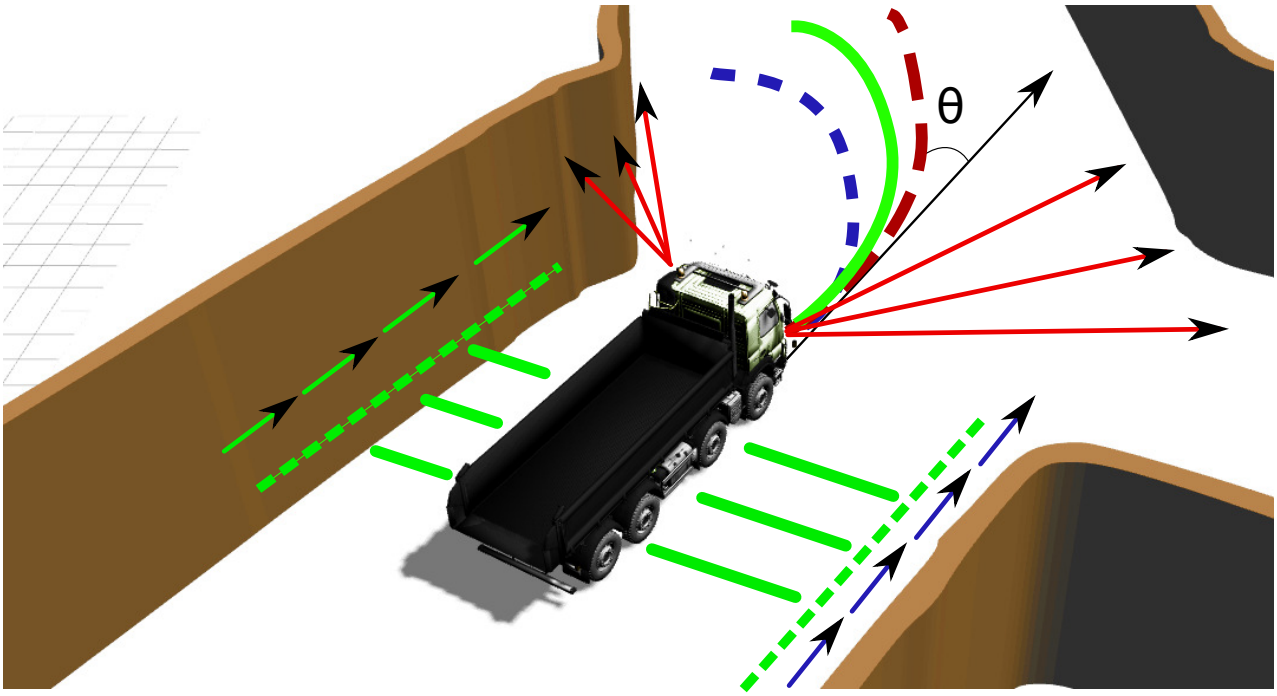
CHALMERS



Rapid close surrounding evaluation for autonomous commercial vehicles

Master's thesis in Applied Mechanics

SHAMIT BAGCHI
EVANGELIA SOULTANI

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2016

MASTER'S THESIS IN APPLIED MECHANICS

Rapid close surrounding evaluation for autonomous commercial vehicles

SHAMIT BAGCHI
EVANGELIA SOULTANI

Department of Applied Mechanics
Division of Vehicle engineering and Autonomous systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2016

Rapid close surrounding evaluation for autonomous commercial vehicles
SHAMIT BAGCHI
EVANGELIA SOULTANI

Master's thesis 2016:40
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle engineering and Autonomous systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Graphical illustration of the navigation algorithms.

Chalmers Reproservice
Göteborg, Sweden 2016

Rapid close surrounding evaluation for autonomous commercial vehicles
Master's thesis in Applied Mechanics
SHAMIT BAGCHI
EVANGELIA SOULTANI
Department of Applied Mechanics
Division of Vehicle engineering and Autonomous systems
Chalmers University of Technology

Abstract

A framework has been developed for local, relative navigation of an autonomous commercial vehicle in confined areas. This enables a vehicle to navigate based only on its immediate surroundings, with the vehicle as the reference. The framework uses a simulator, to simulate the environment of a mine, and a vehicle model. The latter is equipped with 2D LIDAR sensors which allow it to interpret its close surroundings. Based on this information, a navigation algorithm is selected which performs the path planning and moves the vehicle. Four such algorithms have been developed in order to handle two distinct scenarios: navigation in a corridor and navigation at an intersection. The algorithms have been evaluated for safety and accuracy and perform consistently well. The overall root mean square deviation from a reference path is less than 25% of a chosen collision threshold. Throughout the evaluation, a safe distance from all the walls was maintained. In addition, the path generation algorithms were highly efficient with an average execution time of under 2 milliseconds. This work opens up the possibilities for algorithms which can handle additional navigation scenarios and can learn and adapt to its environment.

Keywords: autonomous vehicles, vehicle navigation, local relative path planning, robot operating system, machine learning

Acknowledgements

We would like to thank CPAC systems for giving us the opportunity to carry out this master thesis project. Especially, we would like to express our gratitude to our supervisor, Peter Forsberg, for his valuable guidance, encouragement and for inspiring us during the whole master thesis. In addition, we are grateful to our examiner, Mattias Wahde, for sharing with us his expertise and steering us in the right direction throughout the whole thesis.

Contents

1 Introduction

1.1 Motivation

Research and development in the area of autonomous vehicles is currently seeing a flurry of activity and interest due to the safety, efficiency, productivity and cost benefits involved. An important aspect when developing an autonomous vehicle is navigation. According to Leonard and Durrandt-Whyte [37] navigation can be accomplished by answering three simple questions: *where am I, where am I going and how should I get there?* Based on these questions, a vehicle that navigates autonomously must be able to (i) sense the environment (ii) determine its position within the surroundings, and (iii) generate and follow a path in order to move to the next position. Conceptually, these aspects can be captured in a model that includes the following steps: mapping, localization and path planning.

An autonomous vehicle may navigate based on a predefined map of the entire terrain. Alternatively it may need to incrementally build and update a map of its surroundings based on data from sensors such as cameras, laser scanners, sonars or infrared sensors. This map can be metric (with precise coordinates) or topological (a graph with nodes and links corresponding to places and paths respectively). With respect to this map a vehicle has to find its own location and heading [37]. This process is called localization and can be classified into two main categories: relative and absolute. In the first category a vehicle estimates its relative position and orientation using information obtained from the on-board sensors, while in the second category it uses beacons, landmarks, or satellite signals (GPS) to obtain absolute position co-ordinates.

Once a vehicle localizes itself it generates a path in order to move. This motion has to be collision-free and involves the process of path planning. There are two main path planning approaches: global and local path planning. A global planner requires a vehicle to have prior information or a map of the environment. On the other hand, in local path planning, a vehicle has no map in advance and instead builds, and dynamically updates the information about the environment in real time [8].

Traditionally, the approach followed for autonomous navigation in robotics, is combining a global path planner with absolute localization. However, this approach has certain disadvantages: absolute coordinates may not always be available (for example GPS information inside a mine). Even if they are available it requires a lot of space to store the absolute coordinates of a metric map [69]. Also, a global path planner cannot handle a dynamic, changing environment as it uses a static, predefined map.

1.2 Purpose and scope

This master thesis is a part of a larger project which involves autonomous navigation of a vehicle in a mine. The mining area is represented by a topological map in which nodes correspond to intersections and links between them correspond to corridors. The purpose of this master thesis is to develop a novel technique for close surroundings evaluation of a vehicle and for *local, relative path planning* in a computationally and storage efficient manner. As a result, the scope will be on exploring and implementing different machine learning methods since they can handle dynamic changes in real environments in a robust manner.

1.3 Path planning

Path planning for autonomous vehicles concerns computing an optimal path as a continuous sequence of collision-free segments connecting the initial and goal positions given a set of known or detected obstacles. As mentioned before, depending on the availability of a map, path planning can be of two main types: global and local path planning. Traditionally, a global planner aims at finding a collision free path from the initial position to the goal position of a vehicle. A local planner handles unexpected events such as dynamic obstacles and is generally used together with a global planner [52].

1.3.1 Global path planning

For navigating from a start position to a goal position a map is necessary on which a start and an end point are specified along with the intermediate obstacles. Then a global path planner is used. A global path planner usually generates a path in the following two steps: (i) represents the environment of a vehicle as a graph,

and (ii) performs a graph search to find a sequence of free nodes that connect the start position with the goal position of a vehicle in the graph. In general the choice of the graph depends on which method is used to solve the problem. Three commonly used approaches are the Cell decomposition, the Roadmap and the Potential fields [24].

Cell decomposition
This method partitions the search space of a vehicle into convex regions referred to as cells and constructs a graph in which adjacent cells are linked to each other [7]. Once the cells that correspond to the start and goal position of a robot are determined, the method provides the path as a sequence of obstacle free cells from the start position to the goal position.

Roadmap
This approach creates a map of the search space by capturing the connectivity between free configurations (a complete specification of the position of the robot) as a set of 1-dimensional curves. Examples include Voronoi diagrams and Visibility graphs [33]. Voronoi diagrams are partitions of a plane into regions based on distances from points in a specific subset of the plane. Visibility graphs are graphs of mutually visible locations, where each node in the graph represents a point location, and each edge represents a visible (without an intermediate obstacle) connection between them. Creating such a map is not a trivial task, however once the map is created then generating a feasible path is an easy and fast process.

One of the simplest Roadmap based search algorithms which created the foundations of robotic path planning was developed by Dijkstra [12]. It is based on the idea of finding the shortest path or minimum cumulative edge cost, which could be the physical distance between a start position and a goal position. The algorithm traverses from one node to the next and assigns costs to them based on their distance from the start position. It has the advantage of being complete in the sense that it guarantees the shortest path from the start position to the goal position but it usually searches through the entire graph. Another important algorithm for modern robot path planning is A* [18]. It uses a heuristic estimate to focus the search toward the goal position and can be considered as a combination of Dijkstra and Best First search algorithm [50]. Best first search is an algorithm which explores a graph by expanding the most promising node chosen based on a specified rule. However a disadvantage of these algorithms is that these methods search among all possible paths and this can lead to a combinatorial explosion, slowing down performance.

Potential fields
This method is based on the idea of defining a potential field function using the analogy of gravity [29]. More specifically the vehicle is considered to be a particle which moves towards a minimum height level (its goal) and avoids or moves away from areas with higher height levels (obstacles). The direction of movement is the negative gradient of the potential field. This method is simple to implement and has been widely applied. However, a major disadvantage is that a robot may become trapped in a local minima in the potential field [32]. Several variations have been used to overcome the problem of premature convergence however they are only applicable to limited classes of objects, shapes and configuration spaces [57] [68].

1.3.2 Local path planning

A key requirement in path planning is the ability to handle unexpected events from the surroundings of a vehicle. A global path planner generates a path by relying on a static predefined map. As a result, it is not able to adapt to dynamic changes. Therefore a local path planner that uses sensors for awareness of a vehicle's current, surrounding environment without relying on a predefined map is usually needed. Several local path planning approaches have been proposed, such as the Bug algorithms, the Potential fields and the Probabilistic roadmap.

Bug algorithms
Such algorithms assume only local knowledge of the environment, along with a global goal and consider simple behaviors such as (i) Follow a wall (right or left) (ii) Move in a straight line towards the goal [10][46]. For example a simple Bug algorithm is: head towards the goal, and wall-follow obstacles until you can head toward the goal again. In such a situation, the robot needs to gather information about the environment in real time and update its control laws so as to achieve this. However, using a set of simple predefined behaviours is not very realistic for real world applications in complex environments.

Potential fields
This method has also been used as a local path planner. Khatib [29] uses an artificial potential method to bend locally a given global trajectory around obstacles. The significance of this algorithm is that it allows real time execution. Newman and Hogan [20] use a goal potential function and a braking potential function to drive a manipulator towards its goal position while avoiding obstacles dynamically.

Probabilistic roadmap
Another method which uses a local path planner to generate a path is the Probabilistic roadmap [28] [49]. This method is an extension of the Roadmap method and proceeds in two phases: a learning phase and a query phase. In the learning phase the probabilistic roadmap is obtained by repeatedly sampling collision free configurations of a vehicle and connecting them to other earlier added configurations. Then in the query phase it searches the Roadmap for a sequence of local paths that connect the start position and goal position of a vehicle.

The methods discussed so far try to solve the path planning problem by determining and explicitly handling all possible scenarios in code and thus they are not very robust under unknown, complex environments. An alternative method to overcome this problem is to use the adaptive nature of a variety of machine learning techniques such as artificial neural networks, support vector machines, and genetic algorithms. These techniques can learn from experience in a dynamic environment. They are capable of generating and choosing from a set of learned or trained actions and applying them when necessary. This makes them effective when handling unknown scenarios from the surrounding environment. In addition, such algorithms can lead to faster convergence towards an optimal solution in the problem space.

Artificial neural networks
Artificial neural networks are structures that comprise of a series of learning units with inputs and layers of neurons interconnected among themselves [41]. The inputs are fed in to the input layer and the weights on the connections between the units are adjusted so as to learn the input data based on the expected output data. This is termed as supervised learning. The neural networks can be trained based on training data and the trained network can then be used to do predictions based on new input data. Several attempts have been made on utilizing neural networks for robot navigation and path planning [27] [56].

More specifically, back-propagation, which is one of the most well known supervised learning methods, has been used for gradient based path length optimization with obstacle collision penalty [63]. In another instance [51] a reduced-resolution raw image and range data for road scenes were fed into a 2-layer feed-forward neural network to produce direction output for a mobile robot. In an extension to [51], [43] describes a vision-guided mobile robot navigation system NEURO-NAV, in which the robot does not need a geometric model of the environment since the environment is modeled by the order of appearance of various landmarks and by adjacency relationships. The robot is trained for a reactive behavior that is driven by visual stimuli.

Support vector machines
It is a binary classification method that finds the boundary that separates optimally two classes of a data set in the sense that it maximises the distance between two classes. To achieve the separation the algorithm usually maps the data set to a higher dimensional space where the two classes are linearly separable. In [44] a method to generate a nonlinear smooth path is introduced. The author divides the space into two classes based on the left and right side of a robot and then represents the optimal hyperplane as a collision free path. In [53] this method is extended to detect on-line obstacles using the k-nearest neighbours algorithm. The advantage of the support vector machine method is that it can generate a nonlinear path at low computational cost, in comparison to combinatorial approaches discussed earlier.

Genetic algorithms
Genetic algorithms (GAs) have also been used for path-planning. GAs were originally formulated by John Holland in the 1960s [21]. GAs evolve candidate solutions of a problem represented as bit-strings (chromosomes) using genetically inspired operators such as selection, mutation and crossover. In the case of path planning, candidate solutions (paths) are represented as chromosomes. The necessary step in these algorithms is the determination of the fitness function (optimization criterion). GAs do not require the construction of a graph and hence minimize the computation time for finding a path. In one particular approach [59] GAs were used for local obstacle avoidance (local feasible path) of a mobile robot in a given search space. The method tries to

find an optimal path by setting the objectives as to minimize the length of the path and the number of turns.

Instead of an exclusive global or local path planning approach a combination of these methods can be used in a real-world navigation problem. This combined approach which involves local obstacle avoidance using sensor data and a global topological map for overall navigation will be adopted. The local navigation inside the mine will be performed by a metrical (yet relative) path planning.

1.4 Localization

This section reviews the basic ideas regarding localization in robotics. A described in Section 1.1, localization is the process in which a mobile robot determines its current position and orientation within its surroundings based on sensor readings [37]. It can be divided in two main categories: relative and absolute localization.

Relative localization estimates the robot's position and orientation based on sensors that are placed on the robot. Two commonly used techniques are odometry and dead-reckoning [42]. These methods integrate the kinematic equations of the robot in order to update its position and orientation. The odometry approach uses only wheel sensors to compute the number of wheel rotations of the robot relative to its starting position. The Dead-reckoning approach uses heading sensors in addition to wheel sensors. Even though these approaches are easy to implement, they do not provide accurate estimates over long distances. This occurs due to the fact that (i) wheel rotations are translated into linear displacement relative to the ground surface which reduces the accuracy of estimation in non-planar surfaces, and (ii) noise from wheel sensors is integrated over time. A common solution to the problem of accumulation of error is recalibration [4]. An alternative technique which uses Bayesian inference to predict the position and orientation of a robot is a Kalman filtering [26]. The prediction is a combination of the last estimation and the current measurement.

Absolute localization estimates the robot's position and orientation with respect to a global reference frame using beacons, landmarks, or satellite signals such as GPS [38]. In comparison to relative localization, the position and orientation of the robot is independent of time and location. Therefore, estimation is more accurate as there is no accumulation of error from sensors with distance travelled. However, absolute localization is more costly in terms of computational time and can be used under specific circumstances. For example, localization based on satellite signals is limited when considering indoor areas where the signal is too weak to penetrate the walls of buildings.

A common solution for successful localization is to combine absolute with relative techniques. Relative localization is used with high frequency to update the position and orientation of the robot while absolute localization is used with low frequency to correct errors [5].

This master thesis project is based on the principle that a vehicle does not need to constantly localize itself accurately in order to navigate autonomously. Inside a mine, the problem of localization becomes important only when a vehicle enters and exits intersections and places where it loads and unloads material. As a result, localizing a vehicle within a topological map in which nodes and links represent intersections and corridors, is a sufficient way to solve the problem of localization in this scenario. This project will be making use of this information when developing the local navigation techniques and thus the problem of localization itself will not be addressed here.

1.5 Outline

The remainder of this report is structured as follows. The next chapter aims to give the reader a complete overview of the framework that has been developed. This chapter is then followed by the local, relative path planning approach which introduces and describes the developed algorithms in terms of two scenarios: *navigation in a corridor* and *navigation at an intersection*. Then, a set of criteria is proposed in order to evaluate the algorithms quantitatively. Based on these evaluation methods, the main results are presented and discussed. In the end, the report comes to conclusions and possible ideas for continuation and improvement of the project are proposed.

2 Overview of the framework

The following schematic diagram (Figure 2.1) depicts the framework that has been created in order to achieve local relative path planning. The framework uses a simulator, described in Section 2.1.1 below, to simulate a vehicle and its environment including the associated sensor data. A global path planner performs localization on the basis of a topological map. It then periodically indicates to the local path planner whether the vehicle is in a corridor or at an intersection. Based on that information, an appropriate algorithm is selected for local navigation.

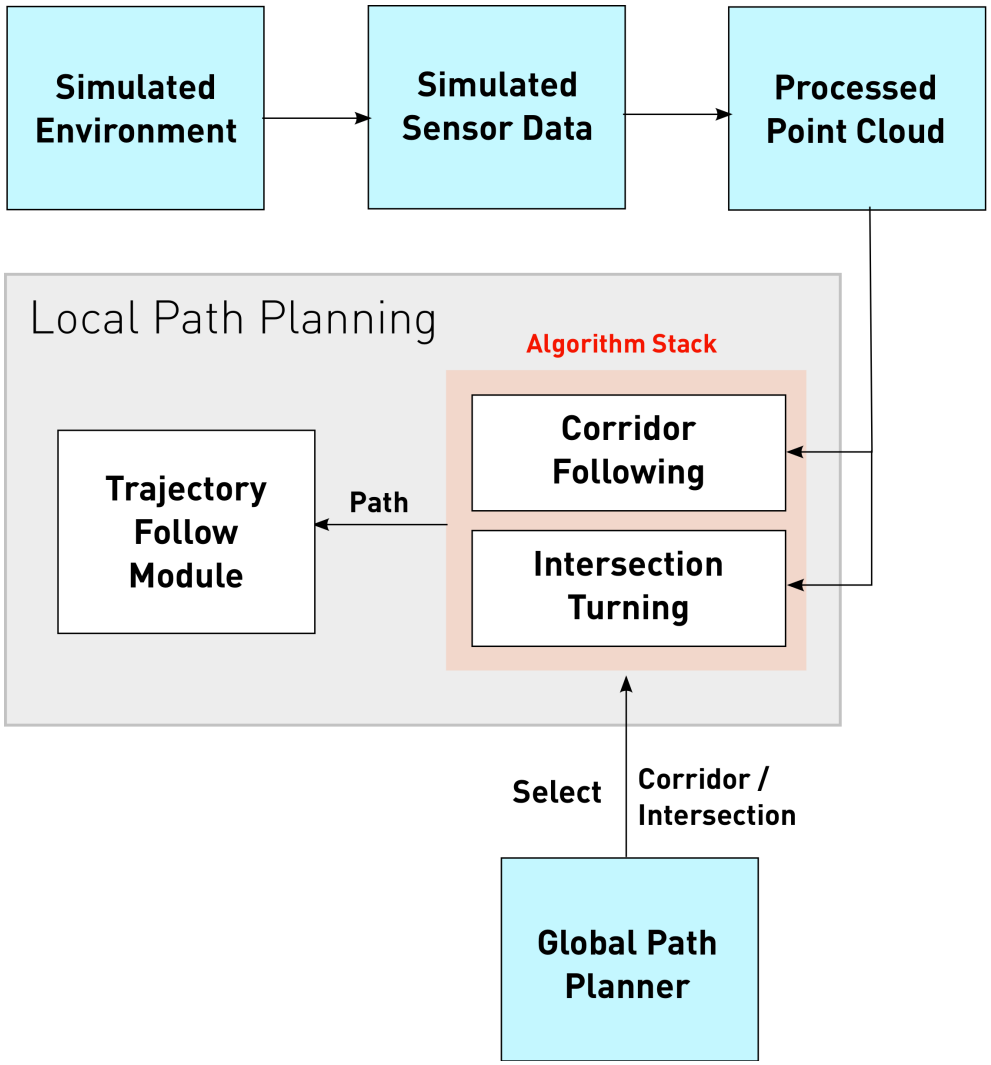


Figure 2.1: *Local path planning framework.*

2.1 Sensor data acquisition and processing

2.1.1 Environment simulator

In this master thesis, the local, relative path planning of an autonomous vehicle has been developed under an open source robotics framework, ROS, using the C++ programming language [55]. The developed approach has been implemented and evaluated in a simulated three dimensional environment of a mine in Gazebo [30] (Figure 2.2). Gazebo is a 3D multi-robot simulation environment which can be used to recreate and simulate the complex world and its behavior. Gazebo provides fine grained control, data visualization, simulation of remote environments that can be integrated with other applications to test robot behaviors and movements.

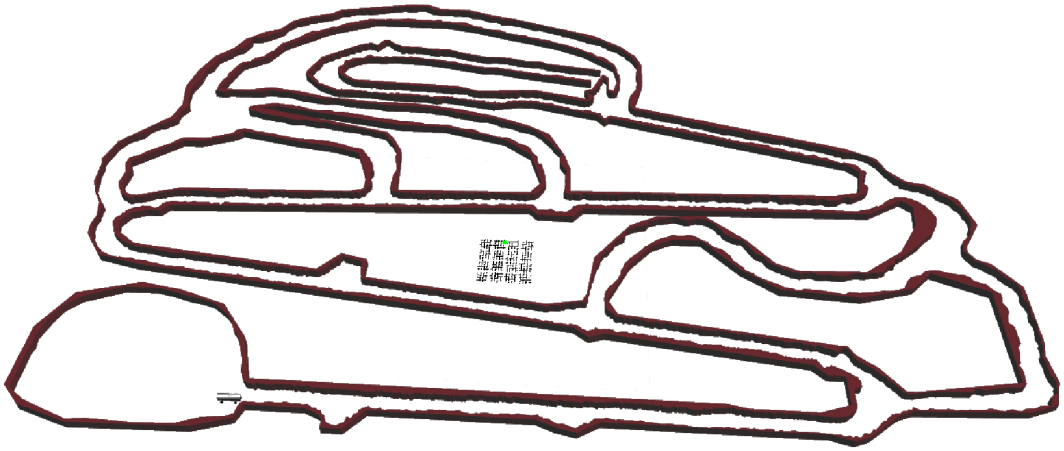


Figure 2.2: *Simulated environment of a mine created in Gazebo.*

2.1.2 Simulated vehicle

A simulated four-wheeled vehicle has been used (Figure 2.3). The vehicle has been equipped (at the four corners) with two dimensional simulated LIDAR sensors to allow it to interpret its close surroundings. LIDAR sensors are laser scanners which provide relative distances and angles from the vehicle to the obstacles in the local surroundings. Such sensors have high level of millimeter-accuracy and work well in both light or dark areas. Therefore, they are appropriate for interpreting the local environment of a mine. Once the sensor data are obtained, they are merged to the middle of the vehicle's rear axle which is considered to be the center of the vehicle.

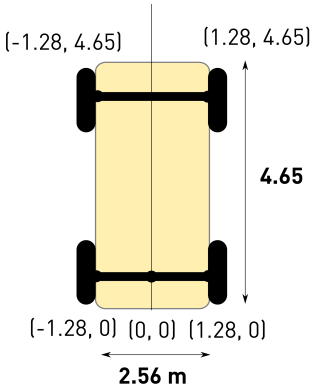


Figure 2.3: *Dimensions of the simulated four-wheeled vehicle. The center of the rear axle is taken as the origin.*

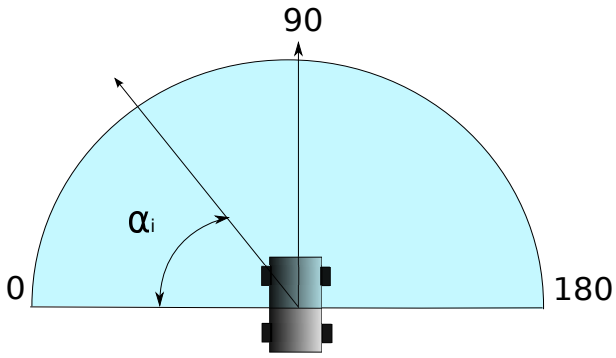


Figure 2.4: *A vehicle as a point of reference obtains sensor data from the upper left and the right quadrants.*

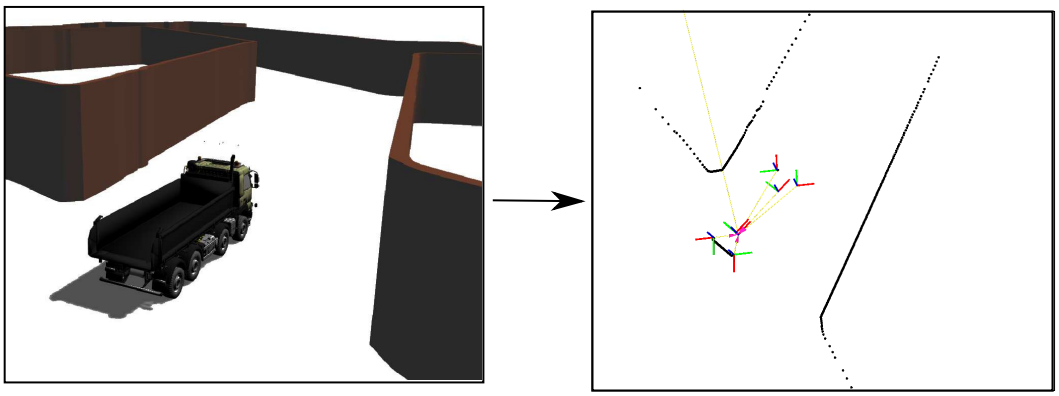


Figure 2.5: *Left panel: a vehicle equipped with 2D LIDAR sensors placed in front of a corridor in the Gazebo simulator. Right panel: the same representation of the vehicle in the RVIZ visualization environment. The dots correspond to the point cloud obtained from sensor readings.*

2.1.3 Point cloud acquisition

A scan from the LIDAR sensors corresponds to a set of distances d_i and angles $\alpha_i \; \forall \; i \; \in \{1, 2, .., 180\}$ relative to the vehicle (Figure 2.4). In order to process the data to generate a local, relative path for the vehicle, the sensor data are converted to a point cloud in the plane, i.e. a set of two dimensional data points $\{(x_i, y_i)\}_{i=1,..,180}$. The point cloud is obtained as follows

$$x_i = x_0 - d_i cos(a_i) \tag{2.1}$$
$$y_i = y_0 + d_i sin(a_i) \tag{2.2}$$

where $(x_0, y_0) = (0, 0)$ is the origin of the point cloud and corresponds to the middle of the rear axle of the vehicle. An illustration can be seen in Figure 2.5.

2.1.4 Noise removal

The point cloud is obtained as a set of data points relative to the vehicle. However laser scanners generate measurements with noise. As a result filtering the point cloud and removing outliers is necessary before generating a local path. This is done using a filter in the Point Cloud Library (PCL) [58] which performs a statistical analysis on each data point's neighborhood. More specifically, for each point it computes the mean distance to all its neighbours within a specified range. Assuming that the resultant distribution of mean distances is Gaussian, it considers as outliers all points whose mean distances are outside a given interval which is defined by the mean and standard deviation of the global distances.

2.1.5 Clustering

The point cloud $\{(x_i, y_i)\}_{i=1,..,180}$ represents a collection of obstacles in the surroundings of the vehicle. These obstacles can be segments of walls in a corridor or at an intersection in the simulated mining area. Therefore in order to generate a path, the point cloud is divided into clusters where each cluster can represent such a segment. More details of the clustering techniques that have been used in this master thesis project will be described in the next chapter.

2.1.6 Vehicle path

Once the sensor data have been acquired, processed into a point cloud and clustered appropriately into segments, the vehicle has relevant information about its local environment. Based on this information a collision-free path is generated. A path consists of one or more poses, where a pose p is

$$p = \big(x, y, z, \alpha, \beta, \gamma\big) \tag{2.3}$$

where (x, y, z) is the next position of the vehicle and (α, β, γ) is its orientation (rotation around x, y and z axes) corresponding to roll, pitch and yaw respectively. The vehicle uses this path to move without collisions. The movement is based on the Ackermann steering geometry [60]. The basic principle is that the rear wheels are fixed while the front wheels are allowed to turn. The steering is modelled by a steering angle (γ in equation 2.3) of the front wheel (Figure 2.6). The next chapter describes in detail the algorithms developed for the process of local, relative navigation.

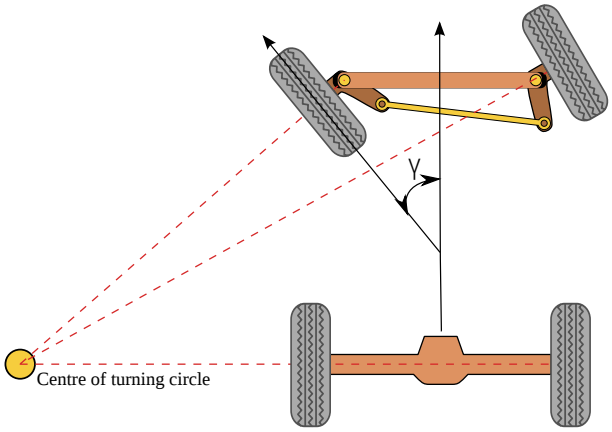


Figure 2.6: *A vehicle model based on the Ackermann steering principle. Source: Wikimedia Commons. Ackermann turning.svg. Illustration of the Ackermann steering geometry, when turning. Note that each wheel, front and back, has an axle alignment that is a radius of a common circle. As the rear wheels are fixed, this circle's centre must lie on the axle line of the rear axle. The figure is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.*

3 Local relative path planning approach

This chapter focuses on the algorithms that have been developed in order to generate short, local, relative paths. At each time step the vehicle obtains:

1. A point cloud of the local surroundings provided by 2D LIDAR sensors that are placed on the vehicle.
2. An indication whether the vehicle should navigate in a corridor or at an intersection. This indication is provided by a global path planner using a topological map.

The key idea in this master thesis project is that the vehicle is able to navigate based only on this information. Therefore, the developed approach requires neither further knowledge of the vehicle's absolute localization nor a complete metric map of the mining area. This results in a solution which is efficient both computationally and in terms of memory, allowing real time execution.

The local path planning approach is shown in Figure 3.1. The algorithms that have been developed are divided into two main categories: ***navigation in a corridor*** and ***navigation at an intersection***. Each category corresponds to a navigating behavior. At each time step a behavior is selected (based on the indication; see above) and an appropriate algorithm is used for navigation.

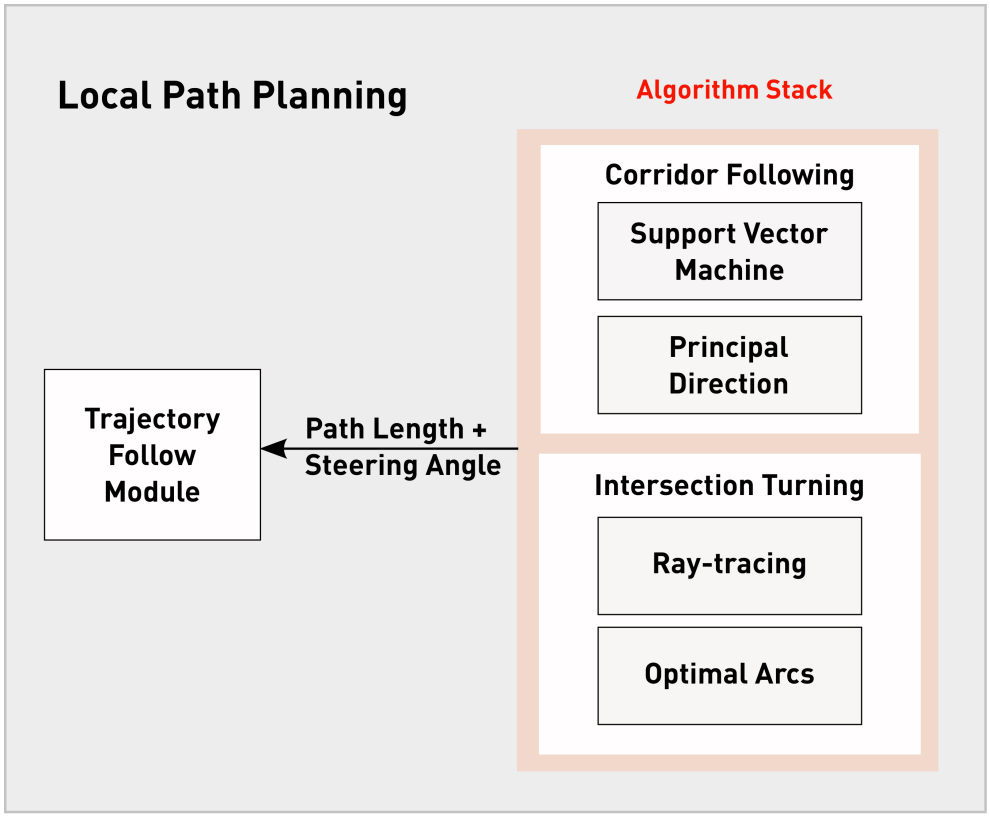


Figure 3.1: *Local path planning approach.*

3.1 Navigation in a corridor

This section consists of two main algorithms that capture the behavior of navigating in a corridor. The input to these algorithms is a point cloud and the output is a short, relative, linear path in the form of a steering angle γ and a path length l.

3.1.1 Corridor algorithm - support vector machine

Motivation

The algorithm is based on the principle that a vehicle should navigate in a corridor while keeping a safe distance from the walls. Inside a mine, corridors are usually narrow. Therefore an ideal path for the vehicle would be a path that has equal distance from both walls to maximize safety. Inspired by this fact, the algorithm uses a support vector machine (SVM) which is a machine learning method, to navigate in the corridor. A brief introduction to the theory of SVMs is presented below [66], [9].

Support vector machine

SVMs is a machine learning method mainly used for binary classification of data based on the concept of margin maximization. In general an SVM classifier is provided with a training data set $S = \{\bar{x}_1, \bar{x}_2, ..., \bar{x}_n\}$, in which each data point $\bar{x}_i$ has a label t_i

$$t_i = \begin{cases} -1, & \text{if } x_i \text{ belongs to class 1} \\ 1, & \text{if } x_i \text{ belongs to class 2} \end{cases} \tag{3.1}$$

The task of an SVM classifier is to find the decision boundary (in the form of a line or curve) that maximizes the perpendicular distance (margin) from the boundary to the closest points in each class. Once this optimization task is complete, the optimized parameters that determine the boundary are used to classify the new set of data. Depending on whether the data are linearly separable or not, an SVM classifier uses a linear or non-linear approach respectively. In the implementation of the algorithm a linear SVM has been used to generate local, relative paths and therefore it will be explained in detail.

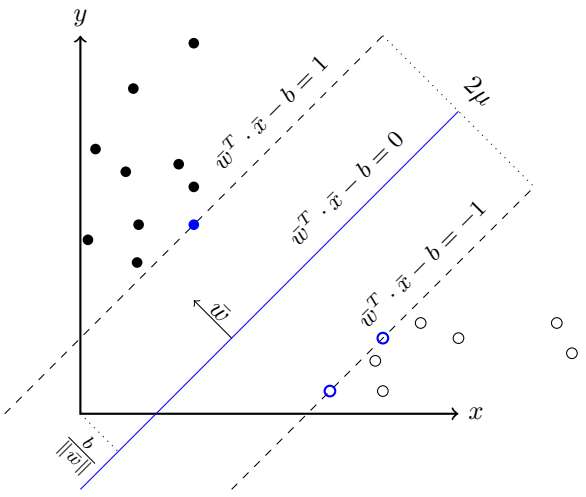


Figure 3.2: *A linear SVM classifier.*

A linear SVM classifier finds the optimal boundary (Figure 3.2) that linearly separates the two classes of data.

The boundary is of the form

$$w_1 x + w_2 y - b = 0, \quad \bar{w} = (w_1, w_2), \quad \forall\ \bar{x} = (x, y) \in R^2 \tag{3.2}$$

and satisfies the condition

$$t_i(w_1 x + w_1 y - b) \geq 1, \quad \forall \quad \bar{x}_i = (x, y) \in S,\ t_i = \pm 1 \tag{3.3}$$

The margin μ is geometrically the perpendicular distance from the boundary (Eq. 3.2) to the closest points in each class (Figure 3.2). It can be expressed as

$$2\mu = \frac{1}{||\bar{w}||} \bar{w}^T (\bar{x}_r - \bar{x}_l) \tag{3.4}$$

where

$$||\bar{w}|| = \sqrt{w_1^2 + w_2^2} \tag{3.5}$$

and $\bar{x}_l, \bar{x}_r$ are two data points close to the boundary which belong to class 1 and 2 respectively. These points satisfy the following equations

$$\bar{w}^T \bar{x}_r - b = 1 \tag{3.6}$$

$$\bar{w}^T \bar{x}_l - b = -1 \tag{3.7}$$

The margin μ can be calculated by subtracting Eq. 3.7 from 3.6,

$$(\bar{w}^T \bar{x}_r - b) - (\bar{w}^T \bar{x}_l - b) = 2 \tag{3.8}$$

$$\bar{w}^T (\bar{x}_r - \bar{x}_l) = 2 \tag{3.9}$$

$$\frac{\bar{w}^T (\bar{x}_r - \bar{x}_l)}{||\bar{w}||} = \frac{2}{||\bar{w}||} \tag{3.10}$$

and then by substituting Eq. 3.4 in 3.10

$$2\mu = \frac{2}{||\bar{w}||} \tag{3.11}$$

$$\mu = \frac{1}{||\bar{w}||} \tag{3.12}$$

The task of maximizing the margin μ to obtain the optimal set of $\bar{w} = (w_1, w_2)$ and b that determine the boundary is equivalent to the task of minimizing $||w||$. As a result the optimisation task can be primarily formulated as

$$\text{argmin}_{\bar{w}}\big(\frac{1}{2}||\bar{w}||^2\big) \tag{3.13}$$

subject to the constraints

$$t_i(\bar{w}^T\bar{x_i} - b) \geq 1, \quad \forall \bar{x_i} \in S \tag{3.14}$$

A solution to this minimization problem can be given by substituting the constraints (Eq. 3.14) in Eq. 3.13 using the Lagrange multipliers $a_i \ \forall i = 1, .., |S|$. Therefore, the problem can be expressed in a dual form as

$$\text{argmax}_{a_1,\ldots,a_n}\big(\sum_{i=1}^{|S|} a_i - \frac{1}{2}\sum_{i,j=1}^{|S|} a_i a_j t_i t_j \bar{x}_i^T \bar{x}_j\big) \tag{3.15}$$

subject to the constraints

$$\sum_{i=1}^{|S|} a_i t_i = 0, \ a_i \geq 0 \ \forall i = 1, .., |S| \tag{3.16}$$

The solution to this optimization problem is unique and is the set of values $\{a_1, .., a_{|S|}\}$ that maximize the objective quadratic function (Eq. 3.15) . The data points $\bar{x}_i$ which have non-zero Lagrange multiplier a_i, belong to either one of the boundaries (Figure 3.2)

$$\bar{w}^T\bar{x} - b = 1 \tag{3.17}$$
$$\bar{w}^T\bar{x} - b = -1 \tag{3.18}$$

and are called support vectors. They are the only data points that contribute to the calculations in order to determine the parameters. Once the solution $\{a_1, .., a_{|S|}\}$ is obtained, the parameters $\bar{w}, b$ can be computed as follows

$$\bar{w} = \sum_{i=1}^{|S|} a_i t_i \bar{x_i} \tag{3.19}$$
$$b = t_k - \bar{w}^T\bar{x_k}, \ \bar{x_k} \ \text{ a support vector} \tag{3.20}$$

and the decision boundary is obtained. The classification of a new data point $(x^1_{\text{new}}, y^2_{\text{new}})$ is based on the parameters of the boundary $\bar{w}, b$, and is done by assigning a label t_{new} as follows

$$t_{\text{new}} = \begin{cases} -1, & \text{if} \quad \bar{w}\bar{z}_{\text{new}} - b > 0 \\ 1, & \text{if} \quad \bar{w}\bar{z}_{\text{new}} - b \leq 0 \end{cases} \tag{3.21}$$

Implementation

The steps of the algorithm are illustrated in Table 3.1. The main idea is to cluster the point cloud into two wall segments and then use them to train a linear SVM classifier. The result is a decision boundary which separates maximally the two wall segments. The parameters of this decision boundary determine a path for the vehicle. The clustering of the point cloud and the training of a linear SVM is done under DLIB framework [13]. More specifically, spectral clustering has been chosen to obtain the left and right wall segments [45]. More details of this method are explained in Table 3.2.

Input: Point cloud obtained from a scan from the LIDAR sensors.

1. Divide the point cloud into clusters where each cluster represents a left and right wall segment respectively.
2. Use the two clusters to train a linear Support Vector Machine classifier.
3. Obtain the optimized parameters (w_1, w_2, b) that determine the boundary (equation 3.2) which separates maximally the two wall segments.
4. The slope of this boundary determines the steering angle of the vehicle.

Output: The steering angle γ.

Table 3.1: Corridor algorithm: support vector machine.

Initialize: The number of clusters M.
Input: Point cloud (N data points) obtained from a scan from the LIDAR sensors.

1. Construct an $N \times N$ similarity matrix, W from the point cloud.
2. Compute the $N \times N$ Laplacian matrix, L = DW, where D is the degree (diagonal) matrix.
3. Compute the k eigenvectors of L (Each eigenvector $\bar{v}_i$ is an $N \times 1$ column vector).
4. Create a matrix V which contains the eigenvectors $\bar{v}_1 ... \bar{v}_k$ as columns.
5. Consider each row in matrix V as a point in k dimensional space.
6. Cluster this set of points using k-means clustering as follows
   (a) Generate randomly M centroids $\bar{c}_1, \bar{c}_2, ..., \bar{c}_M\}$ in the k dimensional space each one corresponding to a cluster.
   (b) Calculate the euclidean distance between each data point and the centroids.
   (c) Assign each point to that cluster for which the distance from the centroid is the minimum.
   (d) Update the centroids $\{\bar{c}_1, \bar{c}_2, ..., \bar{c}_M\}$ using the formula: $\bar{c}_i = \frac{1}{k_i} \sum_{i=1}^{k_i} \bar{x}_i$, where $\bar{x}_i$ is a point in the i^{th} cluster and k_i is the total number of point in the cluster.
   (e) Update the distances between each data point and the centroids.
   (f) Repeat until no more reassignment is required.
7. Repeat until a maximum number of iterations is reached.

Output: The data in clusters.

Table 3.2: Spectral clustering.

3.1.2 Corridor algorithm - principal direction

Motivation

In two dimensional space, walls can be represented as line segments. The algorithm makes use of this fact and clusters the point cloud into such segments. Then it fits a line to each cluster (Figure 3.3). The collection of line segments creates a local representation of the environment. Each line has a slope which can be considered as a local principal direction of the surroundings. Based on this information the algorithm computes the vehicle's principal direction (steering angle). The steps of the algorithm are shown in Table 3.3.

Implementation

The line segmentation of the point cloud is done under the PCL framework [58]. Although there are several line extraction algorithms [47], RANSAC (Random Sample Consensus) was selected due to its simplicity and robustness. RANSAC is an iterative algorithm that generates candidate solutions by sampling data points and estimating the underlying line parameters. More details of the RANSAC algorithm can be found in Table 3.4.

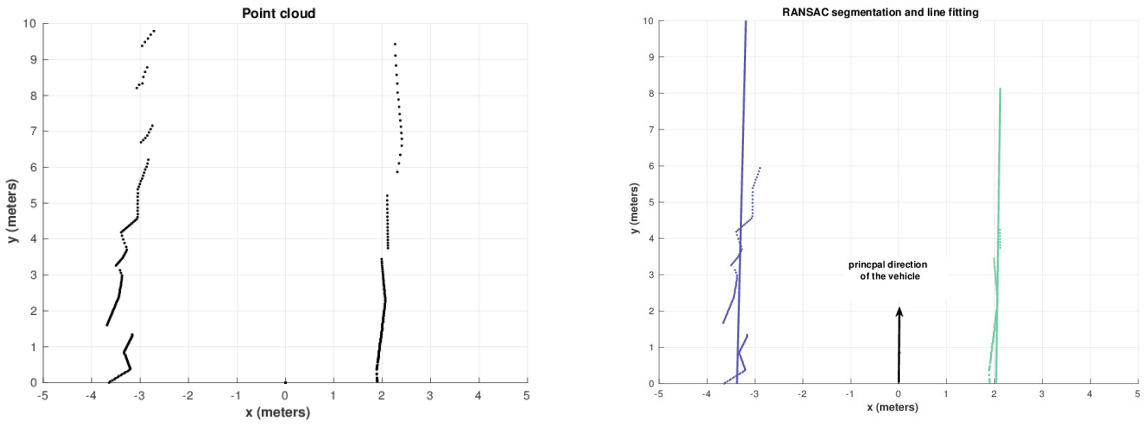


Figure 3.3: *Left panel: A point cloud obtained from a scan in a corridor in the Gazebo simulator (details in Chapter 2). The origin corresponds to the center of the vehicle's rear axle. Right panel: Line segmentation of the point cloud using RANSAC algorithm. The black arrow corresponds to the principal direction of the vehicle as described in Section 3.1.2.*

.

