these parameters are dependent on the road width and on the utilized intersection detection method. A possible improvement would be to use the method proposed by Yuanjian Jiang et al. [39] which is able to generate dynamic intersection turn paths. The reason it was not utilized in this work was due to time constraints: the Hough line transform method is readily available via the OpenCV Python package, whereas the method proposed by Yuanjian Jiang et al. would have needed to be implemented from scratch.

6
Conclusion

As there is no GNSS coverage underground, automation of vehicles operating underground is challenging without the deployment of infrastructure or sensor mapping of the entire environment. The aim of this thesis was to therefore develop an ADS for confined environments without GNSS coverage, propose alternative intersection detection methods and analyze if an ADS designed for a confined environment without GNSS coverage can be generalized to an unconfined environment. The developed ADS was verified to have a high success rate in an office environment. Two new intersection methods were proposed: a ray-casting based method and a path finding based method. Both approaches are limited to intersections with only 90 degrees turns and will thus not generalize well. While the designed ADS could not generalize to an unconfined environment, it is likely that an ADS with a more robust intersection detection and road edge identification will be able to operate in an unconfined environment if the environment can be transformed into a confined environment representation.

With a more robust intersection detection method the developed ADS could be deployed to automate vehicles operating underground and increase the productivity and safety of mining operations. Further work needs to be done to determine if methods developed for confined environments can be extended to unconfined environments as well. If that is the case, then some of the methods designed for confined environments could possibly help automate the vehicles driving on the roads above ground as well.

Bibliography

[1] Z. Ren and L. Wang, "Accurate real-time localization estimation in underground mine environments based on a distance-weight map (dwm)," *Sensors*, vol. 22, no. 4, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/4/1463

[2] K. Nielsen, "Robust lidar-based localization in underground mines," https://liu.diva-portal.org/smash/record.jsf?pid=diva2:1543905, 2021.

[3] S. Rusu, M. Hayes, and J. Marshall, "Localization in large-scale underground environments with rfid," https://www.researchgate.net/publication/224259666_Localization_in_large-scale_underground_environments_with_RFID, 2011.

[4] R. D. Jones, J. E. Diener, Y. Chen, A. Z. Elsherbeni, and J. Brune, "Underground localization system using a combination of rfid and imu technologies," in *2021 International Applied Computational Electromagnetics Society Symposium (ACES)*, 2021, pp. 1–4.

[5] M. Mascaró, I. Parra-Tsunekawa, C. Tampier, and J. Ruiz-del Solar, "Topological navigation and localization in tunnels—application to autonomous load-haul-dump vehicles operating in underground mines," *Applied Sciences*, vol. 11, no. 14, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/14/6547

[6] G. Johansson and M. Wasteby, "Hybrid map for autonomous commercial vehicles - global localization using topological mapping and machine learning," https://odr.chalmers.se/items/dd1ad5e5-e752-4718-bacf-ef0935682c39, 2017.

[7] J. Larsson, M. Broxvall, and A. Saffiotti, "Laser based intersection detection for reactive navigation in an underground mine," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2222–2227.

[8] T. Chen, B. Dai, D. Liu, and Z. Liu, "Lidar-based long range road intersection detection," in *2011 Sixth International Conference on Image and Graphics*, 2011, pp. 754–759.

[9] A. Morris, D. Silver, D. Ferguson, and S. Thayer, "Towards topological exploration of abandoned mines," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2117–2123.

[10] J. Larsson, M. Broxvall, and A. Saffiotti, "A navigation system for automated loaders in underground mines," in *FIELD AND SERVICE ROBOTICS*, ser. Springer Tracts in Advanced Robotics, P. Corke and S. Sukkarieh, Eds., vol. 25. CSIRO ICT; Australian Robot & Automat Assoc, 2006, pp. 129+, 5th International Conference on Field and Service Robotics, Port Douglas, AUSTRALIA, JUL 29-31, 2005.

[11] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[12] C. Gomez-Huelamo, A. Diaz-Diaz, J. Araluce, M. E. Ortiz, R. Gutierrez, F. Arango, A. Llamazares, and L. M. Bergasa, "How to build and validate a safe and reliable autonomous driving stack? a ros based software modular architecture baseline," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1282–1289.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[14] G. Johansson and M. Wasteby, "Hybrid map for autonomous commercial vehicles: Global localization using topological mapping and machine learning," 2017.

