Obstacle Detection and Avoidance for an Automated Guided Vehicle

Detektion av hinder och hur de kan undvikas för ett autonomt guidat fordon

Authors:
Filip Berlin
Sebastian Granath

Supervisor:
Erik Frisk, ISY Linköping University

Examiner:
Björn Olofsson, ISY Linköping University

External Supervisors:
Michal Godymirski, Toyota Material Handling
Filip Sundqvist, Toyota Material Handling
Mattias Arnsby, Toyota Material Handling
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnad som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Filip Berlin and Sebastian Granath
Abstract

The need for faster and more reliable logistics solutions is rapidly increasing. This is due to higher demands on the logistical services to improve quality, quantity, speed and reduce the error tolerance. An arising solution to these increased demands is automated solutions in warehouses, i.e., automated material handling. In order to provide a satisfactory solution, the vehicles need to be smart and able to solve unexpected situations without human interaction.

The purpose of this thesis was to investigate if obstacle detection and avoidance in a semi-unknown environment could be achieved based on the data from a 2D LIDAR-scanner. The work was done in cooperation with the development of a new load-handling vehicle at Toyota Material Handling. The vehicle is navigating from a map that is created when the vehicle is introduced to the environment it will be operational within. Therefore, it cannot successfully navigate around new unrepresented obstacles in the map, something that often occurs in a material handling warehouse.

The work in this thesis resulted in the implementation of a modified occupancy grid map algorithm, that can create maps of previously unknown environments if the position and orientation of the AGV are known. The generated occupancy grid map could then be utilized in a lattice planner together with the A* planning algorithm to find the shortest path. The performance was tested in different scenarios at a testing facility at Toyota Material Handling.

The results showed that the occupancy grid provided an accurate description of the environment and that the lattice planning provided the shortest path, given constraints on movement and allowed closeness to obstacles. However, some performance enhancement can still be introduced to the system which is further discussed at the end of the report.

The main conclusions of the project are that the proposed solution met the requirements placed upon the application, but could benefit from a more efficient usage of the mapping algorithm combined with more extensive path planning.
Acknowledgements

This thesis project would not have been possible to complete without the help of a large amount of helpful and driven people. Firstly, we would like to thank Eva Hädding and Mattias Arnsby at Toyota Material Handling for entrusting us the opportunity to work with this thesis project. Without their belief in our engineering abilities, the project would not have started.

Secondly, we would like to thank our supervisors at Toyota and at Linköping University. We had the luxury of having two in-house Toyota-engineers at our disposal, whenever a question appeared regarding python memory operations, the vehicle software structure or the LIDAR-scanners. Thank you, Michal Godymirski for your help during our reoccurring weekly meetings, and thank you Filip Sundqvist for the company and guidance at the testing area. Best of luck with your future challenges within the autonomous logistic industry.

Further on, our supervisor and our examiner at Linköpings University provided us with thoroughly thought-out questions from which multiple interesting and challenging discussions arose. This helped us bring more aspects to our analysis, and overall helped us improve the report. Thank you, Erik Frisk, for your time, thoughts and support. Thank you, Björn Olofsson, for your keen interest and support.

We would like to direct a thank you to Isabelle Darmanin, the project manager for the vehicle that we worked with. Thank you for the warm welcome and positive encouragement, merci beaucoup!

Furthermore, I (Filip) would like to mediate a tremendous thank you to my partner, Tove, and my family, for the undoubted support. It helped me see the bigger picture, persevere and find joy in stressful times. Lastly, I would like to thank Sebastian for the gratifying cooperation during this thesis project.

Finally, I (Sebastian) would like to direct a personal thank you to my family and my partner. Your encouragements, interest and support really boosted the joy that comes from solving complicated problems. The warmest of thank you, Kiina, Björn G., Klara and Sofia. And lastly, thank you Filip for the good cooperation during this project.
Contents

1 Introduction

1.1 Background and Purpose ........................................ 1
1.2 Prerequisites .................................................... 2
  1.2.1 System Overview ........................................ 2
  1.2.2 Description of the AGV .................................. 4
  1.2.3 SICK LIDAR MicroScan 3 ................................. 4
  1.2.4 AGV Safety Fields ......................................... 6
1.3 Problem Statement ............................................. 7
1.4 Related Work ................................................... 7
1.5 Delimitations .................................................... 9

2 Theory

2.1 Obstacle Detection .............................................. 10
  2.1.1 Occupancy Grid Mapping ................................. 10
  2.1.2 The Inverse Sensor Model ............................... 15
  2.1.3 Bresenham’s Line Algorithm ........................... 17
  2.1.4 A Simple Inverse Measurement Model ................. 21
2.2 Object Avoidance ............................................. 22
  2.2.1 Discrete Feasible Planning .............................. 23
## Contents

<table>
<thead>
<tr>
<th>2.2.2 Graph Search</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3 Graph Search Algorithms</td>
<td>24</td>
</tr>
<tr>
<td>2.2.4 Lattice Planning</td>
<td>26</td>
</tr>
<tr>
<td>2.2.5 Dubins Curves</td>
<td>28</td>
</tr>
</tbody>
</table>

### 3 Method

<table>
<thead>
<tr>
<th>3.1 Object Detection</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1 LIDAR Data Extraction and Conversion</td>
<td>31</td>
</tr>
<tr>
<td>3.1.2 Occupancy Grid</td>
<td>32</td>
</tr>
<tr>
<td>3.1.3 Collision Detection</td>
<td>34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.2 Object Avoidance</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1 Calculating Motion Primitives</td>
<td>37</td>
</tr>
<tr>
<td>3.2.2 Implementing the A* Algorithm</td>
<td>39</td>
</tr>
<tr>
<td>3.2.3 Choosing the Heuristic</td>
<td>40</td>
</tr>
</tbody>
</table>

### 4 Results

<table>
<thead>
<tr>
<th>4.1 Object Detection</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Selection of Inverse Sensor Model</td>
<td>45</td>
</tr>
<tr>
<td>4.1.2 Dynamic Occupancy Grid Mapping</td>
<td>46</td>
</tr>
<tr>
<td>4.1.3 Configurations of Parameters for the Occupancy Grid Mapping</td>
<td>48</td>
</tr>
</tbody>
</table>

| 4.2 Object Avoidance | 51 |

### 5 Discussion

| 60 |
5.1 Object Detection ................................................. 60
  5.1.1 Rotational Drift in the Occupancy Grid Map ............. 60
  5.1.2 Parameter Design in the Occupancy Grid Map ............ 62
  5.1.3 Computational Efficiency of the Occupancy Grid Map .... 64
5.2 Object Avoidance .............................................. 65
  5.2.1 Lattice Planning ........................................... 65
  5.2.2 The A* Algorithm ........................................... 66
5.3 Future Improvements ........................................... 67
  5.3.1 Object Detection ........................................... 67
  5.3.2 Object Avoidance ........................................... 68
6 Conclusions ..................................................... 70
  6.1 Object Detection ............................................. 70
  6.2 Object Avoidance ............................................. 71
  6.3 Concluding the Problem Statements .......................... 71
Nomenclature

Abbreviation
AGV Automated Guided Vehicle
LIDAR Light Detection And Ranging

Notation
\( \alpha \) Obstacle Depth
\( \beta \) Beam Width
\( \theta \) Heading
\( \kappa \) Curvature
\( \xi \) Beam firing angle
\( \chi \) State \((x, y, \theta)\)
\( \chi' \) Resulting State
\( C(x) \) Cost to Come
\( D \) Dubins Set
\( f(x, u) \) The State Transition Function
\( h(x) \) Heuristic
\( m \) The True Map
\( m^* \) Combined Belief of all Map Estimations
\( m_i \) Map Segment Estimation
\( P \) Set of Motion Primitives
\( p \) Probability
\( Q \) Priority Queue
\( r \) Radius
\( u \) Action
\( U \) Action Space
\( x \) Cartesian Coordinate
\( X \) State Space
\( y \) Cartesian Coordinate
\( z \) Sensor Observation or Cartesian Coordinate

Subscript
\( G \) Goal
\( I \) Initial
1 Introduction

This thesis is dedicated to investigating solutions for obstacle detection and avoidance for an Automated Guided Vehicle, AGV. This chapter will give an introduction to the work and explain the motives behind it.

1.1 Background and Purpose

The need for faster and more reliable logistic solutions is rapidly increasing. This is due to higher demands on the logistical services to improve quality, quantity, speed and reduce the error tolerance. An arising solution to the increased demands is automated warehouse solutions, i.e., automated material handling. There are several advantages that come with automated material handling, e.g., eliminating human error, lower costs, better collaboration, efficient routes, etc. In order to provide a satisfactory solution, the vehicles need to be smart and able to solve unexpected situations without human interaction. At the beginning of the project, this function was not available for the vehicle of interest.

The purpose of this Master Thesis was to investigate how a differential driven AGV can detect obstacles using available sensor data when following a predetermined path, and then construct and follow an avoiding path. This would increase the productivity of the AGV by preventing unnecessary, protective stops, and enable operation in a dynamic environment. Another benefit would be that the need for human intervention would be highly reduced, and thereby increased productivity. Different example scenarios that the AGV should be able to solve are shown in Figure 1.1 and Figure 1.2. The client at Toyota Material Handling, who faced this problem, had a great interest in finding a solution and thereby increase the level of automation in their product.
1.2 Prerequisites

Since this thesis project aimed to solve a practical problem with real hardware for a company, some prerequisites and previous knowledge about the existing product is required to understand the problem formulation.

1.2.1 System Overview

The Obstacle Detection and Avoidance application was to be developed in an already existing software system as illustrated in Figure 1.3 that mainly consists of four parts. These are the Mission Planner, the Communication System, the Control & Positioning Software and the to-be-developed Object Detection and Avoidance application. The Mission Planner is an overlaying system that knows the entire route planned for the AGV and continuously sends short path segments to the Control Software, through the Communication System. The Control Software handles the input control of the
electrical propulsion engines, and ensures that the AGV moves along the desired path segments. The Communication System is the link between all other components. The Mission Planner along with the Communication System are in-house developed software at Toyota Material Handling whereas the Control & Positioning is a third-party software. The working principle of the Positioning Software is that it creates a reference map of the operational environment when it is deployed. This allows the software to maintain references to landmarks and by combining this with odometry, it can generate a state estimation. This is one of the reasons why the AGV cannot avoid obstacles that are not presented in the reference map and the Object Detection and Avoidance was the main objective of this thesis. It was expected to act as a surveillance system that could override the Mission Planner’s path when necessary.

Figure 1.3: A basic overview of the system that the Object Detection and Avoidance-application was developed for. The Mission Planner provides the AGV with path segments from the main path. It communicates with the AGV via the Communication System. The Control & Positioning Software is a third-party developed software that allows the AGV to follow the path segments, while it simultaneously provides a position. Object Detection & Avoidance is the to-be-developed application that will act as a surveillance system of obstacles not represented in the map.
1.2.2 Description of the AGV

This thesis worked with an AGV that can carry a standard EUR-pallet with the weight of approximately 1.2 tonnes, handle luggage at airports or be configured for other different applications. The AGV has a LIDAR-sensor mounted at the front, scanning in the forward heading direction and has no sensing capability in the backwards direction. As a consequence of this, it was desired that the AGV would avoid reversing into the blind spot direction or taking too sharp turns. An image of the AGV can be found in Figure 1.4.