Input: Point cloud obtained from a scan from the LIDAR sensors.

1. Divide the point cloud into two line segments representing a left and right wall respectively.
2. Fit lines to each segment.
3. Compute the set of local principal directions based on the slope of line segments.
4. Average this set to obtain the vehicle's principal direction in the form of a steering angle.

Output: The steering angle γ.

Table 3.3: Corridor algorithm: principal direction.

Initialize: Distance threshold from the points to the line.
Input: Point cloud obtained from a scan from the LIDAR sensors.

1. Sample uniformly two points from the point cloud.
2. Fit a line to the sampled points.
3. Compute the distances from all points to the line.
4. Compute and count inliers (points that lie on the line) based on the distance threshold.
5. Repeat previous steps until the number of inliers is sufficiently large compared to the point cloud.

Output: The parameters that determine the line (a point and a direction).

Table 3.4: RANSAC algorithm.

3.2 Navigation at an intersection

Navigation at an intersection is more complex than navigation in a corridor as there are multiple exits going out from an intersection. When a vehicle approaches an intersection, a specific exit must be chosen. This information is obtained from a global path planner. The implementation considers T-junctions, i.e. intersections with only two exits: left and right. The following section discusses two algorithms for navigating at an intersection.

3.2.1 Intersection algorithm - ray tracing

Motivation

At an intersection there are several obstacle-free directions as there are many exits leading out from an intersection. Therefore, an arc of 180 degrees can be scanned to find the obstacle-free headings.

Ray tracing

Ray tracing is a method which projects rays (line segments) at a scene from a given point and finds where these rays intersect with obstacles. The algorithm uses this method to find the next pose in the path that a vehicle must follow. It generates rays at intervals of a small chosen angle (spanning from 0 to 180 degrees) with the center of the vehicle's rear axle as origin. The algorithm then determines the obstacle-free headings by finding the rays which do not intersect with any obstacle in the vicinity of the vehicle (Figure 3.4).

Implementation

The steps of the algorithm can be illustrated in Table 3.5. The line segmentation of the point cloud is obtained using RANSAC algorithm (Table 3.4) under the PCL framework.

Input: Point cloud obtained from a scan from the LIDAR sensors, indication of left or right turn.

1. Cluster the point cloud into obstacle line segments using RANSAC algorithm (Table 3.4)
2. Generate rays in the form of lines between 0 and 180 degrees at 5 degree interval. The origin is the center of the rear axle of the vehicle.
3. Find the intersection points of the rays with the obstacles (line segments). If the intersection points are within a specified range (in the vicinity of the vehicle), the corresponding direction is considered blocked. The rest of the headings represent the obstacle-free directions.
4. Obtain two sets of obstacle-free directions corresponding to left and right turn. Each set consists of consecutive rays.
5. Choose one of the sets based on the indicated direction - left or right.
6. Average the chosen set of direction to compute the steering angle.

Output: The steering angle γ.

Table 3.5: Intersection algorithm: ray-tracing.

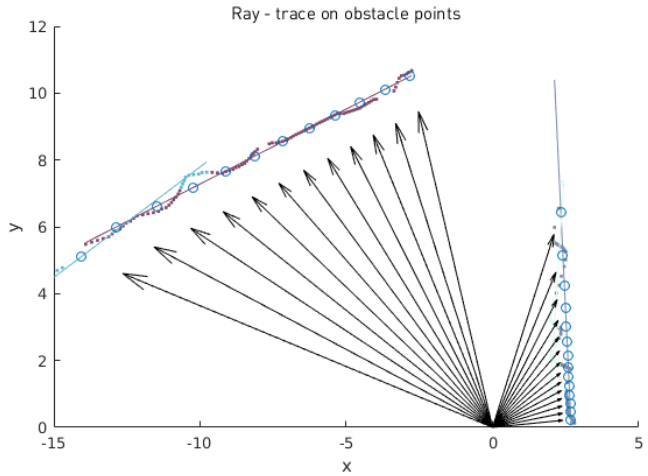


Figure 3.4: *Ray tracing for finding obstacle free heading with the vehicle's center of rear axis as the origin. Black arrows indicate the blocked headings. The obstacle-free directions are empty.*

3.2.2 Intersection algorithm - optimal arc

Motivation

At an intersection when a vehicle has to take a turn (left or right) the steering angle becomes an important parameter. The exact steering angle and the vehicle's turn radius need to be determined to avoid collisions. A possible way to find the appropriate steering angle is to project theoretically, different arc paths, each of which corresponds to a steering angle. Then evaluate the arcs to decide the optimal collision-free path. These arcs can be calculated using a bicycle model which is the basis for the implementation.

Bicycle Model

A simplified bicycle model is considered for simulating the movement of the vehicle. Instead of two wheels in the front and two in the rear, the model considers one wheel in the front and one in the rear, hence the name

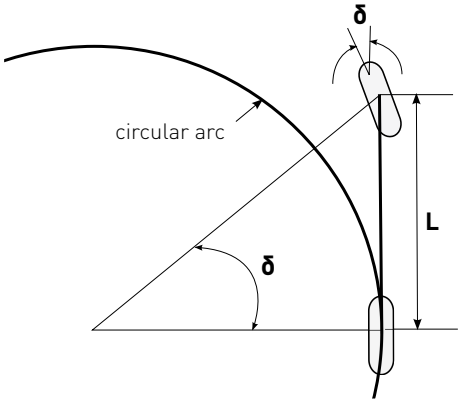


Figure 3.5: *General bicycle model [60].*

bicycle model. The relationship between the front wheel steering angle and the curvature that the rear axle will follow can be expressed using a simple geometric equation [60]:

$$\tan(\delta) = \frac{L}{R} \tag{3.22}$$

where

- δ: Steering angle of the front wheel.
- L: Distance between the front axle and the rear axle.
- R: Radius of the arc that the center of the rear axle of the vehicle will follow.

An illustration of the bicycle model can be found in Figure 3.5. This model approximates the motion of a vehicle reasonably well at low speeds and moderate steering angles.

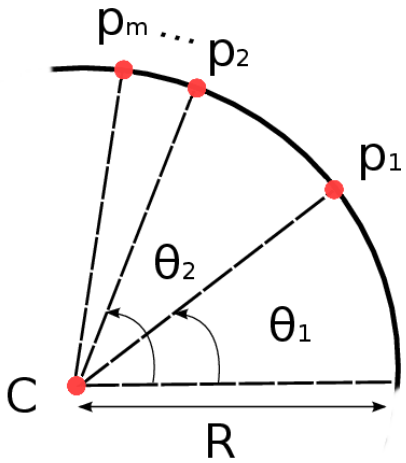


Figure 3.6: *The figure illustrates an arc of radius R and center C corresponding to a left turn for the vehicle. The points $p_1, p_2, ..., p_m$ on the arc are generated based on the algorithm in Table 3.6 at angles $\theta_1, \theta_2, ..., \theta_m$ respectively.*

Implementation

The steps of the algorithm are shown in Table 3.6. The segmentation is done using RANSAC algorithm (Table 3.4) under the PCL framework.

Input: Point cloud obtained from a scan from the LIDAR sensors, indication from the global path planner whether the vehicle should take a right or left turn at the intersection.

1. Cluster the point cloud into line segments representing obstacles, using RANSAC algorithm (Table 3.4).
2. Generate a positive or negative set of steering angles, $\delta_i \quad i \in \{1, ..., \text{number of angles}\}$ based on the indication; see above.
3. For each steering angle δ_i
   (a) Calculate the radius R_i , $R_i = \frac{L}{\tan(\delta_i)}$ that defines the arc along which the center of the rear axle of the vehicle will travel.
   (b) Generate $p_1, ..., p_m$ points on the arc as illustrated in Figure 3.6.
   (c) Create line segments S_j between consecutive points on the arc p_j, p_{j+1} where $j \in \{1, ..., m-1\}$.
   (d) Compute the length l_j of each segment S_j.
   (e) Loop over S_j segments on the arc and check whether there is an intersection between S_j and any of the obstacles in the vicinity of the vehicle. Stop as soon as one of the line segments S_{stop} is intersecting with an obstacle.
   (f) Assign a fitness value f_i, based on the cumulative obstacle-free arc length, to each steering angle δ_i , $f_i = \sum_{j=1}^{S_{\text{stop}}} l_j$.
4. Select the optimal steering angle γ that corresponds to the highest fitness value.

Output: The steering angle γ.

Table 3.6: Intersection algorithm: optimal arc.

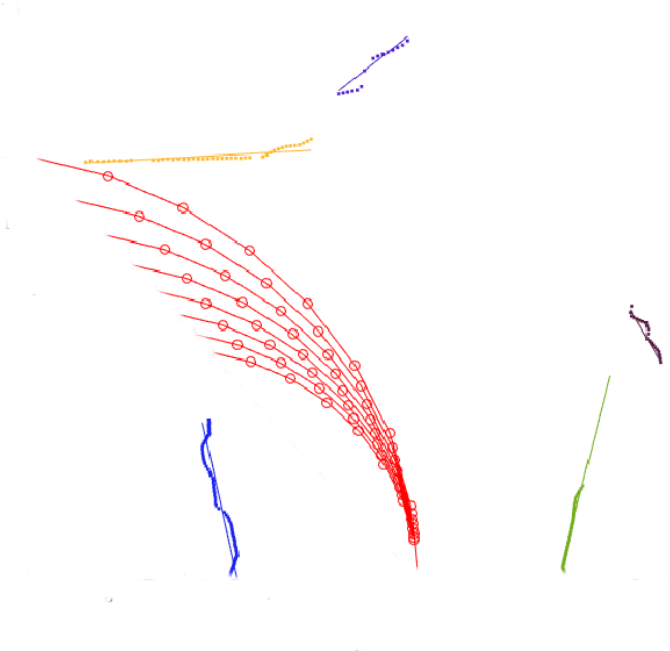


Figure 3.7: *Multiple arcs are evaluated to determine the optimal obstacle-free arc (with longest arc length) that a vehicle should follow at an intersection, in order to turn left in this instance. Each arc corresponds to a steering angle.*

4 Evaluation methods

When developing algorithms for any real world application, one key objective is to ensure that the algorithms are accurate, fast and computationally efficient. The requirements when developing path planning algorithms are no different. Therefore, it is important to test and compare the algorithms in order to qualitatively and quantitatively evaluate their performance. It is necessary that the functionality of all components of the framework is evaluated in a simulated environment that is as close to reality as possible.

4.1 Setup

As discussed in Chapter 2, the framework for the local path planning uses a simulator, Gazebo, to simulate a vehicle and its environment (Figure 2.1). In order to evaluate the performance of the local path planning approach a specific test arena has been selected (Figure 4.1).