[15] A. Ranganathan and F. Dellaert, "Online probabilistic topological mapping," *I. J. Robotic Res.*, vol. 30, pp. 755–771, 05 2011.

[16] S. Thrun, S. Gutmann, D. Fox, W. Burgard, and B. Kuipers, "Integrating topological and metric maps for mobile robot navigation:," 10 1998.

[17] Leah A. Wasser, "The basics of lidar - light detection and ranging - remote sensing," https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics, 2022, accessed: may 17, 2023.

[18] N. N. O. Service, "What is lidar?" 2023, accessed: May 21, 2023. [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html

[19] Valeo, "Valeo scala® lidar," https://www.valeo.com/en/valeo-scala-lidar/, accessed: 2023-05-22.

[20] V. LiDAR, *USER'S MANUAL AND PROGRAMMING GUIDE*, Velodyne LiDAR, Inc, 2016.

[21] L. Weerakoon, "Cartographer_glass: 2d graph slam framework using lidar for glass environments," *arXiv preprint arXiv:2212.08633*, 2022. [Online]. Available: https://ar5iv.org/abs/2212.08633

[22] O. Robotics, "Ros humble documentation," https://docs.ros.org/en/humble/, 2010, accessed: April 26, 2023.

[23] S. Lee, H. Lim, and H. Myung, "Patchwork++: Fast and robust ground segmentation solving partial under-segmentation using 3d point cloud," 2022.

[24] H. Lim, M. Oh, and H. Myung, "Patchwork: Concentric zone-based region-wise ground segmentation with ground likelihood estimation using a 3d lidar sensor," 2022.

[25] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[26] S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics," 2005.

[27] J. Faigl, "Grid and graph based path planning methods," PowerPoint slides for lecture in Artificial Intelligence in Robotics, 2017, czech Technical University in Prague.

[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[29] A. Shaffer, "Dijkstra's algorithm," https://medium.com/codex/dijkstras-algorithm-16c14151f89c, 2021, [Online; accessed 25 April 2023].

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[31] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Addison-Wesley Professional, 2011.

[32] I. C. Science, "A* search algorithm," 2023. [Online]. Available: https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all

[33] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[34] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, p. 1897, 2016. [Online]. Available: https://doi.org/10.1186/s40064-016-3573-7

[35] L. H. Granström and C. H. Hedén, "Real time lidar and icp-based odometry in dynamic environments," https://liu.diva-portal.org/smash/get/diva2:1696544/FULLTEXT01.pdf, 2022, accessed on 30 April 2023.

[36] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[37] J. Larsson, M. Broxvall, A. Saffiotti, and A. Copco, "Laser-based corridor detection for reactive navigation," *Industrial Robot: An International Journal*, vol. 35, 01 2008.

[38] A. Saglam and Y. Papelis, "Realtime corridor detection for mobile robot navigation with hough transform using a depth camera," in *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, 2021, pp. 683–688.

[39] Y. Jiang, P. Peng, L. Wang, J. Wang, J. Wu, and Y. Liu, "Lidar-based local path planning method for reactive navigation in underground mines," *Remote Sensing*, vol. 15, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/2/309

[40] OpenCV, "Hough line transform," https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, accessed on May 13, 2023.

[41] J. Li, T. Ning, P. Xi, B. Hu, and T. Wang, "An analysis-oriented parameter extraction method for features on freeform surface," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 233, p. 095440621986200, 07 2019.

[42] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314299908317

[43] S. Lee, H. Lim, and H. Myung, "Patchwork++," https://github.com/url-kaist/patchwork-plusplus, accessed 2023-04-25.

[44] A. Patel, "Heuristics for grid maps," http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#diagonal-distance, 1997, accessed: May 15, 2023.

[45] G. Markkula, O. Benderius, K. Wolff *et al.*, "A review of near-collision driver behavior models," *Human Factors*, vol. 54, no. 6, pp. 1117–1143, 2012.

[46] D. Shiffman, "The nature of code - chapter 6: Autonomous agents," https://natureofcode.com/book/chapter-6-autonomous-agents/#chapter06_section8, 2012, accessed on 2023-04-25.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY