Høgskolen i Østfold

BACHELOR THESIS

Autonomous Racecar – Mapping of buildings
Group nr BO22_G35

Fredrik Sundre Lauritzen
Magnus Willy Eilefsen
Kristoffer Snopestad Søderkvist
Olav Aleksander Nøklestad
Kristoffer Eriksen Beck

School of Computer Sciences
Østfold University College
Halden
May 20, 2022

Autonomous Racecar – Mapping of buildings

Group nr BO22_G35

Bachelor Thesis 2022

Fredrik Sundre Lauritzen
Magnus Willy Eilefsen
Kristoffer Snopestad Søderkvist
Olav Aleksander Nøklestad
Kristoffer Eriksen Beck

School of Computer Sciences
Østfold University College
Halden
May 20, 2022

Abstract

By using a modified radio-controlled car, outfitted with a LiDAR for 2D scanning as well as a 3D camera to aid. The task given to us was to map areas using this car while having it driving autonomously. Following F1tenth's guide on setting up the car, as was mentioned by our task handler.We were able to achieve 2D mapping and test a basic safety mechanism by having the car stop once it reaches within a certain meter range. Research has been done into the next steps in terms of automation as well as 3D mapping and picture analysis.

Acknowledgments

The group would like to thank our supervisor Ole Edvard Ørebekk for guidence and advisor on the thesis. We would also like to thank Michael A. Lundsveen for the challenging and interesting project. Finally we want thank Carolane Faure for reading and putting her time and effort of correcting spelling grammatical mistakes on our bachelor thesis.

Contents

List of Figures

List of Code

Chapter 1

Introduction

In a society that is rapidly evolving, the need for greater efficiency is increasing in many fields, especially solutions that facilitate and utilise our increasingly advancing technology. Recent developments in technology and machinery has made it possible to streamline or even fully automate processes that would previously be done by humans. Automation is a concept where the goal is to fully automate systems and processes, where minimal human involvement is the ultimate ideal.

In production lines, automation allows for a better use of resources and materials used in yields for higher quality products. It helps increasing the efficiency in organisations and corporations where utilising automation can help reduce work loads on employees.

Automation also encompasses a great variety of technologies such as robotics, telemetry, sensors, wireless applications and a lot more. When it comes to emulating practical or physical human tasks and processes, the robotics field is arguably the most relevant. Common human-related tasks that robots often attempt to emulate include walking, lifting, speech, cognition, driving and many others.

Driving is one activity that demands particularly a lot of involvement from humans. As a result it is also an activity that often involves relatively high risk and can sometimes present hazardous or even fatal outcomes. All it takes is a moment of distraction or a short lapse of judgement for driving to have fatal consequences. People are always vulnerable in the sense that mood, sleep, emotional well-being and mental health can impact driving ability.

In light of this, self-driving cars has been receiving a lot of attention especially the last decade. Emulating the senses and skills required for driving vehicles is no easy task, but the outcome could be widely positive. Robotics could successfully eliminate all the human risk factors that often presents itself during driving.

In order to research self-driving cars in a safer and more controlled environment, the use of smaller autonomous RC vehicles has been increasing. These vehicles still allow us to research the same technology that would also be used on a full-scale self-driving car. In our project we will be using such an RC vehicle equipped with a LIDAR sensor. The ultimate goal is to make the car drive and follow a path by itself, with minimal to no human involvement. [63] [94] [69] [79]

1.1 Project group

This group is made up of five 3rd year computer science students,

1.1.1 Fredrik

27 years old from Tønsberg. Currently studying Computer Science at Østfold University College, specializing in machine learning. Academic interests I have consist of machine learning, image analysis and programming. Interests outside of academics include video games, computers,technology, golf and friends.

1.1.2 Olav

25 years old living at Begby in Fredrikstad. Currently working on a bachelor's degree in Computer Science while specializing in computer security. Mostly experienced with Java, JavaScript with some experience in C and some other script languages such as AutoIt, GDScript and Python. Enjoy working on programming related projects that may have some practical usage or solve problems.

1.1.3 Magnus

25 years old from Fredrikstad and working on a bachelor's degree in computer engineering. Academic interests consist of programming and computer networking. Other interests are computers, video games and technology.

1.1.4 Kristoffer S

24 years old and living in Halden. Right now working on a bachelor's degree in computer science. Subjects I enjoyed the most were mobile programming which delved into Java, and web applications which delved into Next.js, JavaScript and HTML elements. Have some experience with C#, Python and some C. Otherwise interested in motorcycles, video games and technology.

1.1.5 Kristoffer B

24 years old from Halden, working on a bachelor's degree in computer science with a specialization in programming. Mostly experienced with Java and JavaScript, but have used C#, C and Python in earlier subjects. Interested in pretty much anything that can be considered technology.

1.2 Task Owner

Our assignment is part of the list of projects that were presented for all students belonging to the Institute of information technology and communication, spring of 2022. The assignment and overall project specification is defined internally by ØUC's IT-department and was given to our group by chance. Our client in this case would be the IT-department at HiØ, but more specifically Michael A. Lundsveen that would be our main contact during the project period.