![AGV Image](image.png)

Figure 1.4: An image of the AGV for which the software was developed. In the configuration shown in the image, the AGV is configured to transport bags at an airport. The top module can be changed and thereby it can be used for multiple purposes.

1.2.3 SICK LIDAR MicroScan 3

The LIDAR sensor used by the AGV is the SICK MICS3-CBAZ40ZA1P01 (SICK [2021]), a 2D LIDAR shown in Figure 1.5a. Relevant specifications for that sensor can be found in Table 1.1. It is mounted at the front of the AGV and scans the surroundings with maximum angular range from -47.5° to 227.5°. The angles are relative the x-axis (counter-clockwise as positive) as shown in Figure 1.5b.
Chapter 1. Introduction 1.2. Prerequisites

(a) Sick MicroScan3, the LIDAR sensor mounted on the AGV (MICS3-CBAZ40ZA1P01. Safety systems and solutions SICK 2021).

Figure 1.5: The LIDAR mounted on the AGV and a schematic of its field of view.

(b) A schematic of the LIDARs field of view seen from above.

Table 1.1: Data specifications for the SICK MICS3-CBAZ40ZA1P01 LIDAR-sensor used by the AGV. The data have been collected from the retailers website (SICK 2021).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protective field range</td>
<td>4</td>
<td>m</td>
</tr>
<tr>
<td>Warning field range</td>
<td>40</td>
<td>m</td>
</tr>
<tr>
<td>Number of simultaneously monitored fields</td>
<td>≤ 8</td>
<td>-</td>
</tr>
<tr>
<td>Number of fields</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Number of monitoring cases</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Scanning angle</td>
<td>275</td>
<td>°</td>
</tr>
<tr>
<td>Resolution</td>
<td>30–200</td>
<td>mm</td>
</tr>
<tr>
<td>Angular resolution</td>
<td>0.39–0.51</td>
<td>°</td>
</tr>
<tr>
<td>Response time</td>
<td>≥ 95</td>
<td>ms</td>
</tr>
<tr>
<td>Protective field supplement</td>
<td>65</td>
<td>mm</td>
</tr>
</tbody>
</table>
1.2.4 AGV Safety Fields

For safety reasons, the AGV has a so called safety field positioned in the scanners field of view, illustrated in Figure 1.6. This is an area near the vehicle wherein it can detect objects with the help of the LIDAR scanner. Two types of layers exist within the safety field. When detecting an object within the outermost field layer, the AGV slows down and only allows movement with a low velocity. If an obstacle is detected within the innermost layer, it triggers the protective stop and the AGV brakes. The size of the safety fields depends on the AGV’s velocity and steering angle.

Figure 1.6: A schematic picture of how a safety field for the AGV can be designed. There are two safety levels of the field. If an object is detected in the outer field, the AGV is warned and slows down. If the inner field (closest to the AGV) is triggered, a stop is initialized. The safety fields are reconfigured size- and direction-wise depending on the velocity and bearing, but the illustration aims to provide the reader with a conceptual understanding.
1.3 Problem Statement

The AGV was not able to detect and avoid obstacles in its pre-defined path at the start of the project. During encounters with obstacles it would come to a halt, and require human assistance to become operational again which is time consuming. This is obviously not desirable, and motivated the start of this thesis project. Furthermore, the vehicle had a carrying capability of 1.2 tonnes. With that amount of kinetic energy, it is important that collisions are prevented and that unnecessary stops are avoided. The questions that the project aims to answer are the following:

- How can the available data be used to take safe decisions about the future movements for the AGV, in the environment it is operational within?
- How can the path planning be done in such a way that the deviation from the original path is as small as possible, without activating the safety fields, given that the depth and shape of the obstacle are unknown?

1.4 Related Work

The individual problems that this thesis aimed to solve, have been studied by many others. There are many presented methods that can create maps from sensor data, and many others that focus on feature extraction. The common goal is usually to create a decision basis from which a robot or vehicle can navigate. Methods for solving the problem connected to accurate robotic movements that can filter out unwanted or unfeasible movements are also available from multiple sources. A short description of relevant literature for the two main problems is discussed in the following text.

A key reference of probabilistic robotics is *Probabilistic Robotics* (Thrun, Burgard, and Fox 2005) with focus on topics such as uncertainties and unpredictable changes in environments, sensor limitations, inaccurate robot actuation, modeling errors and computational limitations for vehicles/robots. They present methods for approaching and solving the problems connected to these topics and this book was a fundamental informational source for
this work. They also discuss and explain the mapping method called occupancy grid mapping which is a relevant approach for solving the future navigational problem.

In a different paper called *A comparison of line extraction algorithms using 2D range data for indoor mobile robotics* (Nguyen et al. 2007), an experimental evaluation of different line extraction algorithms from 2D LASER-scans is conducted. It is a relevant topic given that multiple line segments could create a map that may be used for future navigation. Further on, in the article by (Wang et al. 2020), six authors approach the topic of combining the dynamic occupancy grid mapping algorithm with multi-object tracking using a camera in order to reduce the needed computational power even further. This is relevant and indicates that new methods are created based on previously presented approaches, that can generate better systems.

In the paper *An Obstacle Classification Method Using Multi-Feature Comparison Based on 2D LIDAR Database* (Lee, Hur, and Park 2015), the three authors propose an obstacle classification method using LIDAR-data. They discuss the computational speed of the previous LIDAR-based methods, but how these methods often fail to extract good enough features and how it limits the object classification. Their method was experimentally evaluated and aimed to improve obstacle classification.

Regarding handling of unknown environments, Topiwala, Inani, and Kathpal (2018) cover the topic of *frontier based exploration*, which commonly is combined with SLAM in order to map an unknown environment. It aims to position the exploring vehicle as closely to the unknown area as possible without entering it, to maximize the exploration of the environment. This approach is the inspiration for the decisions made when circumventing an obstacle in this thesis.

Lattice planning is extensively covered by Bergman (2019), Pivtoraiko, Knepper, and Kelly (2009) and Ljungqvist et al. (2019). Lattice planning is the method used for path planning in this thesis. These authors use different methods to calculate feasible movements. In this thesis, Dubins curves are used to create the feasible path segments in the lattice planner. Dubins curves was originally presented by Dubins (1957) and more recently investigated by Shkel and Lumelsky (2001).

Papers presenting methods for solving the obstacle detection and avoidance problem for similar applications as in this thesis exist, and one example
is the paper by Halldén and Saltvik (2018) called *Obstacle circumvention by automated guided vehicles in industrial environments*. Their approach was to use feature extraction and identify edges/corners to use as reference points. These reference points are then used for path planning.

In summary, the problems discussed have been approached before, but in every implementation case some parameters always differ. The literature discussed gave a lot of different perspectives and approaches to the presented problems but in this thesis project, a method has been designed by integrating and adapting existing methods into a system to solve this specific problem.

### 1.5 Delimitations

The delimitations for this project were the following:

- The world, in which the AGV operates, is considered static while the vehicle is operational. Changes may occur once the AGV is offline.
- The speed of the AGV is constant and low when entering Object Avoidance mode.
- Experiments connected to making changes in the AGV’s existing hardware setup were not investigated.
- The tests with the AGV were only conducted at the assigned test area at Toyota Material Handling.
- Due to the pandemic, the majority of the work was performed remotely. This reduced the time spent with the hardware and in the testing environment.
2 Theory

This chapter aims to present the theory used in this thesis. It is divided into two parts, the theory behind the Object Detection is discussed in Section 2.1 and the theory connected to Object Avoidance is discussed in Section 2.2.

2.1 Obstacle Detection

The detection of obstacles has been studied within the field of robotics for quite some time, and many well documented methods are available that solve the navigational problem by relying on using a given map of the operational environment. The problem with relying on given maps is that they can become outdated and fail to represent the true environment. If the warehouse blueprint is chosen as the a priori map, inaccuracies or lack of information (about, e.g., furniture and reconstructions) can create problems for navigational software.

For a human, the navigational problem (avoiding walking into walls, tables, etc.) is not especially hard once learned. For a robot, this problem can be more challenging. It needs vision (sensor measurements), a method for decoding the recorded data into comprehensible output, and then take decisions about the robot’s future movement. To enable the interaction between a robot and obstacle(s), the first step is to create a situational map in which the unplanned or non-documented obstacle(s) are represented, and then use this map to plan the future movements. An algorithm that offers a link between sensor data and creates a map representation of the current situation is the Occupancy Grid Mapping algorithm, and it will be discussed next.

2.1.1 Occupancy Grid Mapping

The occupancy grid algorithm was first proposed by H. Moravec and A. Elfes in 1985 with the aim to generate maps using scanners placed upon mobile
robots, where the collected measurement data could be noisy or uncertain (Moravec and Elfes 1985). In order to use the algorithm, the state of the mobile robot had to be known from, e.g., odometry, positional landmarks or GPS. Obtaining the vehicle’s state is a problem separated from the mapping problem, but it has also been studied thoroughly. In 1996, the *Journal of Robotic Systems* published an article discussing the different approaches of the positional estimation problem, but that topic will not be further discussed in this thesis and the interested reader is referred to (Borenstein et al. 1997).

A rigorous description of the occupancy grid algorithm, with pseudo-code, can be found in the book *Probabilistic Robotics* (Thrun, Burgard, and Fox 2005). Only the general concepts of the algorithm are presented in the following text and the interested reader is referred to Thrun, Burgard, and Fox (ibid.) for more in-depth details.

The occupancy grid algorithm is capable of generating maps in three dimensions. Given the hardware limitations (2D LIDAR scanner), the theory will be limited to 2D and rooms are represented by planes or horizontal slices. In Figure 2.1, one can see an example of what an occupancy grid map can look like. In that particular figure, the scanner was directed into a dead-ended alleyway, with an obstacle placed in front of the scanner. In the 2D representation of the world, the color of the cells represents how high the probability is that the cell is either occupied (black), free (white) or unexplored (gray).
Figure 2.1: The image illustrates the principle of the occupancy grid map representation. The white cells are explored and considered free, the black cells are explored and considered occupied, and the gray cells are unexplored and unknown. The figure was generated with the readings from a LIDAR-scanner positioned at (43, 40) facing into the dead-end of the alleyway. The units on the x- and y-axis are in [dm], and each pixel is 1 [dm²].