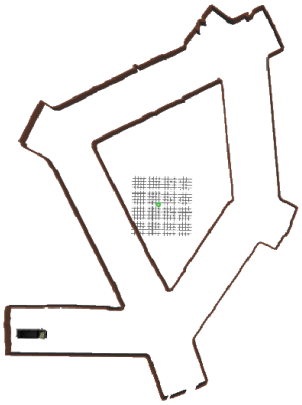


Figure 4.1: *Simulated test arena of a mine in the Gazebo simulator.*

4.1.1 Implementation using ROS

The implementation of the local path planning approach is done under the open source Robot Operating System (ROS) [55]. It is a collection of libraries and tools that help simplify the task of building robot applications. Two important concepts of ROS are *nodes* and *topics*. A node is a process that performs some form of computation. Nodes communicate with each other by streaming messages. A topic is a name that is used to identify the content of a message. Therefore, a node can send a message (*publish to a topic*) as well as receive a message that is published by another node (*subscribe to a topic*). This is illustrated in Figure 4.2. The framework that has been developed consists of such a collection of nodes as can be seen in Figure 4.3.

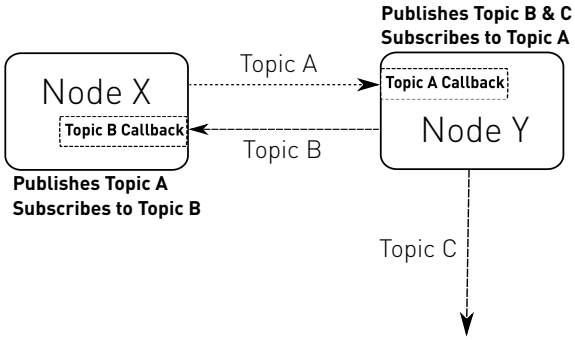


Figure 4.2: *An illustrative diagram for ROS nodes and topics. A node X communicates with a node Y. The nodes X and Y publish the topics A and B, and subscribe to the topics B and A, respectively. This is done via callback functions which are called every time an associated topic is published (usually at a specified frequency).*

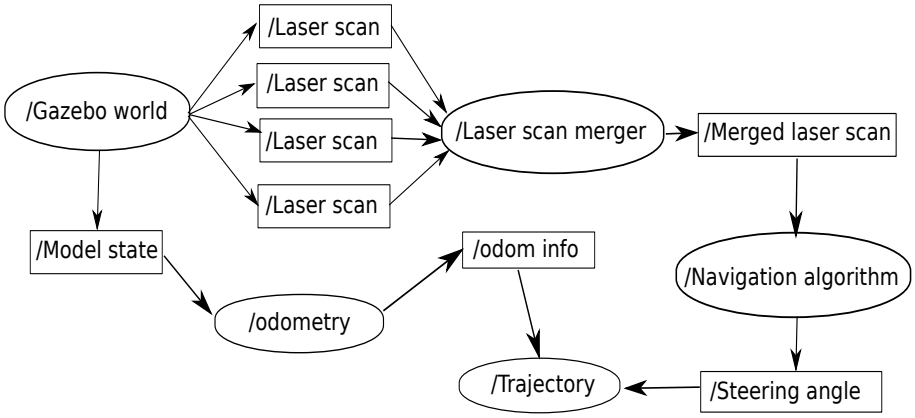


Figure 4.3: *A collection of nodes and topics that has been created for the implementation of the local path planning. The elliptical shapes correspond to nodes while the rectangular shapes correspond to topics. The arrows indicate the links between nodes and topics.*

4.2 Evaluation criteria

In order to evaluate the algorithms, first and foremost a set of evaluation criteria must be formulated. The criteria aim to answer the following questions (i) *are the navigation algorithms that have been developed performing as intended (i.e. are they collision-free etc.)?* and (ii) *how long does an algorithm take to complete a navigation behavior?* A schematic of the evaluation criteria is shown in Figure 4.4. They have been divided into two modes: online and offline. The online mode implies that an algorithm is being evaluated while it is running, i.e. during run-time. The offline mode implies that an algorithm has completed execution and then its overall performance is compared with an expected outcome. In the offline mode the algorithms have been evaluated in terms of *accuracy*, while in the online mode they have been evaluated in terms of *safety* and *temporal performance*. The definitions of the criteria are presented below.

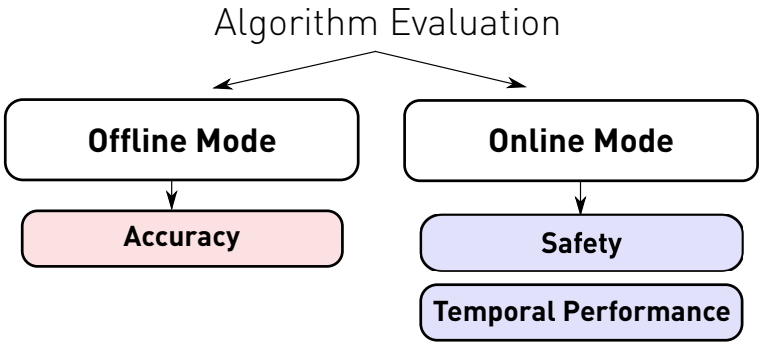


Figure 4.4: *Criteria for evaluation of the developed algorithms as discussed in Section 4.2.*

1. **Accuracy**: concerns how well the algorithm's functionality can approximate a manually driven reference path.
2. **Safety**: concerns the minimum safe distance that the vehicle maintains from all obstacles at any time.
3. **Temporal performance**: concerns the time taken for the algorithm to execute and produce an output.

4.2.1 Accuracy

To evaluate the performance of the algorithms in terms of accuracy, an offline mode has been chosen. Based on this mode, each algorithm is executed in the test arena for a number of runs and then it is compared to a *reference trajectory* which is recorded by driving the vehicle manually with a joystick. The steps of the procedure are described below.

Test procedure

1. Obtain a reference trajectory that a vehicle should follow when it navigates in the simulated test arena (Figure 4.5) as follows:

   (a) Drive the vehicle manually using a joystick.

   (b) Save the pose (absolute position and orientation) of the vehicle while it moves using the odometry technique that localizes the vehicle.

   (c) The sequence of all recorded poses together define the reference trajectory.

2. Execute each corridor or intersection algorithm for a specified number of runs. For each run:

   (a) Initialize the vehicle at a position and orientation. The position can be in a corridor or at an intersection in the test arena based on the navigation behavior of the algorithm.

   (b) Save the poses of the vehicle while it moves.

   (c) Stop when a vehicle exits the corridor or intersection.

   (d) Obtain the current trajectory as the sequence of recorded poses.

3. For each corridor or intersection algorithm:

   (a) Obtain a set of recorded trajectories.

   (b) Carry out a statistical analysis of the set of trajectories based on the measure discussed below.

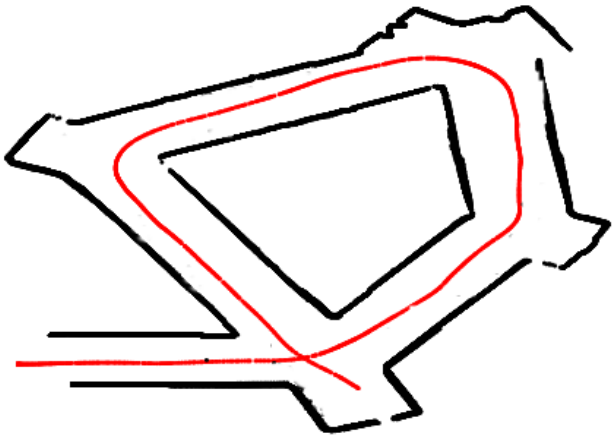


Figure 4.5: *Reference trajectory recorded by driving with a joystick in test arena (Figure 4.1) in the Gazebo simulator. The wall segments are obtained from the simulated LIDAR scans.*

Measure

- Root mean square distance (RMSD) from a trajectory to the reference trajectory.

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n} ||y_i - \hat{y}_i||_2^2} \tag{4.1}$$

  where n is the number of poses of the trajectory, y_i is the i^{th} pose of the trajectory, $\hat{y}_i$ is the i^{th} pose of the reference trajectory and $||\cdot||_2$ is the distance (norm) between two poses.

4.2.2 Safety

To evaluate the performance of the algorithms in terms of safety, an online mode has been chosen. Each algorithm has been executed for a number of runs. During the execution, the minimum distance from the vehicle to the surrounding obstacles is calculated. The four corners of the vehicle (Figure 2.3) have been used as reference points. The minimum distance D_s is computed as

$$D_s = \min\{\mathrm{D_i^j}\} \;\; \forall\, \mathrm{i} \in \{1,2,3,4\},\; \forall\, \mathrm{j} \in \{1, ..., \textit{number of line segments}\} \tag{4.2}$$

where D_i is the minimum distance from the i^{th} corner to the j^{th} obstacle line segment.

4.2.3 Temporal performance

Temporal evaluation is an online evaluation technique which helps find the amount of time or CPU cycles needed to execute an algorithm. Based on this information, the developed algorithms can be compared and ranked. The steps of this procedure are presented below and an illustration is shown in Figure 4.6.

Test procedure

A topic publishes laser scan data with a frequency of 10 Hz via a callback function i.e. a piece of executable code that is passed as an argument to other code, which is expected to execute the argument at some convenient time. The invocation could be immediate as in a synchronous callback, or at a later time as in an asynchronous callback. This callback function in turn invokes one of the algorithms. The process is as follows:

1. Run each algorithm a specified number of times by keeping a counter in the callback function.
2. Insert test probes in the functions (measuring points) to see how long a specific functionality takes to execute.
3. Capture the parameters as shown below for each algorithm for every run.

Measure

Execution time for clustering and path generation functionality.

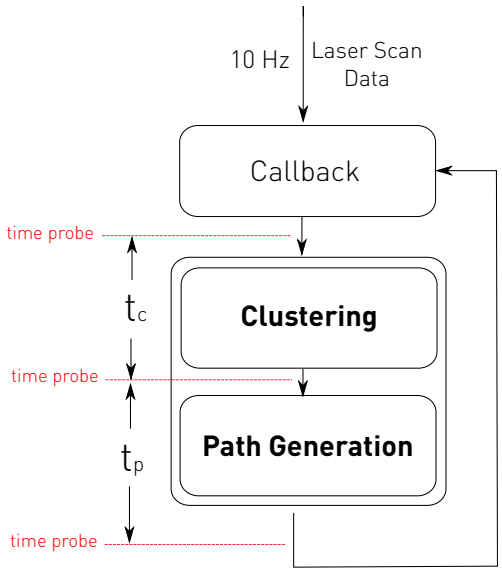


Figure 4.6: *Method for temporal evaluation and comparison of the algorithms based on time probes in callback and ROS node functions. Here t_c is the time required to execute the clustering and t_p for the path generation step.*

The next chapter presents the results that have been obtained based on the evaluation criteria.

5 Results and discussion

This chapter quantitatively evaluates the performance of the navigation algorithms presented in Chapter 3: *support vector machine, principal direction, ray-tracing* and *optimal arc*. As explained in Chapter 3, the algorithms deal with different parts of navigation: the two former concern corridor navigation and the two latter navigation at an intersection. Thus, the arena has been divided into corridor zones: C1, C2, C3, C4, C5 and intersection zones: I1, I2, I3, I4. This is illustrated in Figure 5.1.

Based on the navigation behavior, each algorithm has been executed at a corridor or at an intersection zone for a specified number of runs. To automate this procedure, a test script has been created. The script sets and resets the initial pose (position and orientation) of the simulated vehicle allowing small perturbations. Then, the algorithm is allowed to run until the vehicle crosses the zone from which it has been initiated.