Østfold University College currently reside among two modern campuses, one in Halden (Remmen) and Fredrikstad (Kråkerøy) and offer over 100 different studies. In total there are roughly 7700 students and 620 employees that belong to different departments depending on position and course of study. Østfold University College also offer opportunities for studying abroad, as well as further added studies.

Michael Andersen Lundsveen is chief engineer at the department for information technology at Østfold University College. He is responsible for overseeing ØUC's MakerSpace, FabLab, the podcast room, Duckietown, Autonomous vehicles lab and the Virtual reality lab (VR).

1.3 Purpose, Deliveries and Method,

Our goal with the project, definition of the problem and objectives of the report and approach.

1.3.1 Purpose

The purpose of this report and thesis is to establish how to make a 2D map, and how to make the vehicle drive by it self on a map that's been made. As mentioned in the assignment, Michael Andersen Lundsveen is looking for a solution on how to maneuvering a 1/10 scale vehicle thru maps that have been mapped with a LIDAR sensor, with out the input from a user. To achieve this initial plan, developing a machine learning model to train the vehicle to drive by it self. The thesis will contain answers to why the initial plan did not entirely work, and why the group had to research other option to achieve the objectives.
The main goal for this project is to establish how to make a map using a 1/10 scale vehicle with a LIDAR sensor attached in the front, and then research how to make the vehicle drive by it self without input from the user. Its goal is also to analyse the necessary research on existing solutions that will lie the groundwork and foundation for possible solution for the problem described.
Objective 1 Make a 2D map of an area or a track that is all ready been made.
Objective 2 Develop a method for the vehicle to stop by it self if it comes to close to any hindrances in the way or walls. Developed as a safety feature to not ruin the car while it drives.
Objective 3 Develop and use a path planning algorithm, to make the car drive by it self on a map made by us or and given map.
Objective 4 Develop and use Machine Learning to make the car drive by it self, on tracks or inside rooms in buildings.
Objective 5 If it is no time left for development our research will be used for a background for further development.

1.3.2 Deliveries

What are the goal for the deliveries and what is to be expected on the final results.

Bachelor thesis

Our approach for the thesis will begin with introducing our objectives and problems presented by Michael Andersen Lundsveen. That will set basis to better understand which technologies, sources and articles we want look into and examined.
Then our findings will be summarized, analysed, and will describe how to develop certain aspects of the research.

At the end of the thesis, we will form a conclusion based on what we have done and our progress regarding the development of our thesis.

Development towards a self driving vehicle

The process of development of a self driving vehicle will start by looking up relevant research to our assignment and documentation on problems similar to ours. Then, we will start the process of manufacturing maps we can use to test our vehicle with a controller. After this, we will test and develop a safety mechanism so that the vehicle will not go thru any accidents, and finally we will attempt to test the self driving system.

1.4 Report Structure

Here will present how our report is build up, and give a small description of each chapter.
Chapter 2 Presenting different technologies, similar concepts related work similar to our project on the vehicle.
Chapter 3 Presenting the project description and some highlights from testing and some results.
Chapter 4 Goes over the planning phase of the thesis. It starts with the initial plans, The chapter concludes with what the plan is forward.
Chapter 5 Is about the first time setup of a F1tenth project and what to do if problems occurred.
Chapter 6 Begins with how to install what technologies we have been using and ends with some problems that might occur.
Chapter 7 In dept on algorithms and how it works and the chapter ends with our implementation.
Chapter 8 Concludes the thesis, where group comes with a conclusion of the work that's been done.

Chapter 2

Background

In this section, we will present some of the tools and technologies that are either necessary or relevant to our project. Some of these topics will involve algorithms, programming languages, operating systems, related projects and other tools and technologies related to NVIDIA, path finding, machine learning and robotics.

We will also examine some of the differences among similar topics such as various algorithms and programming languages, their strengths and weaknesses as they pertain to this project.

2.1 Robot tools and technology

Here, we will highlight similar robotics projects and the different technologies that are often necessary or helpful when trying to configure autonomous vehicles. These projects seek to aid the growth of the autonomous robotics field by building communities and connections with experienced people. Competitions or contests are also hosted that allow for practical learning. The technologies outlined in this section are relevant to both general and autonomous robotics. These technologies can be either hardware or software components used by our own vehicle or similar robots. These components are what allows the vehicle to know where it currently is on the map and the surrounding area among other objects.

2.1.1 F1tenth

F1tenth[1] was originally founded at the University of Pennsylvania in 2016, but has since spread worldwide to other institutions. It is now an international community of researchers, engineers, and autonomous systems enthusiasts. F1tenth is a competition in robotics where teams use the same type of one-tenth scale autonomous race car.Their goal is to optimize algorithms to preform the best in the shortest amount of time in randomized racetracks. To get around the racetrack and to develop the car, different tools and technologies are used such as LIDAR, SLAM,ROS, Rviz and Gazebo. [87] [80]

[1] `https://f1tenth.org/about.html` part of our thesis is based on this project

2.1.2 Duckie town

Duckie town is a project that is based on a course made at MIT in 2016. The project became successful and has gone global over the years. For many people this will be their first encounter with automation, robotics and AI development. The premise of the project is to teach students about automation using AI by using affordable, modular and scalable autonomous vehicles. It gives students the freedom to solve the problems however they see fit. [12] [86]