Two prerequisites that have to be fulfilled when using the algorithm are the following. Firstly, the position and orientation of the AGV need to be known in relation to the global map. The state vector $\chi_t$ can be represented as

$$\chi_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

where $x_t$ is the x-coordinate, $y_t$ is the y-coordinate and $\theta_t$ is the heading angle at the time $t$. Secondly, the robot needs a sensor that can obtain distance measurements from the robot to the surroundings. The measurements can be represented as

$$z_t = \begin{bmatrix} r_t \\ b_t \end{bmatrix}$$

where $r_t$ is the range to the detected obstacle and $b_t$ represents the bearing (firing angle) for the scan at the time $t$ relative to the robot ($b_t = 0$ [rad] represents the forward heading direction for the AGV). With both of these prerequisites fulfilled, the occupancy grid algorithm can be applied.
Chapter 2. Theory  

2.1. Obstacle Detection

The fundamental principle of the occupancy grid algorithm is to estimate the most probable map based on smaller individual binary estimations. The combined belief of all the separate map estimations, \( m_i \), contributes to giving an estimate of how the true map \( m \) looks and can be calculated by (Thrun, Burgard, and Fox [2005])

\[
m^* = \arg \max_m p(m|\chi_1, z_1, \ldots, \chi_t, z_t)
\]

where \( m^* \) is the most probable map estimation derived from the sensor observations \((z_1, \ldots, z_t)\) and the states of the AGV \((\chi_1, \ldots, \chi_t)\) in the global coordinate system. In order to simplify the probability estimations, the algorithm relies upon three assumptions (ibid.).

**Assumption 1** The grid cells have two states. \( p(m_i) = 0 \) denotes that the cell was unoccupied (free), and \( p(m_i) = 1 \) that the grid cell is occupied. Partly occupied space will not be considered.

**Assumption 2** The generated map is considered static with no time variant changes.

**Assumption 3** The grid cells are independent from each other.

The first assumption states that each cell only has two states. The second assumption states that nothing changes in the map once it is created, i.e., no obstacles, walls, etc. can move in the operational environment (except for the robot). The third and final assumption states that the cells are independent from each other. Intuitively, this assumption does not correlate with how environments usually appear, e.g., walls and boxes, where the expectation is that many neighboring cells will represent the wall or obstacle. The reason for introducing this assumption is that it significantly reduces the computational complexity of the algorithm and allows the use of Bayes rule.

By exploiting these assumptions, the probability of representing the true map \( m \), given the states \( \chi \) and observations \( z \), is the same as the joint belief of the separate map estimations \((m_1, \ldots, m_N)\) from

\[
p(m|z_{1:t}, \chi_{1:t}) = p(m_1, m_2, \ldots, m_N)
\]
where the individual map segments $m_i$ can be calculated by (Thrun, Burgard, and Fox [2005])

\[
p(m_i|z_{1:t}, \chi_{1:t}) = \\
= \left[ 1 + \frac{1 - p(m_i|z_t, \chi_t)}{p(m_i|z_t, \chi_t)} \cdot \frac{1 - p(m_i|z_{1:t-1}, \chi_{1:t-1})}{p(m_i|z_{1:t-1}, \chi_{1:t-1})} \cdot \frac{p(m_i)}{1 - p(m_i)} \right]^{-1}. \tag{2.1}
\]

The expression can be further simplified by using Bayes rule, the Markov assumption and log-odds ratio. The Markov assumption states that, if the state of the world is known at a given point in time, then what happened in the past will be conditionally independent from what will happen in the future. The log-odds ratio converts the products into summations, which speeds up the calculations. An odds ratio can be written as

\[
\text{odds}(x) = \frac{p(x)}{1 - p(x)} \quad (2.2)
\]

and it is possible to convert probabilities into odds (and reverse) through

\[
p(x) = \frac{1}{1 + \frac{1}{\text{odds}(x)}}.
\]

The log-odds is derived by applying the logarithm to both sides of (2.1). After these simplifications, only three probabilities are needed to be estimated in order to solve (2.1):

- $p(m_i|z_t, \chi_t)$, what is the probability that the current cell is occupied, given the sensor reading and position?
- $p(m_i|z_{1:t-1}, \chi_{1:t-1})$, is the current cell occupied or free, given all the previous sensor observations and position data?
- $p(m_i)$, how probable is the current map estimation?

This converts the three product operations in (2.1) into summations, which is less computationally demanding. This yields

\[
l(m_i|z_{1:t}, \chi_{1:t}) = l(m_i|z_t, \chi_t) + l(m_i|z_{1:t-1}, \chi_{1:t-1}) - l(m_i) \quad (2.3)
\]
where \( l(x) \) is the previous discussed log-odds ratio

\[
l(x) = \log \frac{p(x)}{1 - p(x)}.
\]

Equation (2.3) can be written in a more compact way that also provides a rule for how the new state of a cell can be described at the time \( t \),

\[
l_{t,i} = l(m_i|z_t, \chi_t) + l(m_i|z_{1:t-1}, \chi_{1:t-1}) - l(m_i).
\]

This update rule is used for all the cells in Figure 2.1, which generates the map. The first term in the summation corresponds to something called an inverse sensor model (Thrun, Burgard, and Fox 2005), which will be discussed further in Section 2.1.2. The second is the recursive term and the third is the prior information about the cell. This calculation is iterated over each cell in the grid map, and the algorithmic approach for this method can be found in Algorithm 1 from Thrun, Burgard, and Fox (ibid.). The \( \text{inv} \_\text{sensor} \_\text{model} \) needs to be designed in a way that it captures the behaviour of the used scanner.

\begin{algorithm}
occupancy\_grid\_mapping(\{l_{t-1,i}\}, \chi_t, z_t)
for all cells in \( m_i \) do
    if \( m_i \) in perceptual field of \( z_t \) then
        \( l_{t,i} = l_{t-1,i} + \text{inv} \_\text{sensor} \_\text{model}(m_i, \chi_t, z_t) - l_0 \);
    else
        \( l_{t,i} = l_{t-1,i} \);
    end
return \{l_{t,i}\}
\end{algorithm}

2.1.2 The Inverse Sensor Model

The \textit{inverse sensor model} in Algorithm 1 can be designed in many different ways, and varies with the type of sensor that is being used. The aim of
introducing an inverse sensor model is to convert a scan reading into infor-
mation about which cells that have been explored, are occupied or are free
(Thrun, Burgard, and Fox 2005). More specifically, the inverse sensor model
estimates which environment could explain the current scan? The scanner
records a hit at a distance $r$ with the bearing $b$ as illustrated in Figure 2.2a.
Then the space between the scanner and the detected object is classified as
free, see Figure 2.2b.

![Figure 2.2a](image1.png)

(a) The scanner measures an obstacle at the bearing $b$ and range $r$, seen
from above.

![Figure 2.2b](image2.png)

(b) The map after applying the occupancy grid algorithm, together with
the inverse range sensor model.

Figure 2.2: The figure illustrates the principle of how the inverse sensor
model is expected to behave.

The cell nearest to the hit is most likely occupied, and the cells behind
the hit are hidden from the view of the scanner. Following this reasoning,
the hidden cells can not be updated and one can imagine it as searching
through a dark room with a flashlight. In addition, all the cells occupied by
the AGV are also considered to be free and the resulting map is expected
to be something similar to Figure 2.3.
Figure 2.3: The desired result after detecting an obstacle with the scanner, and applying the algorithms. The width of the beam and depth of the obstacle can be configured with two parameters in the algorithm.

The goal with the inverse range sensor model is to obtain information about the world in a systematic way, based on the sensor scans. One algorithm that transforms a line drawn in a continuous space into a grid representation of that exact line is the Bresenham’s line algorithm.

### 2.1.3 Bresenham’s Line Algorithm

Bresenham’s line algorithm was officially introduced in 1963, by J. E. Bresenham at the Association for Computing Machinery’s national convention in Denver, USA. Then it took two years before it was published in IBM’s system journal, see Bresenham (1965). The algorithm presents a solution to identify the squares that the line passes through when drawn over a grid. Bresenham’s algorithm is fast and require low computational power which is a big advantage. In Figure 2.4, this principle is illustrated.
Chapter 2. Theory  

2.1. Obstacle Detection

![Diagram](image.png)

Figure 2.4: The figure illustrates the general principle of Bresenham’s Line Algorithm. In the continuous world, the line between point A and B is an ordinary straight line, but in the grid-based world it has to be represented by the colored squares. The goal of the algorithm is to return the indices of the colored squares.

In general, the algorithm uses the axis that increases/decreases the fastest and increments/decrements over it, depending on the direction it is heading towards. In Figure 2.5, the quadrants have been divided into eight different regions (octants). If a line starts at origo and heads into the two top octants where $y_1 < y_2$, the $y$-value will always increase faster than the $x$-value. In that region, Bresenham’s algorithm will always increment the $y$-value and it will be considered the fast axis. In addition to the fast axis, the slow axis needs to be determined. If the $y$-axis is the fastest, the $x$-axis will be the slow axis, and depending on which direction the line is heading (positive/negative) the value will be incremented/decremented. See Figure 2.6 for a schematic illustration.
Figure 2.5: The figure illustrates which parameters Bresenham’s Line Algorithm use in order to operate. The coordinate plane is divided into eight pieces (octants), and the first point of a line is assumed to be in origo. Depending on which octant the line is heading into, the algorithm will operate in different ways.

The two octants closest to an axis are coupled with that particular axis. For example, if the $y$-axis is the fast axis (heading upwards) the text in the two octants beside the positive segment of the $y$-axis is related to that scenario.
The algorithm needs the coordinates for the start point \((x_0, y_0)\) and the end point \((x_1, y_1)\) as an input. Depending on which octant the line is heading into, the algorithm will use different ways to calculate which pixel that needs to be colored. The steps that the algorithm follows can be found in the pseudo-code in Algorithm 2.
2.1.4 A Simple Inverse Measurement Model

In the literature where the implementation of the occupancy grid algorithm is described, a different inverse measurement model is presented (Thrun, Burgard, and Fox 2005). The model calculates the range \( r \) and bearing \( b \) to the detected obstacle, and then identifies which cells that are within the cone-shaped representation of the laser beam. The area of the cone-shaped laser representation can be configured by the assumed obstacle depth \( \alpha \), and the opening angle \( \beta \) (the same as the beam resolution/width). The value of \( \beta \) is calculated by

\[
\beta = \frac{\text{Scanner’s Total Field of Vision}}{\text{Number of Beams}}
\]
Furthermore, this inverse measurement model uses the information about the scanner’s maximum range $z_{\text{max}}$, the index to the beam closest to the detected obstacle $k$ and the heading $\theta$ of the AGV. When occupied cells are identified, $l_{\text{occ}}$ is added to their log-odds value, increasing the belief that the cell is occupied. The cells between the obstacle and AGV are added with the value $l_{\text{free}}$. The pseudo code for the algorithm can be found in Algorithm 3. The advantage of creating a cone over each laser beam, is that they will cover a circular area when they are combined, instead of just creating multiple lines with increasing spacing between them.