The performance of the algorithms has been evaluated based on three main criteria (i) accuracy, (ii) safety, and (iii) temporal performance, as discussed in Chapter 4. The following sections present the results that have been obtained.

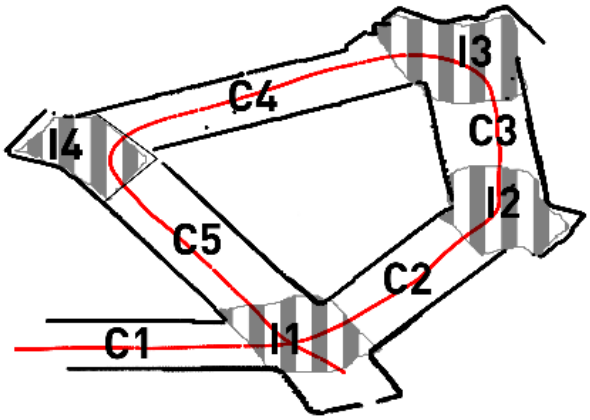


Figure 5.1: *Simulated test arena of a mine in the Gazebo simulator. The zones C1, C2, C3, C4, C5 and I1, I2, I3, I4 indicate the areas in which navigation in corridors and intersections takes place respectively.*

5.1 Accuracy

This section presents a statistical analysis of the data that have been recorded while executing the algorithms at the corridor and intersection zones of the test arena. The data are in form of poses i.e. positions and orientations, and the analysis is based on the root mean square distance (RMSD) discussed in Chapter 4, Section 4.2.1.

Corridor algorithms

The corridor algorithms, *support vector machine* and *principal direction* have been tested in the C1, C2, C3, C4 and C5 zones. Each algorithm has been executed for 100 independent runs. For each run, the RMSD per zone has been computed and the results are shown in Figure 5.2 in the form of a violin plot i.e. a combination of a box plot and a probability density function [19]. Such a type of plot is useful when visualizing the distributions of different sets of data.

From Figure 5.2 it can be inferred that the *support vector machine* algorithm has more variations in RMSD than the *principal direction* algorithm. That means that the *support vector machine* algorithm is more susceptible to deviations when small perturbations of the initial pose of the vehicle are considered. In that sense, the *principal direction* algorithm is more stable.

Based on the mean RMSD, the overall performance of the *principal direction* algorithm is slightly better than that of the *support vector machine*. However, the RMSD for both algorithms does not reach values higher than five metres. For this specific test arena, this is considered as a successful outcome.

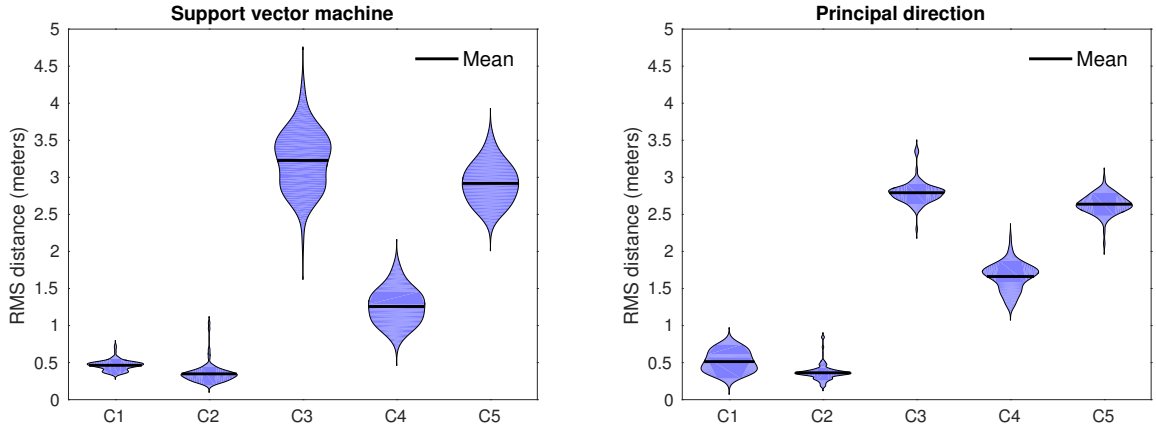


Figure 5.2: *Violin plots of the RMSD for different corridor zones. The left panel corresponds to the support vector machine algorithm while the right panel corresponds to the principal direction algorithm.*

Intersection algorithms

The algorithms *ray-tracing* and *optimal arc* have been evaluated at intersections I2, I3 and I4. A representative left turn was tested at each of these intersections.

Ray-tracing: An observation while running the algorithm is that steep turns require smaller path lengths between the current and next pose of the vehicle. Therefore, a dynamic path length was used based on the steepness of the turn. Another observation is that the ray-tracing algorithm has been optimized for turns, so it has the tendency not to stay in the centre once it enters a corridor. This increases the deviation from the reference path (high RMSD). This is evident at intersection I2 which is more corridor-like.

Optimal arc: An increased time to execute the radius, i.e. delays helps make steeper turns as observed while executing the algorithm. For example, a 3 second delay is better than a 2 second delay in the code to execute a turn at the maximum steering angle when the obstacles are close. Therefore this needs to be controlled dynamically. In a less steep turn the arc angle has to be close to zero as is evident from the evaluations.

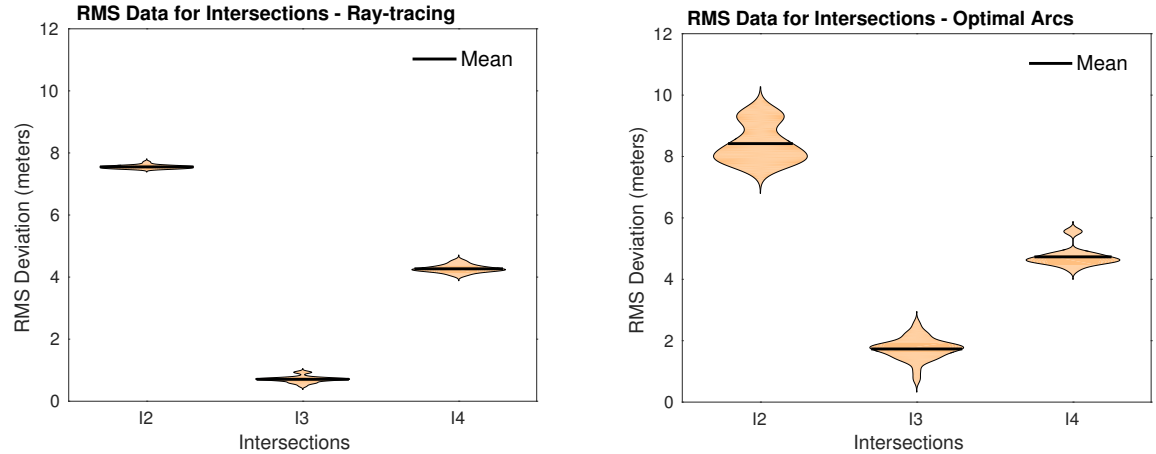


Figure 5.3: *Violin plots of the RMSD for different intersection zones. The left panel corresponds to the ray-tracing algorithm while the right panel corresponds to the optimal arc algorithm.*

Comparison As seen in Figure 5.3, the results from ray-tracing algorithm have a lower deviation than the optimal arc algorithm with respect to a human driven vehicle simulation. If all intersections are considered the

overall mean RMSD was 4.17 metres for the ray-tracing and 4.93 metres for optimal-arc method. Moreover, the variance between the different generated trajectories within each intersection is higher in the optimal arc method indicating unpredictability or a fine-grained reaction to obstacles or perturbations. In general, the RANSAC algorithm used for clustering is based on random samples from the obstacle data and this can lead to stochastic results.

General discussion

As discussed above, the RMSD is based on a trajectory recorded by a human driving a vehicle using a joystick. There are arguments in favour of making automated driving approximate the way humans drive. Thus, in that sense this evaluation gives an idea of how similar to human driving the paths generated by the algorithms are. However, a more objective measure is required to evaluate the performance. It is possible in some cases that the trajectories generated by the navigation algorithms are more safe than that of a human and a low variation in the results does not necessarily mean a safer path.

5.2 Safety

This section presents the performance of the algorithms in terms of safety distance i.e. the minimum distance from the vehicle to the obstacles in its vicinity as discussed in Chapter 4, Section 4.2.2.

Corridor algorithms

Each algorithm has been executed for a specified number of runs. At each time step, the minimum distance from the vehicle to the surrounding obstacles is calculated and the evaluation of each algorithm is based on the ensemble average which is presented below

$$\overline{d^t_{\text{minN}}} = \frac{1}{N}\sum_{i=1}^{N} d^t_i \tag{5.1}$$

where N is the total number of runs, d^i_t is the minimum distance that has been calculated in the i^{th} run, at time step t. In order to obtain the results, the vehicle has been initially placed close to one of the walls which is approximately two metres distance. The results are illustrated in Figure 5.4. The *support vector machine* algorithm has a tendency to move the vehicle away from the walls. This is due to the fact that the algorithm generates a path which maximises the perpendicular distance between the two walls. On the other hand, the *principal direction* algorithm maintains the initial distance. Therefore in terms of safety, the *support vector machine* algorithm performs better.

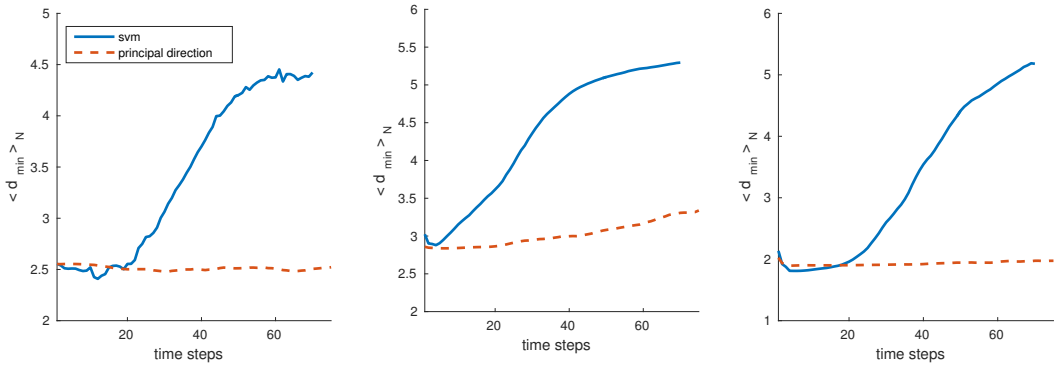


Figure 5.4: *The ensemble average of the minimum distance of the vehicle from the walls as a function of time steps. The solid line corresponds to support vector machine, while the dashed line corresponds to the principal direction. Left panel: corridor C1, Middle panel: corridor C2, Right panel: corridor C4.*

Intersection algorithms

The results for the intersection algorithms are illustrated in Figure 5.5, which compares the distributions of the two algorithms in two distinct types of intersections. It can be observed that in general the *ray-tracing* algorithm maintains a higher safe distance (ranges 7-9 m in I2 and 5-6 m in I3). The intersection I2 is more corridor-like and as a result the *optimal arc* algorithm which uses arcs, tends to go closer to the walls and hence the larger frequency of values in the range 1 m. The *ray-tracing* is able to center the path after making the turn and maintain a reasonable safety distance throughout.