2.1.3 LIDAR sensor

LIDAR[2] ("light detection and ranging" or "laser imaging, detection and raging"), sometimes simply referred to as 3-D laser scanning is a sensor that can determine the ranges and distances of objects. It does this by targeting objects with its laser and then measures the time it takes for the light to reflect back to the receiver. The sensor is also capable of mapping areas in 3-D, by using different laser wavelengths and by calculating the differences in laser return times. This technology has uses and applications in many different markets where its mapping capabilities can be utilized: surveying, archaeology, geography, geology, seismology, and forestry among other applications. The technology can also be used to control and navigate autonomous cars, which will be our main usage. [76]

2.1.4 Rviz

Rviz is short for ROS Visualization, and is a graphical interface that can be used to visualize information with topic-specific plugins for ROS. This tool allows us to run simulations where we can see the surroundings from the robot's perspective. The software will attempt to recreate the surroundings and current environment of the robot based on the robot's sensor data. The use of plugins such as GRID, Robot Model, and TF allow the user to get a more detailed graphical interface for ROS. These plugins also provide additional useful information about the state of the robot and sensor data. [33]

2.1.4.1 GRID

GRID is one of the available display types in Rviz. As the name implies, GRID will display a grid of lines that can remain visible while running simulations. A grid has multiple optional properties for improved visualization such as cell count, cell size and line color. One benefit of a grid is that it functions as a constant static point of reference. This can make it easier to gauge how much the robot has moved during a simulation. [102] [6]

2.1.4.2 URDF

URDF is an Extensible Markup Language(XML) file format that represents the model of a robot. Users can create models by defining certain properties such as height, width or radius. These models can vary from basic geometric shapes to complex jointed models with kinematic capabilities. The URDF file can be loaded into Rviz which will then attempt to visualize and display the model. [43] [89]

[2]https://en.wikipedia.org/wiki/Lidar/

2.1.4.3 Robot Model

Robot Model is another optional display type available in Rviz. This display type allows the user to visualize a model according to the properties provided by a URDF file. Once a model is loaded the user can browse through additional information and settings of the model. [28] [43]

2.1.4.4 TF

To navigate a robot in a space, certain computations are required. These computations are rooted in fairly complex mathematics and can be quite demanding to work with. ROS TF (Transform) is meant to simplify this process by providing an easy-to-use package that handles these complicated calculations. More specifically the TF package shows where the position and orientation of all the frames that are part of the TF tree. [39] [30]
[19] [103] [21]

2.1.5 Java Virtual Machine(JVM)

JVM is a virtual machine that makes it possible to run Java applications regardless of the underlying platform or hardware. Normally with most programming languages, the code is converted into machine readable code. Java on the other hand is converted into Java bytecode, which can then be read and executed by the JVM. Any application that can be converted to Java bytecode can run on a JVM, regardless of the original language that was used to write the application. A clear specification and description of what is required of a JVM helps ensure that future applications launch without fail, no matter the platform.
[75] [59]

2.1.6 SLAM

SLAM (simultaneous localization and mapping) is a method for autonomous vehicles that allows the user to build a map and simultaneously map the vehicle's surroundings. The algorithms used by SLAM enable vehicles to map unknown environments. Using map data, engineers execute tasks like path planning and obstacle avoidance.

2.1.7 Hector SLAM

The Hector SLAM contains `hector_mapping` as well as all related packages. Hector allows for a mapping of the environment in 2D and displays information about the obstacles and available paths. It allows the user to fetch out information needed to map out a path for the vehicle to drive. One does not need to use the parameter odometry while using hector slam, since it allows to use other packages to tell where the position is over time.

2.1.7.1 Visual SLAM

The application acquires an image from a camera and other image sensors. Based on the information in the image, it calculates the position and orientation of the device while continuously gathering data and mapping the environment.

2.1.7.2 LIDAR SLAM

Laser sensors are used to generate an environment. According to what type of information is needed, the environment can either be 2d(x,y) or 3d(x,y,z). With that information, it knows the distance from each object and generates a mapping of the environment. [109] [91]

2.1.8 Gazebo

Gazebo gives the user the ability to create 3D scenarios with robots, obstacles and other objects on the computer. It contains a physical engine for lighting, gravity, inertia and more. It is possible to evaluate and test one's robot in difficult and dangerous situations without causing any harm to the robot. Simulators are often faster than starting up the actual robot to run the simulation. [90] [49] [22]

2.1.9 Webots

Webots is a software package for simulation of mobile robots. An essential component of the software is a rapid prototyping environment that lets users create 3D virtual worlds with physics properties, such as mass, joints, friction coefficients, and more.

There are many types of robots capable of locomotion: A robot can be wheeled or limbed.

In addition to this, they may be equipped with a variety of sensors and actuators, such as distance sensors, cameras, motors, touch sensors, emitters, receivers, etc. Robots are programmed independently to carry out specific tasks. [107] [85]

2.1.10 Bullet/pyBullet

The Python module Pybullet implements the Bullet Physics SDK to simulate physics and automate robotics. It supports a number of sensors, actuators, and terrains. Fundamentally, this is a wonderful tool for beginners who want to get started in robotics or physics simulation. [51]

2.2 Operating system

Certain operating systems are required in order to communicate with our autonomous vehicle. ROS is technically not an operating system, though it contains necessary tools and libraries to run simulations, tests and send messages to our vehicle such as commands. We use a ROS 1 distribution named Melodic which targets Ubuntu 18.04 (Bionic), which happen to be the most compatible with our vehicles hardware.