Algorithm 3: Algorithm inverse range sensor model($m_i; \chi_t; z_t$):

\begin{verbatim}
inverse_range_sensor_model($m_i, \chi_t, z_t$)
Let $x_i, y_i$ represent the center-of-mass of $m_i$, and $\chi_t = [x, y, \theta]$
$r = \sqrt{((x_i - x)^2 + (y_i - y)^2)}$
$\phi = \arctan2((y_i - y), (x_i - x)) - \theta$
$k = \arg\min_j|\phi - \theta_{j,\text{sens}}|$
if $r > \min(z_{\text{max}}, z_{k,t} + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then
| return $l_0$
end
if $z_{k,t} < z_{\text{max}}$ and $|r - z_{\text{max}}| < \alpha/2$ then
| return $l_{\text{occ}}$
end
if $r \leq z_{k,t}$ then
| return $l_{\text{free}}$
end
\end{verbatim}

2.2 Object Avoidance

The Object Avoidance part of the application is a matter of planning a collision-free path around the obstacle, with a given map of the environment. This section contains the theory of path planning in a discrete space with lattice planning.
2.2.1 Discrete Feasible Planning

Discrete feasible planning is a matter of reducing the available and feasible configurations of the AGV in order to reduce the complexity of the problem. The general idea is that for every configuration in the world, there is a state and the set of all combined states is called the state space $X$ (LaValle 2006). For discrete planning, it is of importance that this set is countable and preferably finite. In addition, the set has to be large enough to solve the task, i.e., contain enough states. Coupled to the state space, there exists an action space $U$ for every state $\chi$. The transition from one state to another can be defined as $\chi' = f(\chi, u)$, where $\chi'$ is the state resulting from the action $u$ from the state $\chi$ and $f(\chi, u)$ is the state transition function. Furthermore, an initial state $\chi_I \in X$ and a set of goal states $X_G \subset X$ are required. A summary of the formulation as given by LaValle (ibid.) is:

1. A nonempty state space, $X$, which is finite or countably infinite set of states.
2. For each state $\chi \in X$, a finite action space $U(\chi)$ exists.
3. A state transition function $f$ that produces a state, $f(\chi, u) \in X$, for every $\chi \in X$ and $u \in U(\chi)$. The state transition equation is derived from $f$ as $\chi' = f(\chi, u)$.
4. An initial state $\chi_I \in X$.
5. A goal set $X_G \subset X$.

2.2.2 Graph Search

To introduce graph search, it is essential to have a basic understanding of graphs. A graph consists of nodes and edges, where each node represents a state and each edge can represent a feasible path between two adjacent nodes. The idea is to find a feasible path from a start node or initial state $\chi_I$, to a goal node/state $\chi_G$. Each edge normally has a cost of some sort, e.g., length or time. A common problem is then to find a feasible path with minimal cost. An illustration of a graph is shown in Figure 2.7.
Figure 2.7: An example of a graph. Nodes and edges are marked in the picture, where nodes represent states and edges represent motions between states.

To utilize graph search, a discrete set of nodes needs to be distributed throughout the operational environment. The search itself can be conducted with one of the many algorithms dedicated to solve the graph search problem, for example Dijkstra’s or the A* algorithm that are explained next. The common approach is to begin the search at the start node, and continue to search nodes until the goal is reached. To avoid cycling (if cycles exist), it is important to keep track of previously visited nodes.

### 2.2.3 Graph Search Algorithms

There are several algorithms that use the discrete planning approach, e.g., Dijkstra and A* are examples of graph search algorithms. Graph search algorithms use graphs consisting of nodes and edges to plan a path. Both Dijkstra and A* are covered by LaValle [2006] where A* is an extension of Dijkstra. Dijkstra’s and the A* algorithm are presented below in Algorithm 4 and 5, notice the differences that are underlined.

The idea of Dijkstra’s algorithm is to find the shortest path to all nodes from an initial node, i.e., constructing a shortest path tree. To find the shortest path, the algorithm keeps track of the cost-to-come for each visited node and explores the search tree prioritized by this parameter. Cost-to-come is typically length of the path or time, i.e., the shortest path/least amount of time is prioritized.
Dijkstra’s algorithm initiates with a cost-to-come ($C(\chi_I) = 0$) associated with the current node and the priority queue, $Q = \{\chi_I, C(\chi_I)\}$. Initially, the queue only contains the initial node, $\chi_I$. Then all the actions $u$ in the action set $U$ that are associated with $\chi_I$ are examined. If the resulting node of any action, $\chi'$, has not yet been examined or the cost-to-come of that node is larger than the current cost-to-come ($C(\chi)$) added with the cost-to-go from $\chi$ to $\chi'$ ($d(\chi, \chi')$), then the cost-to-come of $\chi'$ is $C(\chi') = C(\chi) + d(\chi, \chi')$. The successor state $\chi'$ along with its cost-to-come $C(\chi')$ is therefore updated and inserted into the priority queue. The previous node of $\chi'$, i.e., $\chi$, is also tracked in order to avoid cycles. This is then repeated for every node in the state space $X$ until the path with the lowest cost to come to the goal node is found.

The procedure for A* is very similar apart from the underlined difference, and it involves something called the heuristic, $h(\chi)$. This allows the algorithm to prioritize the nodes with the lowest potential cost-to-go according to the implemented heuristic. One approach is to use the Euclidean distance from the current node to the goal node. This obviously has advantages in terms of computational complexity, as the number of nodes searched can be significantly reduced. To ensure optimality with A*, it is required that the heuristic is admissible, i.e., $h(\chi) \leq h^*(\chi)$ where $h^*(\chi)$ is the (unknown) true cost-to-go function. Furthermore, to provide an efficient solution and avoid reusing the same node twice in the search it is of importance to have a consistent heuristic, i.e., the heuristic becomes better along the path, $h(\chi) - h(\chi') \leq d(\chi, \chi')$. So to summarize:

- $Q$ is a priority queue.
- $C(\chi)$ is the cost-to-come, i.e., the cost from the start node to the current node.
- $h(\chi)$ is the heuristic, i.e., the estimated cost-to-go to the goal node.
- $h^*(\chi)$ is the true cost-to-go function.
- $u$ is a feasible action for any particular node.
- $U$ is the action space, containing all feasible actions for any particular node.
- A heuristic is admissible if $h(\chi) \leq h^*(\chi)$. 

25
• A heuristic is consistent if \( h(\chi) - h(\chi') \leq d(\chi, \chi') \).

<table>
<thead>
<tr>
<th>Algorithm 4: Dijkstra’s Algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C(\chi_I) = 0 )</td>
</tr>
<tr>
<td>( Q.insert(\chi_I, C(\chi_I)) )</td>
</tr>
<tr>
<td>while ( Q \neq \emptyset ) do</td>
</tr>
<tr>
<td>( \chi = Q.pop() )</td>
</tr>
<tr>
<td>if ( \chi = \chi_G ) then</td>
</tr>
<tr>
<td>( \text{return SUCCESS} )</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for ( u \in U(\chi) ) do</td>
</tr>
<tr>
<td>( \chi' = f(\chi, u) )</td>
</tr>
<tr>
<td>if no_previous(( \chi' )) or ( C(\chi') &gt; C(\chi) + d(\chi, \chi') ) then</td>
</tr>
<tr>
<td>( \text{previous}(\chi') = \chi )</td>
</tr>
<tr>
<td>( C(\chi') = C(\chi) + d(\chi, \chi') )</td>
</tr>
<tr>
<td>( Q.insert(\chi', C(\chi')) )</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>return FAILURE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 5: The A* Algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C(\chi_I) = 0 )</td>
</tr>
<tr>
<td>( Q.insert(\chi_I, C(\chi_I) + h(\chi_I)) )</td>
</tr>
<tr>
<td>while ( Q \neq \emptyset ) do</td>
</tr>
<tr>
<td>( \chi = Q.pop() )</td>
</tr>
<tr>
<td>if ( \chi = \chi_G ) then</td>
</tr>
<tr>
<td>( \text{return SUCCESS} )</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for ( u \in U(\chi) ) do</td>
</tr>
<tr>
<td>( \chi' = f(\chi, u) )</td>
</tr>
<tr>
<td>if no_previous(( \chi' )) or ( C(\chi') &gt; C(\chi) + d(\chi, \chi') ) then</td>
</tr>
<tr>
<td>( \text{previous}(\chi') = \chi )</td>
</tr>
<tr>
<td>( C(\chi') = C(\chi) + d(\chi, \chi') )</td>
</tr>
<tr>
<td>( Q.insert(\chi', C(\chi') + h(\chi')) )</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>return FAILURE</td>
</tr>
</tbody>
</table>

### 2.2.4 Lattice Planning

For this particular application, there is a desire to maintain a smooth path, when encountering and avoiding obstacles. Therefore, it is of interest to look into **lattice planning**. The fundamental idea with lattice planning is to restrict the allowed vehicle movements to a discrete subset of the valid actions and by doing so, discretize the problem to a graph search problem (Pivtoraiko and Kelly 2005). By constructing a **state lattice representation** as a generalization of a grid, connectivity between nodes can be achieved. If there is a feasible path between two discretized **lattice nodes** (states), then
they are connected with a lattice edge corresponding to that path. A state discretization is also needed, i.e., choosing the properties for the state vector. For a differential-driven vehicle, the state vector can be chosen as a 3D vector consisting of the Cartesian coordinates \((x, y)\) and the heading \(\theta\). Each node in the graph then represents a state in the state space, \([x, y, \theta]\). The construction of a state lattice is dependent on a set of motion primitives \(P\) that contains feasible motions between selected states (Bergman 2019). The computation of these motion primitives can be calculated off-line to save computational time online (online meaning during operation). The construction of the state lattice was primarily done in the three steps presented below and in Figure 2.8.

1. Discretize the state space.
2. Select which pairs of states to connect in the discretized representation.
3. Solve the optimal control problem to compute the motion primitive set \(P\).

An advantage of this method is that the motion primitives only need to be calculated once, from the origin to the surrounding nodes, in step three. From that it is possible to translate the primitives to all other nodes, hence connecting the entire grid in a graph.
2.2.5 Dubins Curves

In the 2D plane, consider a forward-only car (only capable of moving forwards) moving smoothly from an initial configuration to a goal configuration, i.e., from $q_I = [x_I, y_I, \theta_I]$ to $q_G = [x_G, y_G, \theta_G]$. Step 3 in the lattice planning approach is then particularly simple if minimum-length solutions are considered. By adding the constraint that the minimum curvature is $\kappa = \frac{1}{r}$ one faces a common problem in non-holonomic motion planning. This particular problem was addressed by Dubins (1957) who provided a solution for this problem. He showed that any geodesic (i.e., the shortest path between two points on a surface) can be represented by exactly three path segments, either $CCC$ or $CSC$, see Figure 2.9. $C$ (for circle or curve) is an arc with radius $r$ and $S$ (for straight) is a line segment. Each arc, $C$, can turn either left (L) or right (R), which together with the straight, $S$, leaves six different combinations denoted as the Dubins set. The Dubins set is $D = \{LSL, RSR, RSL, LSR, RLR, LRL\}$. Shkel and Lumelsky (2001) have based their paper on Dubins work and summarizes the essence in an intuitive way. The theory presented here is therefore based on the paper by Shkel and Lumelsky.
Dubins approach defines an admissible path as a continuously differentiable curve, which is either an arc of a circle of radius \( r \), followed by a line segment, followed by an arc of a circle of radius \( r \) (CSC) or three subsequent arcs of a circle with radius \( r \) (CCC). A third option applies as well, being a subset of either CSC or CCC. As mentioned, there are three fundamental motions — turning right, turning left and straight line motion. Shkel and Lumelsky (2001) introduce three corresponding operators, \( R_v \) (right turn), \( L_v \) (left turn) and \( S_v \) (straight line), which transform an arbitrary point \([x, y, \theta] \in \mathbb{R}^3 \) into its corresponding image point in \( \mathbb{R}^3 \):

\[
R_v(x, y, \theta) = (x - \sin(\theta + v) + \sin \theta, y + \cos(\theta - v) - \cos \theta, \theta - v)
\]
\[
L_v(x, y, \theta) = (x + \sin(\theta + v) - \sin \theta, y - \cos \theta + v + \cos \theta, \theta + v)
\]
\[
S_v(x, y, \theta) = (x + v \cos \theta, y + v \sin \theta, \theta)
\]  

(2.4)

where index \( v \) indicates that the motion has been along the (C or S) segment of length \( v \). From (2.4) any path in the Dubins set

\[
D = \{LSL, RSR, RSL, LSR, RLR, LRL\}
\]

can be expressed in terms of the corresponding equations. The result of this is a computationally easy way of calculating Dubins curves for a forward-only vehicle to find the shortest path between two configurations, as presented by Shkel and Lumelsky.
This chapter addresses the methods used in this thesis project. The application was developed as a subsystem with access to the AGV’s position, orientation and nominal path. The basic principle of operation is that the Obstacle Detection module continuously updates the occupancy grid map representation and checks whether the nominal path is free or not. If the path contains an obstacle, the Obstacle Avoidance module is activated, which attempts to find an alternate path around the obstacle. When the obstacle is successfully avoided, the Obstacle Avoidance module hands over the command to the external Mission Planner, that would continue to move the AGV on its original path. An overview of the general structure of the application can be found in Figure 3.1 and an in-depth explanation is provided in the following sections.