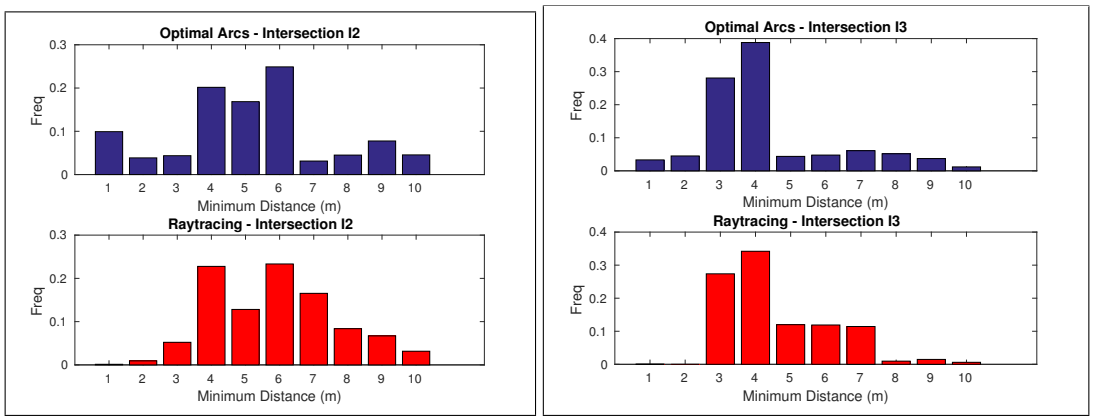


Figure 5.5: *Comparison between the minimum safety distance distributions for the two intersection algorithms.*

5.3 Temporal performance

This section evaluates the navigation algorithms in terms of execution time. The algorithms consist of two main parts: *clustering* and *path generation*. Thus, each part has been timed separately. The algorithms have been executed for 10,000 runs and the results, presented in milliseconds, can be seen in Table 5.1.

In general, it can be observed that the execution time for the clustering part is greater than that of the path generation part. This result is expected as the clustering part deals with data processing which is a time consuming procedure. More specifically, the *principal direction*, *ray-tracing* and *optimal arc* have similar execution time, ranging from 4 to 6 milliseconds. This can be explained as the latter three have been developed under the same PCL framework (RANSAC, Chapter 3, Table 3.4). However, the *principal direction* has the best performance overall as it navigates in a corridor and thus it has to process only two wall segments. In comparison, the other two navigate at an intersection and thus have to process a more complex environment.

A significant difference can be seen in the *support vector machine*. The complex nature of spectral clustering slows down the computations and thus this limits the frequency with which the algorithm can generate and update a path. For example, a frequency rate of 10 Hz would not be possible for the SVM algorithm. However, SVM clustering is more robust compared to RANSAC and hence a trade off between accuracy and efficiency is evident. Regarding the path generation algorithms, all of them have a high algorithmic efficiency.

Algorithms	**Clustering**		**Path generation**	
	Mean	*Std. dev.*	Mean	*Std. dev.*
Support vector machine	352.8	*23.19*	2.212	*0.540*
Principal direction	4.218	*0.757*	0.075	*0.028*
Ray-tracing	5.979	*2.216*	0.736	*0.278*
Optimal arc	6.582	*2.608*	2.123	*0.872*

Table 5.1

6 Conclusions and future work

Autonomous robot or vehicle navigation is a well developed area of research and has seen some intense activity in the past few years. However, the problem of local, relative path planning can be considered as a less explored area of research. The scope of this thesis was to investigate solutions that could solve this navigation problem in an efficient manner, both regarding computational complexity and memory usage. Therefore, the first task was to carry out an extensive literature review in order to get a deeper understanding of the problem.

Throughout this thesis project, several forms of navigation algorithms were explored and a local path planning framework was developed. This framework is currently able to handle certain scenarios in a mining area such as: *navigation in a corridor* and *navigation at an intersection*. As of now, four such stand-alone algorithms have been developed which take as input a two dimensional point cloud of the close surroundings, and predict a local, relative path for the vehicle. This framework could be easily extended to include more such scenarios, such as navigating within loading and unloading zones. These zones have more stringent accuracy requirements for vehicle navigation.

The corridor and intersection scenario handling can be improved as some of the developed algorithms are simple and highly efficient whereas others are inefficient. More specifically, the principal direction algorithm is a good starting point to navigate within a corridor. However, it is not always capable of centering the path along the corridor. On the other hand, the support vector machine is more robust in that sense but it requires more computational time due to the inherent complexity of the spectral clustering. Alternative clustering methods could make the algorithm more efficient. This highlights the trade-off between robustness and efficiency of the algorithms.

Another possible area of improvement is that some of these algorithms need tuning and dynamic parameter adjustments which can be difficult. For example the ray-tracing algorithm for navigation at intersections needs a dynamic look-ahead path length based on the steepness of the turn. The optimal arc length needs a dynamic delay, in order to provide the trajectory follower module, enough time to execute the turn completely. These dynamic parameters can be identified and auto-tuned by means of feed-forward neural networks or other machine learning algorithms. While some machine learning algorithms *were* used here such as SVMs and spectral clustering, one could consider generating the entire navigation framework using only such algorithms. This will allow the algorithms to learn from and adapt to a dynamic environment.

The current implementation uses surrounding information from 2D LIDAR data. In addition, 3D point cloud data can be used to handle navigating in those areas where navigation needs to be highly precise and the obstacles are numerous or complicated. In these cases, processing can be done using convolutional neural networks on 3 dimensional point cloud data from cameras mounted on the vehicle.

Lastly, the framework implementation is currently running on a simulator and makes use of simulated sensor data from a simulated vehicle. The next step would be to implement it on an independent hardware platform such as a Raspberry Pi. The developed algorithms will enable implementing the framework on a real truck. The challenge ahead lies in the additional work required in interfacing with the sensor and actuator hardware on a truck.

References

[1] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *Systems, Man and Cybernetics, IEEE Transactions on* **22**.2 (1992), 224–241.

[2] J. E. Bell and P. R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics* **18**.1 (2004), 41–48.

[3] N. Bin et al. Recurrent neural network for robot path planning (2004), 188–191.

[4] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *Robotics and Automation, IEEE Transactions on* **12**.6 (1996), 869–880.

[5] J. Borenstein, L. Feng, and H. Everett. Navigating mobile robots: systems and techniques (1996).

[6] T. Bowden and B. Bauer. *Linux kernel.org /proc filesystem*. Oct. 2009. URL: `https://www.kernel.org/doc/Documentation/filesystems/proc.txt`.

[7] R. A. Brooks. Solving the find-path problem by good representation of free space. *Systems, Man and Cybernetics, IEEE Transactions on* 2 (1983), 190–197.

[8] N Buniyamin et al. A simple local path planning algorithm for autonomous mobile robots. *International journal of systems applications, Engineering & development* **5**.2 (2011), 151–159.

[9] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* **2**.2 (1998), 121–167.

[10] H. M. Choset. Principles of robot motion: theory, algorithms, and implementation (2005).

[11] W. Commons. *Illustration of Ackermann steering geometry*. File: Ackermann turning.svg. 2010. URL: `https://commons.wikimedia.org/wiki/File:Ackermann_turning.svg`.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik* **1**.1 (1959), 269–271.

[13] *DLIB C++ Library*. Aug. 2003. URL: `http://dlib.net/`.

[14] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE* **13**.2 (2006), 99–110.

[15] H. Durrant-Whyte, D. Rye, and E. Nebot. Localization of autonomous guided vehicles (1996), 613–625.

[16] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning (2005), 9–18.

[17] L. Freeston. Applications of the kalman filter algorithm to robot localisation and world modelling. *Electrical engineering final year project. University of Newcastle, Australia* (2002).

[18] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* **4**.2 (1968), 100–107.

[19] J. L. Hintze and R. D. Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician* **52**.2 (1998), 181–184.

[20] N. Hogan. Impedance control: An approach to manipulation: Part II—Implementation. *Journal of dynamic systems, measurement, and control* **107**.1 (1985), 8–16.

[21] J. H. Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. (1975).

[22] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **79**.8 (1982), 2554–2558.

[23] A. Hornung et al. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots* (2013).

[24] Y. K. Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Computing Surveys (CSUR)* **24**.3 (1992), 219–291.

[25] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research* (1996), 237–285.

[26] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering* **82**.1 (1960), 35–45.

[27] M. Kasper et al. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems* **34**.2 (2001), 153–164.

[28] L. E. Kavraki et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* **12**.4 (1996), 566–580.

[29] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research* **5**.1 (1986), 90–98.

[30] N. Koenig and A. Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator (2004), 2149–2154.
[31] S. Kohlbrecher et al. A flexible and scalable slam system with full 3d motion estimation (2011), 155–160.
[32] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation (1991), 1398–1404.
[33] J.-C. Latombe. Robot motion planning. **124** (2012).
[34] S. M. LaValle. Planning algorithms (2006).
[35] H Le-Huy, C Gosselin, et al. Robot path planning using neural networks and fuzzy logic. **2** (1994), 800–805.
[36] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature* **521**.7553 (2015), 436–444.
[37] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. Robotics and Automation. *IEEE Transactions on, 7(3):376 –382. ISSN 1042-296X* (1991).
[38] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on* **7**.3 (1991), 376–382.
[39] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot (1991), 1442–1447.
[40] E. Masehian and D. Sedighizadeh. Classic and heuristic approaches in robot motion planning-a chronological review. *World Academy of Science, Engineering and Technology* **23** (2007), 101–106.
[41] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**.4 (1943), 115–133.
[42] B. McNaughton, L. Chen, and E. Markus. "Dead reckoning," landmark learning, and the sense of direction: a neurophysiological and computational hypothesis. *Cognitive Neuroscience, Journal of* **3**.2 (1991), 190–202.
[43] M. Meng and A. C. Kak. NEURO-NAV: a neural network based architecture for vision-guided mobile robot navigation using non-metrical models of the environment (1993), 750–757.
[44] J. Miura. Support vector path planning (2006), 2894–2899.
[45] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* **2** (2002), 849–856.
[46] J. Ng and T. Bräunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems* **50**.1 (2007), 73–84.
[47] V. Nguyen et al. A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics (2005), 1929–1934.
[48] S. Nissen. Implementation of a Fast Artificial Neural Network Library (fann) (2003).
[49] M. H. Overmars and P. Svestka. A probabilistic learning approach to motion planning (1994).
[50] J. Pearl. Heuristics: intelligent search strategies for computer problem solving (1984).
[51] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network (1989).
[52] H. H. Poole. Fundamentals of robotics engineering (2012).
[53] C. Qingyang et al. Local path planning for an unmanned ground vehicle based on SVM. *International Journal of Advanced Robotic Systems* **9** (2012).
[54] M. Quigley et al. ROS: an open-source Robot Operating System (2009).
[55] M. Quigley et al. ROS: an open-source Robot Operating System. **3**.3.2 (2009), 5.
[56] J. Racz and A. Dubrawski. Artificial neural network for mobile robot topological localization. *Robotics and autonomous systems* **16**.1 (1995), 73–80.
[57] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on* **8**.5 (1992), 501–518.
[58] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL) (2011).
[59] K. H. Sedighi et al. Autonomous local path planning for a mobile robot using a genetic algorithm. **2** (2004), 1338–1345.
[60] J. M. Snider. Automatic steering methods for autonomous automobile path tracking (2009).
[61] K. Sugihara and J. Smith. Genetic algorithms for adaptive motion planning of an autonomous mobile robot (1997), 138–143.
[62] S. Thrun et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of field Robotics* **23**.9 (2006), 661–692.
[63] D. D. Tsankova. Neural networks based navigation and control of a mobile robot in a partially known environment (2010).

[64] R. J. Urbanowicz and J. H. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications* **2009** (2009), 1.
[65] V Vapnik. Estimation of Dependencies Based on Data (1979).
[66] V. N. Vapnik and V. Vapnik. Statistical learning theory. **1** (1998).
[67] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing* **17**.4 (2007), 395–416.
[68] Y. Wang and G. S. Chirikjian. A new potential field method for robot path planning. **2** (2000), 977–982.
[69] K. M. Wurm et al. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. **2** (2010).