2.2.1 Robot operating system

Robot Operating System(ROS)[3] is a free open source software that consists of a collection of libraries and tools for building robots and machine learning applications. Along with providing useful developer tools, such as providing drivers and top of the line algorithms and much more to ease the process of building applications.

[3]`https://www.ros.org/.html` ROS website

Behind ROS is not just the software itself, but also a large community[4] of people from around the world who have contributed to the project by sharing their solutions, resources, services and other relevant implementations. Some of these services include logging and diagnostics, fleet management, deep learning, data collection, testing, quality assurance and more. ROS delves into virtually any market where robotics can have useful applications, like agriculture, factory logistics, autonomous vehicles, service robotics, food preparation, heavy industry, drones and planetary exploration.

Most robots generally have three typical main components: actuators (objects that move), sensors (capture images that read the world) and control systems (the robot's brain). ROS is helpful to our project because it allows us to set up easy communication between all these different components in our automated vehicle. ”Topics” and ”messages” are some of the tools that can be used to communicate between the different components.

Another advantage with ROS is that it also provides visualisation tools useful for digital testing. This makes it possible to do simulated testing with a robot that would behave just like the robot outside simulations. Lastly, ROS also supports the hardware interface of the sensor we will be using called LIDAR.[64]

2.2.2 ROS Melodic Morenia

ROS Melodic is the 12th distribution of ROS. It's intended to support Ubuntu 18.04 (Bionic), Ubuntu 17.10 (Artful) and Debian Stretch. Melodic is one of the last distributions built around the first version of the Robot Operating System, ROS 1. We use Melodic for this project, as our vehicle only inherently supports this distribution of ROS 1. [16] [78]

2.2.3 OpenCV

OpenCV (Open Source Computer Vision Library) provides machine learning, computer vision, and pattern recognition capabilities. To increase the use of machine perception by commercial products, OpenCV was built to provide a standard environment for computer vision applications. Among more than 2500 algorithms in OpenCV's library one can find classic and newer algorithms for computer vision and machine learning. These algorithms include ways to recognize faces, objects, human actions, track camera movements and objects etc. It supports the interfaces of C++, Python, Java and MATLAB. Also supports Windows, Linux, Android and Mac OS. OpenCV focuses on real-time applications and takes advantage of MMX and SSE instructions as needed. There are currently efforts underway to develop fully-featured CUDA and OpenCL interfaces. [38]

2.2.4 Linux

Linux is an operating system created in the 1991 by Linus Torvalds, and was built as an open source alternative to Minix(Operating system). Linux is an operating system and have turned into one of the most used platforms. Android is built on top of Linux operating system. All hardware resources associated with the machine is managed by an operating system. [108] [27]

[4]`https://www.ros.org/blog/community/` ROS community

2.3 Machine Learning

Machine Learning(ML) is a branch of Artificial intelligence(AI) and computer science with a distinct aim on using data and algorithms to emulate the human learning pattern. This way ML algorithms can train data as input to predict output values. The accuracy can be improved over time, with gradual improvements by training the program

Data can be gathered and collected from image, video and sound detection software. This allows machines to emulate more specific human features, like voice and face recognition.

Deep learning and Machine learning have traditionally been a very slow and expensive process since training the AI can take hours and up to days to complete.

With the support of a Graphical Processing Unit (GPU) it is also possible now to accelerate the process.

[44] [42]

2.3.1 Tensorflow

Tensorflow is an open source AI framework for machine learning that is created and maintained by Google. With its robust use-case in differently fields inside of ML such as the use-cases listed below. [10] [23] [32] [104]

2.3.1.1 Voice/Sound Recognition

In sound-based applications, the use of neural networks with the proper data can be utilized for audio processing. One of the most common uses is voice-search and voice-activated assistants such as Apple's Siri, Amazon's Alexa, and Microsoft's Cortana. In most cases where speech to text applications are used, the application is used to determine snippets of sounds, and then the spoken words are transcribed into text. These sound-based applications can also be used for Customer Relationship Management(CRM). TesnorFlow algorithms will serve as a substitute for customer service agents. To accomplish this, the customer is required to answer some questions before speaking with an agent, and the algorithm will then identify the agent that can best meet the customer's needs. [10] [23] [32] [104]

2.3.1.2 Image Recognition

For applications such as Face Recognition, Image Search, Motion Detection, Machine Vision, and Photo Clustering. The goal is to recognize and identify people/objects as well as comprehend the content. Larger images are generally used in engineering applications as models, in 3D space construction, and for forming two-dimensional images. In social networks for photo tagging[5]. The algorithms are able to identify different types of cars by analyzing thousands of photographs of vehicles, for example. Now, image recognition can be used in the Healthcare Industry as well, where TensorFlow algorithms can now process more information and will be able to review scans and find more illnesses. [10] [23] [32] [104]

[5]`https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/explainTensorflow`

2.3.1.3 Video Detection

Neural networks now also work in motion detection. Some areas it used is for example in gaming and security and airports.
[10] [23] [32] [104]