![Diagram of application structure](image)

Figure 3.1: Overview of the application structure.
3.1 Object Detection

In order to detect obstacles using the data from the AGV-equipped LIDAR, different methods and techniques were used. The procedure presented here describes the steps taken in order to extract and convert data to the occupancy grid map representation.

3.1.1 LIDAR Data Extraction and Conversion

As explained in Section 1.2.3, the LIDAR shoots beams of light in different directions, from $-47.5^\circ$ to $227.5^\circ$ as the maximum range, see Figure 3.2. The light travels through the air until it collides with a reflective surface, and is then reflected to the LIDAR. The scanner registers the time-of-flight and beam number for each beam that is fired. The time-of-flight $\Delta T$ is then used to calculate the distance to the object.

The raw LIDAR data is given in the form $[d, beamNr]$ where $d$ is the distance to the object it hits, and $beamNr$ is the beam number of that particular beam. The beam numbers are counted from $-47.5^\circ$ (the configured starting angle used) and upwards. Given that the angular beam resolution is known, the firing angle for each individual beam, $\xi$, can be calculated as

$$\xi = startAngle + beamNr \cdot resolution$$  \hspace{1cm} (3.1)

Through this, the polar coordinates are obtained for the beams on the format $[d, \xi]$.  

31
Chapter 3. Method

3.1. Object Detection

(a) A schematic view from above for the LIDAR’s field of view.

(b) The LIDAR as it can be configured. The maximum angular range is from $-47.5^\circ$ to $227.5^\circ$, measured positive in the counter-clockwise direction from the $x$-axis. The resolution (res) is determined by the configured start and end angle.

Figure 3.2: A description of the LIDAR’s functionality.

3.1.2 Occupancy Grid

When implementing the occupancy grid algorithm, there are a few design parameters that initially have to be configured, and they can be found in Table 3.1.
Table 3.1: The table contains the initial configuration parameters for the occupancy grid algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z_{\text{max}} )</td>
<td>40</td>
<td>[m]</td>
<td>The parameter defines the maximum length that the scanner could register readings from.</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>0.1</td>
<td>[m]</td>
<td>Assumed depth of an obstacle.</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( \frac{0.51\pi}{180} )</td>
<td>[rad]</td>
<td>The opening angle/angle resolution of the scanner.</td>
</tr>
<tr>
<td>( l_{\text{occ}} )</td>
<td>( \log \frac{0.65}{0.35} )</td>
<td>[-]</td>
<td>The a priori assumption of cells being occupied, used as a design parameter.</td>
</tr>
<tr>
<td>( l_{\text{free}} )</td>
<td>( \log \frac{0.35}{0.65} )</td>
<td>[-]</td>
<td>The a priori assumption of cells being free, used as a design parameter.</td>
</tr>
</tbody>
</table>

The first configuration parameter is the maximum distance the LIDAR could detect an obstacle at, \( z_{\text{max}} \). This information could be found in the data sheet for the scanner, and the maximum distance was set to \( z_{\text{max}} = 40 \) [m]. The second configuration parameter is the depth of an obstacle, \( \alpha \), and it was set to be equally large as the grid discretization size, or in other terms, the grid resolution \( \alpha = \text{gridRes} \). In practice, this parameter controls how many cells that are classified as occupied at the end-point of each laser beam and with this configuration the obstacle depth was set to be one cell. The third parameter is the initial angular width of a beam, \( \beta \). The value of a beam’s angular width changes with the distance it travels, given the phenomenon called light beam spread. That is, the beam can be looked upon as a cone with increasing cross-sectional area. By calculating the relative angle between each cell and the AGV, it is possible to extract all cells that lay within the cone–shape laser representation, and the initial width of the beam was set to \( \beta = \frac{0.51\pi}{180} \) [rad]. The simple inverse sensor model was extended with Bresenham’s line algorithm, which was applied as another post-processing step after the simple inverse sensor model registered a hit. The input coordinates to the algorithm were firstly the AGV’s position and orientation, and secondly the nearest occupied cell for the current beam. The extended algorithm that was used can be found in Algorithm 6. The
change, where Bresenham’s algorithm is used, is underlined.

**Algorithm 6:** Algorithm: extended inverse range sensor model($m_i; \chi_t; z_t$):

\[
\text{inverse\_range\_sensor\_model}(m_i, \chi_t, z_t)
\]

Let $x_i, y_i$ represent the center-of-mass of $m_i$

\[
r = \sqrt{((x_i - x)^2 + (y_i - y)^2)}
\]

\[
\phi = \arctan_2((y_i - y), (x_i - x)) - \theta
\]

\[
k = \arg \min_j |\phi - \theta_{j, \text{sens}}|
\]

if $r > \min(z_{\text{max}}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k, \text{sens}}| > \beta/2$ then

| return $l_0$

end

if $z_t^k < z_{\text{max}}$ and $|r - z_{\text{max}}| < \alpha/2$ then

| bresenhams{AGV position and orientation, first occupied cell} (Algorithm 2)

| return $l_{\text{occ}}$

end

if $r \leq z_t^k$ then

| return $l_{\text{free}}$

end

3.1.3 Collision Detection

In order to detect obstacles, the method utilizes the knowledge of the predefined nominal path. The points on the path are checked, and if any point is located within or at a set distance $d$ to an occupied cell, the path is classified as a collision course. This was done in order to account for the size of the vehicle and the safety fields and Figure 3.3 illustrates the principle. The path is deemed to be collision free if the vehicle has enough room to pass the obstacle with some safety margin (represented by $d$). This is done by checking if each point in the path has any occupied cells within a radius of $d = 1.2$ [m], leaving some sideways safety margin for the vehicle with the dimensions $1.2 \times 0.8$ [m²].
3.2 Object Avoidance

Occupancy grids are as mentioned a way of transforming a continuous world into a discrete representation, and an extension is to use discrete planning methods for the path planning. The idea is to search the grid for free space and construct a feasible path from an initial configuration to a goal configuration in the state space. This is done using graph search with the A* algorithm combined with lattice planning, where the motion primitives are constructed with respect to the allowed turning radius of the vehicle.

A few things have to be taken into consideration in order to solve the path planning problem:

- How many primitives are needed? How many start and goal configurations?
- What is an appropriate heuristic to use for the A* algorithm?
- How can the path planning take into consideration that it cannot see beyond obstacles, i.e., how can the path planning be conducted such that the vehicle is safe from collision at all times?

Regarding the last item, there is no way of knowing what is behind an obstacle, so the approach used to address this issue was to choose a global goal and a local goal. As the AGV follows the nominal path and encounters an obstacle, a point further down the path is chosen as a global goal. The global goal is chosen with the requirement of being 1.8 meters (can be modified as
a parameter) from any occupied cell, even if it is in an unknown environment. The A* algorithm can then plan a path to this global goal with the current knowledge of the environment from the occupancy grid, where unknown cells are considered free. However, the last point in this path that is located in a known free area is identified. To account for any obstacles that may occur in the unknown area, a point on the path is chosen in the free space at a given distance to the limit to the unknown area, i.e., the frontier, as the local goal. This distance can be modified as a parameter. The path up to the local goal is then sent to the AGV to follow, and as it approaches the local goal, a new plan can be calculated with more knowledge of the environment. This is then repeated until the local goal and the global goal coincide. See Figure 3.4 for an illustration of this, and Figure 3.5 for the decision structure. If the global goal is discovered to be occupied when the unknown area has been explored, a new global goal is chosen further ahead on the global path until it is located in a known free area.

(a) A local goal (2) and a global goal (1) are chosen, where a plan is computed to the global goal. The path is, however, only sent until the local goal, after which a new decision and plan can be made.

(b) When reaching the local goal, the AGV has more information about the environment and can make a new plan. In this case, it can create a path directly to the global goal.

Figure 3.4: A point is chosen further down the original path as a global goal (1) and an A*-search is performed, treating unknown area as free. If a path is found, the path is sent as far as the last point in the new path that is known to be free, i.e., the local goal (2). The local goal is chosen with margin to the unknown area to account for any obstacles that may be located in the unknown area. When the local goal is reached another search is performed.
3.2. Object Avoidance

Collision Detected?

Send Path to Global Goal

Choose Global Goal

A*-Search

Yes

Global Goal free?

Close enough to Local Goal?

No

Yes

Choose Local Goal

Local Goal = Global Goal?

No

Yes

Send Path to Global Goal

Go to Global Goal

Close enough to Global Goal?

No

Yes

Return to Main Path

Figure 3.5: The decision structure for when the AGV is in Avoidance Mode. Note that the global goal is considered free if it is located in an unknown environment.

3.2.1 Calculating Motion Primitives

The choice and computation of motion primitives were based upon two considerations. Firstly, the speed when avoiding obstacles is set to a low constant velocity to keep the safety fields at their minimal configuration.
Secondly, they are designed to avoid standstill rotations for the AGV due to risk of collision. The first condition allows for a simpler case compared to when the speed of the AGV varies or is too high. The shortest and fastest path is therefore the same, and the dynamics of the vehicle are negligible. Hence, it was possible to base the motion primitives upon the geometry of the vehicle by utilizing Dubins curves. The second condition reflects the fact that this version of the AGV with the described scanner can not see what is behind due to the 270 degree field of view. Therefore it is required to use a turning radius larger than half the track width of the vehicle to avoid rotating on the spot, see Figure 3.6. A turning radius set to exactly half the track width could cause the vehicle to rotate on the spot due to uncertainties in the control software.

Figure 3.6: The sharpest turn the AGV can take depends on the turning radius. As long as the turning radius is larger than half the track width, $r > \frac{L}{2}$, there is no risk of the AGV rotating on the spot.

Furthermore, the choice of motion primitives affects the computational power needed during operation. Therefore, it is important to choose enough start and goal configurations in order to reach enough states but not too many, as it affects the computational complexity of the $A^*$ algorithm. The track width of the vehicle is 800 [mm], and therefore the minimum turning radius was set to 500 [mm] to account for the uncertainty in the control software. Using (2.4) along with start configuration $[x_I, y_I, \theta_I] = [0, 0, \theta_I]$ and end angles in $[\theta_I, 0, \theta_I]$ relative the start angle $\theta_I$, the primitives seen in Figure 3.7
are obtained.

![Motion Primitives](image)

Figure 3.7: Motion primitives calculated with turning radius 500 [mm] and end angles \([-\frac{\pi}{4}, 0, \frac{\pi}{4}\] [rad] relative each start angle.

### 3.2.2 Implementing the A* Algorithm

There are many different search algorithms that could be used to solve the planning problem here. However, the choice of algorithm is not trivial as the selection made needs to consider completeness, optimality, time complexity and space complexity. Completeness occurs when the algorithm can find a solution if there is one. Optimality occurs when the found solution is the best one in terms of cost, which in this case is the length of the path. Time complexity concerns how long time it takes to find a solution whereas space complexity considers how much memory needs to be utilized to perform the search. The presented theory in Section 2.2.3 provides a basic understanding of how Dijkstra’s algorithm works, and that it provides a solution that is guaranteed to be optimal. However, since it searches every available node in the configuration space it is not necessarily an optimal algorithm in terms of space complexity and, in a large configuration space, not in time complexity. Therefore, in order to save computational effort, i.e., space complexity, the A* algorithm was selected since it with a consistent, well–designed heuristic more efficiently can provide a solution. Algorithms that consider time
complexity such as ARA* (Likhachev, Gordon, and Thrun 2004) were not considered as the confined space for operation of the AGV was considered enough to keep the time complexity at a manageable level.

The A*-algorithm is used for a rough search, i.e., not all the way to the goal node. Due to the limited number of motion primitives it is hard for the algorithm to find a solution that is close enough to provide a good transition to the main path. Therefore, a rough search is used to find a path with a rather relaxed condition for when the goal is found. When this condition is fulfilled, the environment until the goal is assumed to be free. To properly re-connect to the main path, a number of samples from the main path are obtained to which a Dubins curve segment is calculated for each one and the shortest segment is chosen as the transition back to the main path, see Figure 3.8 for illustration.

Figure 3.8: The path generated by the A* algorithm ends when the relaxed condition is fulfilled (the circle). A number of samples were then taken from the main path originating in the global goal to which a Dubins curve is calculated for each one. The shortest one is then used as the transition back to the main path.

3.2.3 Choosing the Heuristic

In order to provide an optimal solution, A* requires an admissible heuristic as explained in Section 2.2.3 $h(\chi) \leq h^*(\chi)$. A simple choice of heuristic is the Euclidean distance, i.e., a straight line from the current node to the goal node. It is trivially admissible and was considered satisfactory for the
application. Including the orientation in the heuristic was not considered due to the negative effects it has on flexibility. For example, if the optimal path is two U-turns then the inclusion of orientation in the heuristic would cause the A* algorithm to not prioritize the nodes facing the wrong direction. Furthermore, to take the safety fields and the size of the vehicle into consideration, an extra cost was added for those nodes that are within 1.2 [m] of occupied cells, found by doing the same comparison as explained in Section 3.1.3. This allows the vehicle to prioritize nodes that are at a safe distance from any obstacles, see Figure 3.9.

Figure 3.9: With the global goal chosen at point (1), the motion primitive (2) would be chosen, if only the Euclidean heuristic would be considered. But with the addition of the added cost of adjacent occupied cells, motion primitive (3) is prioritized.
4 Results

This chapter presents the findings of this thesis using the previously explained theory and methods, as well as the results from the experiments conducted with the AGV at the testing facility at Toyota Material Handling. The chapter is divided into two parts. Section 4.1 contains the results from the Object Detection, and the results from the Object Avoidance is presented in Section 4.2.

4.1 Object Detection

The functionalities expected of the Object Detection module, were that it could generate an occupancy grid map based upon the raw LIDAR data, both when the AGV was stationary and when it was in motion. A few steps were taken to validate the functionality of the static occupancy grid map generation.

**Step 1:** A separate LIDAR scanner was placed in a known environment and a comparison between the blueprint and the registered LIDAR hits was conducted. The purpose of this was to validate that the communication with the scanner worked, and that the angles to each beam were calculated correctly. The initial environment that was chosen was the office landscape at Toyota. Figure 4.1a shows a rough drawing of the office blueprint and Figure 4.1b shows a plot of raw LIDAR measurement data sampled from that placement (one rotational measurement/circular sweep).
Chapter 4. Results

4.1. Object Detection

(a) A rough drawing of the office blueprint, seen from above.

(b) The raw LIDAR measurements from when it was placed in the office in accordance with Figure 4.1a. The scanner was located at the origin.

Figure 4.1: Office description and raw LIDAR data.

When overlapping the office blueprint with the raw LIDAR data, it was possible to see that the scanner was able to capture the structure of the room, see Figure 4.2. However, the scanner was placed at a height of 1.6 m, and therefore it could not detect the nearby desks but managed to detect the walls of the entire office. The recorded dots over the desks are items placed upon the office furniture (hand sanitizer bottles, pictures, etc.). The hits recorded at \( x, y = (-6, 12.5) \) passed through one of the windows and managed to hit another building.
Figure 4.2: The office map and the raw scanner data placed over each other. The scanner was placed at a height of 1.6 [m], and therefore failed to capture the nearby desks. The map was built from five scans.

**Step 2:** The extended occupancy grid algorithm was applied to the raw LIDAR data, and the generated occupancy grid map is presented in Figure 4.3. Any objects limiting the scanner’s vision prevented the beams from reaching the back of the room, and in the shadow of the objects, the environment was classified as unknown. The two steps of firstly extracting the raw LIDAR data and then secondly generating the occupancy grid map, was able to validate the expected performance of the static occupancy grid mapping. It resulted in a map describing the environment from which future movement decisions could be based upon.
Figure 4.3: The office map and the post-processed scanner data placed over each other. The scanner was placed at a height of 1.6 [m], and therefore failed to capture the nearby desks. The axis has the same unit as the grid resolution [dm].

### 4.1.1 Selection of Inverse Sensor Model

The extended inverse sensor model, presented in Section 3.1.2, is a combination of the simple inverse sensor model and the Bresenham’s line algorithm (Algorithm 2). It is better at identifying the free pixels along the beam than the ordinary simple inverse sensor model, and a graphic comparison between the two models can be found in Figure 4.4. In the figure, the extended and the simple algorithm were applied to the same beam. The gray (lighter) squares are identified free cells, when using Bresenham’s line algorithm extension. The burgundy colored pixels (darker) are the cells identified as free,
when the non-extended inverse sensor model was used. In Figure 4.4a, the extended model identified six more cells as free, and in Figure 4.4b it found five more. The amount of cells that could be classified as free was almost doubled, and in the final version of the code, the extended algorithm was implemented.

(a) A comparison between the simple inverse sensor model and the extended inverse sensor model presented in Sections 2.1.4 and 3.1.2. Both algorithms were applied on the same beam. The gray pixels (lighter) are from the extended, and the burgundy (darker) are from the unmodified simpler inverse model.

(b) A comparison between the simple inverse sensor model and the extended inverse sensor model presented in Sections 2.1.4 and 3.1.2. Both algorithms were applied on the same beam. The gray pixels (lighter) are from the extended, and the burgundy (darker) are from the unmodified simpler inverse model. The title contains information about how many pixels the two models identified as free.

Figure 4.4: Comparison of the two different inverse sensor models presented in Section 2.1.4 and 3.1.2 respectively.

4.1.2 Dynamic Occupancy Grid Mapping

In the previous results, the occupancy grids presented were generated when the scanner was statically positioned in a corner of a room. When running the algorithm online on the moving vehicle, the maps in Figure 4.5 were obtained.
(a) An Occupancy Grid Map generated from the belief of the first LIDAR scan.

(b) An Occupancy Grid Map generated from the combined belief of 14 LIDAR scans.

(c) An Occupancy Grid Map generated from the combined belief of 21 LIDAR scans.

Figure 4.5: A description of how the probability-based occupancy grid is constructed based on the number of LIDAR scans.

As seen in the upper right corner of Figure 4.5b, the map has a slight drift in orientation. This is further discussed in Section 5.1.1.

The assigned test area, in which all the experiments were conducted, was quite busy and for almost every experiment something had been moved which resulted in slight variations in the occupancy grid maps generated. However, apart from the walls, the test area had a few stationary mounted objects and they are illustrated in Figure 4.6.
Chapter 4. Results

4.1. Object Detection

Figure 4.6: A schematic drawing over the assigned test area in which the tests were conducted. Only the stationary objects in the room have been drawn into the picture.

4.1.3 Configurations of Parameters for the Occupancy Grid Mapping

By changing the configuration parameters from Table 3.1 in the occupancy grid mapping, different results were obtained. The final values of these parameters were set to $\alpha = 0.3$ [m], $\beta = \frac{0.51\pi}{180}$ [rad] and $z_{max} = 40$ [m]. Firstly, different values of $\alpha$ (assumed obstacle depth) generated different results. In Figure 4.7, three different values of $\alpha$ were used and one can see how the obstacles get thicker when increasing the value of $\alpha$. The number of $\alpha$ roughly corresponds to how many cells that got classified as occupied.
Chapter 4. Results  

4.1. Object Detection

(a) An Occupancy Grid Map generated from two LIDAR scans. α: 1 [dm], β: 0.51 [deg].

(b) An Occupancy Grid Map generated from two LIDAR scans. α: 3 [dm], β: 0.51 [deg].

(c) An Occupancy Grid Map generated from two LIDAR scans. α: 5 [dm], β: 0.51 [deg].

Figure 4.7: A description of how the parameter α affects the resulting occupancy grid.

Secondly, different values of the parameter β were used in Figure 4.8. One can see how the cone that is wrapped around each beam increases in cross-sectional area with a higher β-value.
Chapter 4. Results

4.1. Object Detection

(a) An Occupancy Grid Map generated from two LIDAR scans. \( \alpha \): 3 [dm], \( \beta \): 0.051 [deg].

(b) An Occupancy Grid Map generated from two LIDAR scans. \( \alpha \): 3 [dm], \( \beta \): 0.51 [deg].

(c) An Occupancy Grid Map generated from two LIDAR scans. \( \alpha \): 3 [dm], \( \beta \): 5.1 [deg].

Figure 4.8: A description of how the parameter \( \beta \) affects the resulting occupancy grid.

Thirdly, and finally, the parameters of \( l_{\text{occ}} \) and \( l_{\text{free}} \) generated different behaviour. The two parameters were chosen to be the same value but with opposite signs. The parameter \( l_{\text{occ}} \) was always chosen to be \( l_{\text{occ}} > 0 \), and \( l_{\text{free}} \) to \( l_{\text{free}} < 0 \). In Figure 4.9, different values of the two parameters were used. One can see that low values of the parameters generate a map in which the majority of cells are classified as occupied. When using higher values of the parameters, a larger contrast is generated faster in the resulting map.
4.2 Object Avoidance

The expected results of the Object Avoidance part were that a search tree would be constructed, directed to the goal. The tree was expected to maintain a safe distance to any occupied cells, resulting in a free path to the goal. To validate and verify this, several tests were made, including the scenarios described in Figures 4.1 and 4.2. These tests are described in the following text and the implementation of the A* algorithm was conducted according to the theory presented in Section 2.2. It was implemented in Python with the parameters presented in Table 4.1.