2.3.2 Keras

Keras is a deep learning Application Programming Interface (API) in python developed by François Chollet, a Google engineer. By utilizing framework libraries such as TensorFlow and Theano. An API with an interface to help humans rather than machines, having reduced user actions for common use cases and as clear error messages as possible for the user to understand, and act upon.
[60] [41]

2.4 NVIDIA graphics

NVIDIA designs and Graphical Processing Unit (GPU) for both professional computing and gaming markets. For the professional market they have products for super computing and AI acceleration. The AI accelerators are made to accelerate the process used for machine learning applications. Among some of the product lines they are selling is the Jetson line, that is specially used for learning, building and teaching AI robotics.
[67] [98]

2.4.1 NVIDIA Jetson

Jetson is a collection of computing boards made by Nvidia that are all complete System on Module(SOM). Depending on the use case and the intended scope there are a few different series of modules to pick from such as Xavier, Nano and TX2. Each module includes a , , memory, interface, and a power management system, though the overall performance and power efficiency of the unit will depend on the series. Jetson is also a platform in a broader sense that provides services and SDKs to assist with the development of autonomous applications. The JetPack SDK is supported across all modules of Jetson, and is the software that will be installed on the autonomous vehicle used in this project. JetPack is a full software solution for Jetson modules that contains multiple components like OS images, libraries, APIs, samples, and developer tools.
The autonomous vehicle used in this project carries the Xavier NX module.
[50] [99] [53] [97] [61]

2.4.2 NVIDIA DeepStream SDK

A collection of streaming analytics software for AI-based multi-sensor processing, video, audio and image understanding. These tools can assist with performing Intelligent Video Analysis (IVA) by capturing and utilizing data from cameras and sensors around the world. The captured data can be used to generate useful statistics and other analytical graphs, or even alerts. A few of the many possible applications of these analytical tools consist of detecting component defects at manufacturing facilities, or surveying store aisles for improved customer satisfaction.
[62]

2.4.3 NVIDIA CUDA

CUDA is a parallel computing platform and programming model[6]. With the help of GPU, it drastically increases the performance during intensive applications. CUDA in deep learning has a need for computing speed. In some cases using GPUs instead of only the CPU to train the models can take months rather than a week. Also other frameworks like TensorFlow are requiring the CUDA packing to work. The cuDNN library for computations in neural networks. Training the model with deep learning and cuDNN is so important that essentially it does not matter which version of cuDNN that are in use, the performance are equivalent in every use case. When a new version of CUDA and cuDNN is released and the framework is updating to the new version tends to improve. Where difference and where the framework tends to differ is how well they scale to multiple nodes and GPUs. [14]

2.5 Visualizing and path planning

An autonomous vehicle will require an algorithm in order to evaluate its surroundings and any potential paths that it discovers. This section will further detail how the vehicle knows where it currently is and how different search algorithms dictate how the vehicle will attempt to solve path planning problems. Basic path planning works by making the robot take an assumption-based approach to making operations and attempting to solve problems.

2.5.1 Potential Field algorithm

Potential field algorithms follow Laplace's equation to accomplish their goal. The system consists of mapping out a space, dividing it into a grid, then filling each cell with information. This information includes the start node, the goal node, and obstacles. It assigns a value to each cell based on how likely it is to reach its goal. The start node has the highest potential, while the goal node has the lowest. The algorithm solves the pathing by going from highest to lowest potential and making the pathing based on this information. [20]

2.5.2 Self localization

A concept in robotics that entail how the robot knows where it currently is. [26]

2.5.3 Path planning methods

In terms of making something autonomous move and reach a set goal requires some form of path planning method. These have various ways of working and computing a path based on current location or potential objects blocking the path. [8]

2.5.3.1 Global planning methods

Global path planning uses an algorithm called Rapidly Exploring Random Trees (RRT). In the RRT algorithm, the first node is the starting point. The system also uses an existing map of the environment. It then generates a bunch of random samples that, if valid, connect to the start node. Every node in the tree will be connected to the next in the tree,

[6]`https://developer.nvidia.com/cuda-faq`explain CUDA

with the top node being the start node. After sampling, the shortest branch will be chosen. [17] [8]

2.5.3.2 Local planning methods

Local planning is conducted in real time without a prior knowledge of the landscape. It fetches environmental information in order to generate a simulated field where it can identify a path. This enables a path to be found in real time as well as change its path dynamically when new obstacles appear.
Here are some types of local planning methods
-Probabilistic Roadmap (PRM)
-Evolutionary Artificial Potential Field (EAPF)
-Indicative Route Method (IRM)
-Modified Indicative Routes and Navigation (MIRAN) [8] [15] [17]

2.5.4 Planning with Freespace Assumption

Planning with Freespace Assumption is to attempt to move the robot on a shortest potentially blocked path in a partially known terrain to the goal coordinates. This method can involve a lot of re planning since the robot has to update information as it discovers it. It first assumes that there are no obstacles, but if an obstacle is found it has to add that information to its map and plan a new potential path. For instance, the A* algorithm will evaluate only nearby nodes and move to the most attractive node based on a total evaluation score (cost). As a result, only nearby nodes are evaluated and unknown nodes are ignored until discovered. Freespace assumption on the other hand is a bit different. The cost between unknown nodes is the same as the cost of nearby known / free nodes. This is essentially what it means when we say that the method assumes there are no obstacles in the terrain; all nodes are treated as potential candidates that one may move to.

[79]

2.6 Path planning algorithms

Algorithms will also vary when it comes to efficiency based on whether or not the terrain is known and how fast they can observe and map obstacles. We will also discuss other further differences between each algorithm and elaborate on which particular algorithm might be the best to start with.

2.6.1 Dijkstra's algorithm

Dijkstra's algorithm is an older path finding algorithm and relatively common graph based algorithm. Its use is to find the shortest or easiest path from point A to point B by checking every possible node between A and B until a clear path has been found

As mentioned, it searches for the easiest path between A and B while checking all nodes until it reaches its destination. It does this by constantly checking new nodes around where it has recently checked, marking it as a clear/checked or obstacle node. If the path is considered clear/checked it will scan for other nodes around that one. If it's marked as an obstacle or previously visited, those nodes will be skipped from further actions. The reason it checks every potential node from A to B is because it considers the distance between

A and B as infinite, leaving it to check for as many potential paths as it possibly can to validate the shortest path to travel. In short the algorithm creates a direct line between point A and B, then tries to find the shortest path while marking out visited and obstacle nodes so it finds the path forward.

[72]

2.6.2 A* algorithm

A*(Pronounced ”A star”) is a variant of Dijkstra’s algorithm, where the logic is similar but executed in a different fashion by finding the shortest path available and refraining from scanning more nodes than necessary. It was the first published by Peter Hart, Nils Nilson and Bertram Raphael in 1968 and was part of a project called ”The Shakey Project” which was centered around giving mobile robots the ability to plan their own actions.

The algorithm will keep going in a straight line between point A and point B, should an obstacle appear in the path it will start to scan nodes around the obstacle until it finds the quickest unobstructed path towards point B. As long as there is an actual path it will only scan the least amount of nodes needed, only ever scanning in a broader pattern should an obstacle block the algorithm. This algorithm is faster than Dijkstra’s due to the latter using infinity between A and B, this being why A* does not scan or expand unless it has to.

[66]

2.6.3 D* algorithm

D* (pronounced D Star) is an incremental search algorithm developed by Anthony Stentz, though the term itself is also used as a broad descriptor for all the related algorithms; Focused D*, D* Lite and the original D* itself included. All the algorithms attempt to solve general path planning problems with an assumption-based method. They are also intended to handle path planning with freespace assumption where the terrain is unknown, and not partially known like with some algorithms. Ultimately the vehicle is supposed to reach a goal or destination based on coordinates, and tries to do so by making assumptions.

For starters, the vehicle might assume that there are no obstacles in the unknown terrain, and will therefore simply try to find the shortest path from A to B. The vehicle will then attempt to follow this path it discovered and then add any new information it observes to its map along the way. If a new obstacle is observed by the vehicle while it tries to follow a path, the vehicle may then have to replan a new route based on the updated map information, and again try to find a new shortest path. This process will repeat until the vehicle reaches its goal coordinates, or until it determines that the goal cannot be reached.

New obstacles can appear frequently in unknown terrain, so it’s important for the vehicle to be able to quickly re plan new paths. All the D* algorithms are useful in this regard as incremental search algorithms can speed up similar subsequent searches based on the experience of previous problems. In terms of efficiency, all the D* algorithms are considered to be more efficient than the A* algorithm, assuming the destination remain the same.

[71]

2.6.4 Focused D* algorithm

This algorithm is based on the original D*, but include some adjustments that changes how the algorithm prioritizes different states. The basic procedure of D* is to keep a list of nodes known as the OPEN list, and these nodes are to be evaluated at a later stage. A node can be marked with one of the following states: NEW, OPEN, CLOSED, RAISE and LOWER. A node marked with NEW means it has never been in the OPEN list, and a node marked with CLOSED means it is no longer on the list. RAISE and LOWER indicate that the evaluation score of the node is either higher or lower compared to when it last was on the list. Focused D* differs from original D* by making the robot focus more on the RAISE and LOWER states instead of updating other potentially less relevant states. [71]

2.6.5 D* Lite

D* Lite is another algorithm developed by Anthony Stentz, though despite its name is not exactly a derivative of the original D* algorithm even though it implements similar behavior. Instead it is based on another algorithm called Lifelong Planning A*, which is again based on A*. It's more accurate to say that D* Lite is an entirely rewritten algorithm that uses fewer lines of code compared to the original D*, and is also supposed to be easier to understand code wise. Performance wise it outmatches the original D* and can also match or sometimes outperform Focused D* as well. In more recent systems, Stentz very own labs seem to favor the D* Lite algorithm instead of the original D* in some cases. [71]

2.7 Programming languages

In order to write and implement a path finding algorithm, we need to pick a programming language. Our selection is somewhat limited since ROS currently only supports either Python or C++. We will however also discuss some of the strengths and weaknesses of Java. It is a popular programming language that arguably falls somewhere in between C++ and Python. It can therefore help us with establishing a solid baseline when comparing different languages.

For this project we decided to use Python. Python is a very user friendly language that receives a lot of support through various unofficial packages and libraries.