Figure 4.9: A description of how the values of $l_{\text{free}}$ and $l_{\text{occ}}$ affect the resulting occupancy grid.
Table 4.1: Parameters used for the A* algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x)$</td>
<td>$</td>
<td>x - x_G</td>
<td>$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>1.8</td>
<td>[m]</td>
<td>Radius around the global goal that must be free from occupied cells.</td>
</tr>
<tr>
<td>$d_2$</td>
<td>1.2</td>
<td>[m]</td>
<td>Radius around each point on the path that must be free from occupied cells.</td>
</tr>
</tbody>
</table>

The initial tests were simply to pick a goal in the free space and perform a path planning. The tests showed an expected search tree and a path resulting in the specified goal node, see Figure 4.10. The goal node configuration was set to $[x, y, \theta] = [10, 10, 0]$ in [m] and [rad]. The end state of the vehicle was considered acceptable if it was within $\frac{\pi}{4}$ [rad] of the goal orientation and within 0.2 [m] of the goal position. The goal orientation is set to be within $\frac{\pi}{4}$ [rad] due to the start and ending angles of the motion primitives, as described in Section 3.2.1.

![Figure 4.10: A search tree with the blue dot as the start node, the red dot as the goal node and the black line as the chosen path. Note that the tree is constructed with a non-visible wall at top of the figure, hence the downward oriented search.](image-url)
More conclusive tests were conducted at the testing facility at Toyota Material Handling, according to the test scenarios shown in Figures 1.1 and 1.2. In Figures 4.11, 4.12a and 4.12b the results from straight-line tests are shown. The straight-line tests were conducted with an obstacle placed right beside (Figure 4.12a) and on (Figures 4.11 and 4.12b) the main path. The AGV then followed the main path until an obstacle was detected after which it deviated from the main path and rejoined further down the main path. Figure 4.11 shows that the AGV deviated from the path, with plenty of space in order to avoid the adjacent obstacle and rejoined using a Dubins path. The yellow dot in the figure is the AGV’s end position during the search which is roughly the same as the starting position. This gave the AGV enough knowledge of the map to generate a path from one search, i.e., it knew the area behind the obstacle was free. Figure 4.12a shows the same expected result, but without the Dubins curve as a last segment. As seen in the figures, the Dubins curve provided a much smoother transition back to the main path.
Figure 4.11: The occupancy grid of the lab in which the test was conducted. The yellow dot is the AGV’s position at the end of the test, the yellow oval is the main path, the black line is the path generated by the A* algorithm and the red line is the last segment of the path and is a generated Dubins curve. The starting position of the AGV was roughly the same as the end position, which was why the area around the obstacle was known during the search. This allowed the AGV to generate a complete path. The AGV finds the obstacle in the middle of the main path and avoids it as planned. See Figure 4.12b for a zoomed in picture of the path and search tree.
Chapter 4. Results 4.2. Object Avoidance

(a) Zoomed in on the path from a test without the Dubins curve as an ending segment. The requirement for reaching the goal node was to be within 0.2 [m] and not deviate from the goal angle more than $\frac{\pi}{4}$ [rad]. This method does not allow a very smooth transition to the main path, which is why the Dubins curve was introduced to generate the last segment.

(b) Zoomed in on the path from Figure 4.11. The path generated by A* is seen in black and the Dubins curve is seen in red. The vehicle deviates as expected and smoothly rejoins the main path. The vehicle’s position at the end of the test is represented by the yellow dot. The local goal is represented by the blue dot and the global (resampled) goal is represented by the red dot.

Figure 4.12: Two different straight-line cases. In 4.12a a Dubins curve is not used as an ending segment whereas in 4.12b a Dubins curve is used as an ending segment.

To test how the AGV handles an unknown environment, two obstacles were set up after each other. The first obstacle hiding the second one, forcing the AGV to do multiple searches and resample the global goal. In Figure 4.13a it is demonstrated how the AGV samples a global goal. From the starting position, only the first obstacle was visible and the global goal is chosen with respect to that, treating unknown area as free. An A* search is performed to the global goal and a path to the last known free point (local goal) was
sent to the AGV. From there, a new global goal is required to safely avoid the second obstacle. Note that the scanner has limited performance, and the local goal is chosen after only one scan of the environment. Therefore, there are some unknown cells close to the AGV, which is why the local goal appears to be chosen closer than necessary.

Two complete loops around the main path with the same setup are shown in Figure 4.13b. During the first loop, three searches were conducted which are visible as the three black consecutive segments. During the second loop the environment was hence known, allowing the AGV to generate a complete path during a single search. This is shown as the second continuous blue line, followed by the Dubins curve represented by the red line. The reason for the much more deviating path during the first loop is due to uncertainties in the map, i.e., some cells may appear to be occupied when they are not. This also explains the much steeper approach to the nominal path. The steep approach may seem strange in this particular test, but it is important to keep in mind that this is not a normal nominal path for the AGV. A normal nominal path normally does not contain U-turns, as seen here. The behaviour is therefore correct, since it chooses a global goal as close after the last obstacle as possible.
Chapter 4. Results

4.2. Object Avoidance

(a) The figure illustrates the principle of when the global goal needs to be resampled. The global goal (red dot) is sampled at the start of the search. It was located in an unknown environment at that point in time and hence considered free. The path generated by A* (black line) was sent to the AGV to the local goal (blue dot). From the local goal, the global goal was visible and was determined to be resampled. A zoomed in image of the path and search tree is visible in Figure 4.14a.

(b) Two consecutive runs, where the first was conducted in an unknown environment. The three separate, consecutive black lines are from the first loop, where the occupancy grid had to be updated several times to allow the path planner to find a complete path. During the second loop, the environment was already known which allowed the planner to find a complete path in one search. This is shown as the continuous blue line, followed by the Dubins curve represented by the red line. A zoomed in image of the path and search tree is visible in Figure 4.14b.

Figure 4.13: Demonstration of a search in an unknown environment.
4.2. Object Avoidance

(a) A zoom-in of Figure 4.13a. The black line is the path generated by the A* algorithm, with the red dot chosen as a global goal. The path was, however, only sent to the local goal (the blue dot), since that was the last known free point in the occupancy grid.

(b) A zoom-in of Figure 4.13b. The three consecutive, black line segments are the paths generated by the A* algorithm during an unknown search. The continuous blue line ending with the red segment is the path generated during a search when the environment is known. The red segment is a Dubins curve generated to provide a smooth transition to the main path. The slight deviation between the path segments from the unknown search are due to uncertainties in the positioning software.

Figure 4.14: Zoom-in of Figure 4.13

The tests for Scenario 2 in Figure 1.2 showed that the AGV behaves similarly as for Scenario 1. In Figure 4.15 a test regarding a curve is shown. The AGV found a way around the obstacle with ease, i.e., the constructed search tree is small. A second test would have been preferred using a hidden global goal, i.e., a blind curve. However, the possibilities for testing this scenario were limited due to lack of material in the testing facilities. It is although
assumed that this would not be an issue, since all other scenarios were handled without major issues.

(a) A test regarding a curve. The path generated by A* is seen as the black line, and the red line is the Dubins curve generated to provide a smooth path to the main path. As seen, this case is easily handled by the application.

(b) A zoom-in of Figure 4.15a. The path generated by A* is the black line, and the red line is the Dubins curve generated to provide a smooth transition to the main path.

Figure 4.15: A test regarding a scenario involving a curve.
5 Discussion

A general and in-depth discussion about the findings is presented in this chapter. The discussion is divided into two parts, Object Detection and Object Avoidance in Section 5.1 and Section 5.2 respectively.

5.1 Object Detection

The Object Detection part of the project performed well for the application. The general features of the testing area were possible to capture in the occupancy grid maps. However, some minor noise in the map was noticeable as discussed in the next section.

5.1.1 Rotational Drift in the Occupancy Grid Map

The noise that was noticed in the map was a result of a rotational error that appeared in the corners of the map, see Figure 5.1.

![Figure 5.1: An occupancy grid map over the free testing area, with an enlargement of the top right corner segment.](image)

This drift, or overlapping, of the map increased with the speed of the AGV.
and the effect was especially apparent when the AGV traveled along a curved trajectory. The first thought was that the effect could be a result of the 0.5 [m] scanner offset, that it has in relation to the wheel base. But the offset error was adjusted for numerically, and with the effect still present it was concluded that something different caused this unwanted drift. It could be a result of the time-delay between obtaining the information about the AGV state, and then obtaining the scanner data. In a sharp turn, it is possible that the angular velocity of the scanner was too fast for the software to follow, as illustrated in Figure 5.2. If this is the true cause, the scanner data and the positional information would not match and therefore cause a drift.

Figure 5.2: A schematic view of how the AGV moves between the point where it receives the state \( t = 0 \) and when the AVG starts recording data from the LIDAR \( t = t1 \).

Another possible error could lay in the software providing the algorithm with the state. Given that the test area changed frequently (vehicles were moved, doors were opened, etc.), it is likely that it affected the position confidence and through this returned some inaccurate states due to the environmental noise. Yet another possible source behind this drift could be in the Obstacle Detection software itself that might have been overlooked. When developing the mapping algorithm, the drift of the scanner was taken into account and each state used in the algorithm was displaced with 0.5 [m] in the heading direction for the AGV. This solved the issue with the scanner offset from where the position was given (center of the wheelbase), but might have created numerical errors or some other issue.
5.1.2 Parameter Design in the Occupancy Grid Map

As presented in the results, different values of the parameters for $\alpha$, $\beta$, $l_{occ}$ and $l_{free}$ were investigated. The parameters that generated the best map with the presence of the previously discussed angular drift can be found in Table 5.1.

Table 5.1: The table contains the final configuration parameters used in the occupancy grid algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_{max}$</td>
<td>40</td>
<td>[m]</td>
<td>The parameter defines the maximum length the scanner can register readings from.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.3</td>
<td>[m]</td>
<td>Assumed width of an obstacle.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\frac{0.51\pi}{180}$</td>
<td>[rad]</td>
<td>The opening angle/angle resolution of the scanner.</td>
</tr>
<tr>
<td>$l_{occ}$</td>
<td>$\log \frac{0.90}{0.10} = 2.2$</td>
<td>[-]</td>
<td>The a priori assumption of cells being occupied, used as a design parameter.</td>
</tr>
<tr>
<td>$l_{free}$</td>
<td>$\log \frac{0.40}{0.60} = -0.41$</td>
<td>[-]</td>
<td>The a priori assumption of cells being free, used as a design parameter.</td>
</tr>
</tbody>
</table>

With this configuration, the generated occupancy grid maps were somewhat robust and unaffected by the angular drift. With $\alpha = 3$ [dm], the obstacles were thick enough to avoid undesired disappearing. When $\beta = \frac{0.51\pi}{180}$, it gave the detected obstacles somewhat smooth structures and prevented the algorithm from converting straight walls into jagged lines, as seen in Figure 5.3.
5.1. Object Detection

(a) An example of how a too large value of $\beta$ can affect the creation of maps. In this figure, a laser beam is covered by the cone that the $\beta$-parameter creates. The beam hits the edge of a box.