2.7.1 Python

Designed by Guido van Rossum and first released in 1991, it is one of the most popular programming languages in the world. Guido van Rossum was the lead developer of the language until he officially stepped down in 2018, and the language is now further developed and maintained by the Python Software Foundation. Compared to other popular contemporary languages, Python can be regarded as a relatively old language in comparison. However, the modern version is different enough from its original iterations to the point where the language is not backwards compatible with these older versions.

As a high level language, Python attempts to lessen the need for user based resource management often required in more low-level languages, such as memory allocation for objects. Another core design philosophy of Python and other high level languages is that they should also be easy to understand and learn and hide or automate more complex parts that may run behind the scenes. Compared to other languages, Python can also generally accomplish tasks with a lot less lines of code used.

Other defining properties of a scripting language compared to Java or C++ for instance, is that the code is interpreted at run time instead of being compiled into executable code. Another characteristic is the use of indentation instead of curly brackets to declare code blocks.

Python is intended to keep a small core language, but instead offer a large standard library of modules that can be implemented in projects. [110] [57]

2.7.2 Java

Java is one of the most popular programming languages in the world, arguably thanks to its platform independence and general coverage of a wide variety of devices. It was developed at Sun Microsystems by James Gosling and first released in 1995 mainly for use on their own platforms. In 2006 Sun decided to make the source code open source, allowing for world wide contributions from users, although the main supporting body would be Oracle Corporation.

The language itself is an object oriented language with the goal to have as few implementation dependencies as possible. Java code is not compiled to machine code like many other languages. The code is instead compiled to bytecode that can run on the Java Virtual Machine(JVM) which makes it possible to run Java applications on any platform that has a JVM. To further strengthen this design choice there is also a prominent underlying philosophy within the Java language to promise what is known as Write once, run anywhere(WORA).

Comparatively, Java has similar syntax to languages like C and C++, but generally less low-level facilities like memory management and allocation required. Java uses instead a garbage collector so that the user does not have to consider memory management, although other low-level facilities may still have to be manually managed. [74] [58]

2.7.3 C++

Development of C++ was initiated by Bjarne Stroustrup at Bell Labs in 1979, and first appeared in 1985. His goal was to enhance the C language by providing it with features from Simula, another language considered to be the first Object Oriented Programming(OOP)-language invented. From 1998 and on wards, C++ was later developed and maintained by the International Organization for Standardization (ISO).

Generally the code is fairly concise as opposed to being verbose. This means that C++ may not be as "self-documenting" as other languages, and can therefore be a somewhat challenging language to learn and is often a common sentiment. It is a low-level language built with performance, efficiency and flexibility in mind, and users may have to consider hardware limitations and resource usage when writing C++.

There is no garbage collector, so memory and size of objects must be manually allocated and managed. C++ is a powerful language especially in applications and projects where resources are constrained like desktop applications or video games. [70] [106] [105]

2.8 Related work

This section will contain any topics related to the tools, technologies, libraries and other components relevant to our project. Despite the fact that they may not directly relate to our project, they are relevant in other ways.

2.8.1 Roomba

Roomba is a computerized, compact vacuum cleaner that automatically navigates inside of the house, where dirt is picked up by spinning brushes and a vacuum, just like a conventional cleaner.
Roomba uses a wide variety of onboard sensors to map its terrain. The device has an infrared beam and a photocell sensor in front, as well as a touch sensor built in to the front plastic bumper.There is also an infrared sensor mounted underneath pointing straight down to detect so-called ”cliffs”(stairs and deep drops). The infrared beam in the front detects walls and obstacles so the Roomba knows when to slow down when it gets to close. The touch sensor is built in the front bumper stops the Roomba when it hits walls/objects. From all this information, the Roomba makes a mapping of the room for later use. The mapping helps the Roomba save cleaning time by 20%. [54]

2.8.2 Spot

Spot is a industrial robot that is made to inspect routes in terrain that is unsafe for manual work. Here the robot can make mapping with different tools, such as infrared and heat cameras. Also if the work needs to handle transport of equipment the robot can carry up to 14 kg.
[82]

2.8.3 Serve Robotics

Serving Robotics, a spin-off of Uber eats that builds sidewalk delivery robots, is moving ahead with its next generation of robots that can make some commercial deliveries without a human being involved. Therefore in certain domains of operational design, or geofenced areas, Serve won’t rely on remote operators to teleassist robots or following robots to ensure safety.
[65]

2.8.4 FORD autonomous vehicles

Technology installed on the cars are advanced LIDAR higher resolution cameras. One of the LIDAR sensors that are being used is a higher resolution, with a 125-beam sensing to provide a 360-degree field of view. Fitted next to is near-field cameras and short-range LIDAR, that have objectives to look behind and on the side of the vehicle. When all are combined it helps for improving detection of fixed and moving objects. For now the focus lies with delivering goods and a cooperation with Wallmart and Argo AI.
[46] [45]

2.8.4.1 Argo AI

The autonomy platform Argo is trying to make a platform where the goal is design more freedom to the people to shape their own journey with out anyone behind the wheel. By creating an AI that will drive a car by itself and with no human contact, so it can become a goods delivering system.