(b) An example of how a too large value of $\beta$ can affect the creation of maps. A beam that hits the edge of a box creates a virtual circular edge, instead of recording it as a single pixel obstacle.

(c) An example of how a too large value of $\beta$ can affect the creation of maps. In this figure, two beams have hit the edges of the box, which creates a jagged obstacle representation instead of true representation of a straight line.

Figure 5.3: Demonstration of how $\beta$ affects the creation of the occupancy grid.

With a relative low value of $\beta$, this behavior was avoided with the trade off that fewer cells were classified as free. However, the Bresenham-extension of the inverse sensor model compensated for this shortcoming in a satisfactory way. The values of $l_{occ}$ and $l_{free}$ were selected to different values in contradiction to what was mentioned in Section 3.1.2. This was also a counter measure to deal with the rotational drift. With $l_{occ}$ roughly 5.5 times greater than $l_{free}$, the algorithm needs five free scans for an occupied cell in order to change the classification from occupied to free. This prevented occupied cells from vanishing in the map when the rotational drift was present, which enabled the avoidance algorithm to function properly. The parameter $z_{max}$ was in accordance to what the scanner could detect. This parameter filtered out cells that were outside the scanner’s range. Given that the test area was fairly small, $z_{max}$ had to be very limited to notice how it affected the algorithm. In Figure 5.4 two different values of $z_{max}$ were used and one can see how $z_{max} = 5 \text{ [m]}$ creates a smaller exploration sphere around the AGV.
(a) The map created when using a small value of the parameter $z_{\text{max}} = 5$ [m].

(b) The map created when using the selected value of the parameter $z_{\text{max}} = 40$ [m].

Figure 5.4: The parameter $z_{\text{max}}$ is the maximum range of each beam from the scanner. It directly affects how much of the map that is updated to be either occupied or free.

5.1.3 Computational Efficiency of the Occupancy Grid Map

The final version of the occupancy grid map took approximately one second to initialize and 0.3–0.4 [s] to update after each scan. This was the most time demanding computational part of the application. Several methods could be applied to address this, and they are discussed in Section 5.3.
Chapter 5. Discussion

5.2 Object Avoidance

As previously mentioned, path planning can be conducted in a number of ways. However, by considering that the environment is a confined space and the occupancy grid discretizes the world of operation, it was intuitive to use discretized methods for the path planning. The results showed that this worked sufficiently well and the findings are discussed in the following sections.

5.2.1 Lattice Planning

The use of lattice planning was considered a suitable choice for the application due to the discretization, and this was deemed true by the results. However, with the current state discretization and set of motion primitives, it does not possess enough precision to properly transition to the nominal path on its own. This is why the last segment constructed by the Dubins curve method was necessary. This is a direct result of the discretization, since the nominal path may occur in between the connected nodes. The nodes are connected by the motion primitives, and the choice of primitives were motivated by keeping the number of primitives as low as possible for computational efficiency, while maintaining the number of solutions to fulfill the relaxed requirement for reaching the global goal. Therefore, the primitives were constructed to be as short as possible with the smallest turning radius, which expectedly should have provided a flexible planner. While this was true to some extent, the AGV used more turns than expected which had a negative effect on the occupancy grid as explained in Section 5.1.1. While the turns may provide the shortest path, it is not desirable for a load-handling device to turn more than necessary due to risk of damaging the goods, even if the speed is set properly. A solution to this could have been to introduce a cost for turning, forcing the A* algorithm to prioritize straight motion primitives. This would, however, further limit the number of solutions and was not implemented. Instead, while in Avoidance-mode, the updating of the map was paused and the updating was limited to in between A*-searches while the AGV was standing still.

Shortening the length of the motion primitives even more was attempted, but did not provide good results. The generated path was not continuous, causing unwanted behaviour from the AGV. The cause of this is not entirely
clear, but an explanation could be that the shortening of the motion primitives caused the end points, i.e., the next states, to occur in the same cell in the occupancy grid, see Figure 5.5 for illustration. This could cause the planner to incorrectly backtrack from the goal node to the start node.

Figure 5.5: Using too short motion primitives might cause problems for the path planner. If the end points/next states of the motion primitives occur in the same cell in the occupancy grid, the wrong motion primitive might be chosen. This could cause non-continuous paths represented by the thicker line.

5.2.2 The A* Algorithm

The prediction of the A* algorithm being sufficiently computationally effective was deemed true, and the time for a single iteration was 2–3 [ms]. The limiting factor for the application was, as already mentioned, the processing of the data from the scanner that was 0.3–0.4 [s] per scan. Therefore, increasing the number of primitives would have negligible effects on the computational time compared to the number of solutions gained.

Furthermore, the solution is adapted to only consider the shortest path. Since the velocity is constant during Avoidance-mode, this is equivalent to the fastest trajectory. However, there could be situations where a longer path might be preferred to provide a smoother, less complicated path, see Figure 5.6. This is especially important if trajectories were to be considered and the dynamics of the AGV would be integrated in the planner.
5.3 Future Improvements

While the current application provides a sufficient solution for the given problem, there are many possible extensions and improvements. With more knowledge about the system’s functions and limitations, it is easier to visualize how the application could have been developed to increase the solution’s efficiency. This section aims to provide anyone who is about to embark a similar task or perhaps further improve this very application with valuable insights.

5.3.1 Object Detection

As discussed, it would be beneficial if the occupancy grid was more computationally effective, since it is among the more computationally heavy
segments of the application. To speed up the program, one could choose when to update the map in a more clever way. In the final implementation, the map was updated each iteration, regardless if the entire map was explored or not. By introducing logic that only updates the map when needed, and perhaps storing this knowledge for future access could greatly reduce the need for updating the map. When implementing this improvement, one can add a rule to only update the map when the vehicle is at a standstill. This would hopefully reduce the effects of the rotational drift, and further increase the general quality of the map.

A second approach to reducing the occupancy grid map’s computational complexity would be to create a local occupancy grid map over the obstacle avoidance segment, instead of mapping the entire area of operation. By assuming that the obstacle is within two meters, and that it will be successfully avoided after five meters, the algorithm could operate in a smaller square of say ten by ten meters, which would generate 10 000 cells with a grid resolution of 1 [dm]. This could even be modified such that the map is created only in the near distance of the original path, thus even further reducing the cells in the grid map.

Thirdly, one could experiment further with the parameters to try optimizing the computational speed. For example, by using a grid resolution of 2 [dm], it would reduce the amount of pixels with a factor of four (given that it is two dimensional, $2^2 = 4$).

Fourthly and finally, one could investigate if the modified inverse sensor model could be further improved. Bresenham’s line algorithm has been studied in terms of efficiency, and in the literature one can find alternative modifications that can reduce the needed computational power.

### 5.3.2 Object Avoidance

The path planning during this project was conducted with the assumption that the speed of the AGV was constant and that the world of operation was static. However, to increase efficiency it would be beneficial to plan with regard to the full dynamics of the vehicle. This would be an interesting challenge, since the AGV is primarily a load-handling device meaning that the weight would have to be measured, or estimated, online to provide the right dynamics for the path planner. This might also aid the vehicle in
taking smoother paths, not taking so many turns and provide better prerequisites to transition smoother to the main path. The next step would then be to account for moving obstacles. This is, however, a much more complicated extension, especially in an environment with humans, due to unpredictability. If the area is restricted to robots only, a solution would be to introduce connected vehicles which would give the robots knowledge of each other’s position, movement and updated knowledge of the environment.

However, this is a substantial change to the current solution, but there are simpler investigations that can be made. For example, changing the motion primitives would be the first step to investigate if better, i.e., smoother, paths can be constructed.
6 Conclusions

This thesis examined the possibilities of detecting and avoiding obstacles using a LIDAR sensor for an AGV. The solution was to construct an occupancy grid map representation using an inverse sensor model from the sensor data. The occupancy grid was then utilized in the path planning, by using the A* algorithm to search the grid for free space. Combined with lattice planning and the developed approach for choosing the goal configurations, a feasible path could be found. To properly re-connect to the nominal path, Dubins curves were used. Conclusions made about the findings are presented here.

6.1 Object Detection

The occupancy grid was expected to provide a solution for this application, and the results showed that it performed well (besides some minor errors and noise). The occupancy grid provided an accurate representation of the environment, which was fundamental for the logic used in the Object Avoidance algorithm. However, it is computationally demanding in the current configuration, given that the whole domain of the operational area had to be encapsulated inside the grid map. One example that visualizes how this could create unnecessary computations is if the algorithm would be used in an L-shaped corridor. To encapsulate the corridor, a box would have to be created which results in a lot of grid pixels that never would be used.

This issue could be redeemed with some smaller modifications of the current application. For example, by creating local maps when a collision course is detected instead of mapping the entire environment would greatly reduce the number of grid cells needed. In addition, this information could also be stored in a way that it could be accessed when the AGV re-encounters the same obstacle a second time. It would reduce the need to run the application every time the AGV encounters a previously explored obstacle, which would
6.2 Object Avoidance

While the prediction of the A* algorithm being sufficient regarding time and optimality was deemed true from the results, it can not be ruled out that other algorithms might have performed better. Furthermore, a better choice of primitives would have provided a smoother path that is more desirable for a load-handling application even though the shortest path was theoretically found, given the constraints on movement and closeness to obstacles.

6.3 Concluding the Problem Statements

- How can the available data be used to take safe decisions about the future movements, for the AGV, in the environment it is operational within?

By implementing an occupancy grid map algorithm, it is possible to generate a map representation of an environment and capture previously unknown obstacles. That map can then be used to take decisions about a vehicle’s future movements and guide it through the free space, ensuring a safe transportation of goods.

- How can the path planning be done in such a way that the deviation from the original path is as short as possible, without activating the safety fields, given that the depth and shape of the obstacle is unknown?

The Object Avoidance part was deemed successful for the application. All goal states and constructed paths were chosen with a sufficient safety margin to the obstacles, allowing the AGV to avoid them without risk of activating the safety fields. This is done using the method to ensure collision-free paths, along with the developed frontier-inspired solution to choose a goal.
configuration described in Sections 3.1.3 and 3.2. Furthermore, the A* algorithm provides the shortest path with regard to the constraints on movement and distance to obstacles thanks to the admissible and consistent heuristic. Lastly, a smooth transition to the main path was successfully performed with a Dubins curve, after the obstacle was circumvented.
Bibliography


Ljungqvist, O. et al. (Dec. 2019). “A path planning and path-following control framework for a general 2-trailer with a car-like tractor”. In: *Journal*

Nguyen, V. et al. (Aug. 2007). “A comparison of line extraction algorithms using 2D range data for indoor mobile robotics”. In: Autonomous Robots 23.2, pp. 97–111. ISSN: 1573-7527. DOI: 10.1007/s10514-007-9034-y (Visited on 02/05/2021).