[68]

2.8.4.2 Lane assists

Lane-keeping assist systems(LKAS) mainly are using cameras as sensors to know where the vehicle are based on the marks painted in the road. LKAS might stop working if it is are snow or dirt covering the lanes. This is something called partial driving automation.It started as a feature in 2000 on the Mercedes Acrtos trucks to warn drivers if they where drifting from their lane. Some of the manufactures that are using this on there vehicles today are Nissan, Toyota, Honda, Mercedes, the Volkswagen Group and Ford. [47]

Chapter 3

Analysis

This chapter give an walk thru of the project described, chapter will with some highlights from the testing done with mapping and safety stops.

3.1 Project

This section will give an detailed description of the project and what the group will develop. Next to the detailed description of the project, the group will also address whether or not it is necessary or if it is advantageous for the group to use surveys or focus groups.

3.1.1 Description

The group wish to establish how to make a map using a 1/10 scale vehicle with a LIDAR sensor attached in the front, and then research how to make the vehicle drive by it self without input from the user. When the vehicle is driving by it self the group wants to have safety features that will make the vehicle stop when it is hindrances in the way that it can avoid or stop, to make sure it does not crash while it drives. The goal is to develop a machine learning algorithm that will make the vehicle drive by it self, if this does not work the group wish to use a path planning algorithm to make the vehicle drive on the area that the group have made a 2D map from.
If we have time to develop the part 2 of the project is make a 3D map of an area and look into how to make the car drive faster in a mapped area.
The group also finds it not necessary or advantageous for the group to neither use surveys or focus group to help with the project, as it would take too much time to get into our project and we would need to ask specific group of people with the right amount of knowledge.

3.2 Testing, results and (evaluation)

This section will cover our findings when it comes to testing and results. What we could have done differently to achieve better results from the testing. Evaluate the safety stop and how the path planning would have worked in theory.

We took multiple tests on a large area, and tried mapping with different speed and different situation. One of our first test we tried to map with higher speed than normal mapping speed since it was a large area with a lot of open space. But as in figure

3.1 the map was fine at the start but later when it got more area to scan it would drift apart and misplace the map location and look nothing like figure 3.2, that is a complete map.

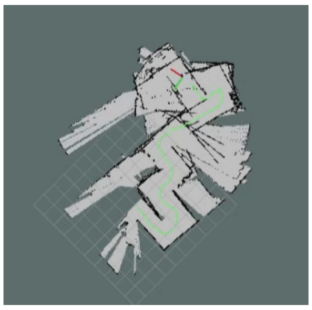


Figure 3.1: When mapping with too much speed

This time we tried to map this area with a normal speed, and it went much smoother as shown in figure 3.2. The mapping process is slow but it would produce a better result than mapping to fast, since the map render for creating the map is not to fast.

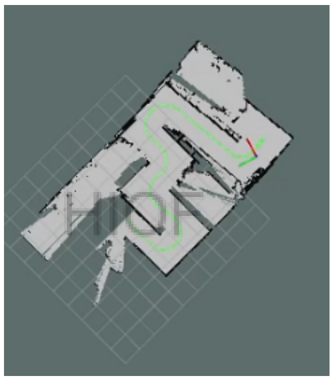


Figure 3.2: Mapping complete

In figure 3.3 is right after the mapping that was done in figure 3.2, we tried to turn the car around and map on the way back. But when it got to a area that was to large for it, it will misplace it's location and start mapping in a another direction.

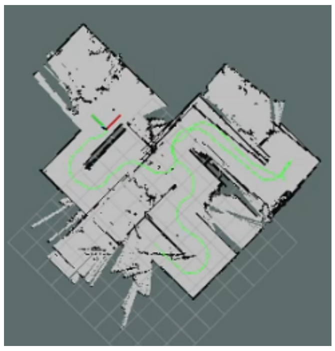


Figure 3.3: Mapping after complete, mapping on the way back

The last test we took was the safety_control.py[1] self driving with mapping. With this running it can only drive straight forward and if something gets too close to the sensor it will stop driving and start the emergency brake. So when we tested it and it would only drive a bit ahead and stop close to the wall as shown in figure 3.4. When it got to close to the wall it would start the safe_brake as shown terminal at figure 3.5.

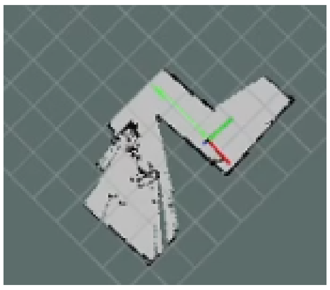
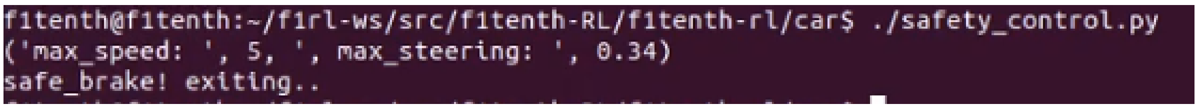


Figure 3.4: Mapping after driving with safety_stop.py



```
f1tenth@f1tenth:~/f1rl-ws/src/f1tenth-RL/f1tenth-rl/car$ ./safety_control.py
('max_speed: ', 5, ', max_steering: ', 0.34)
safe_brake! exiting..
```

Figure 3.5: Terminal after using safety_control.py

For more in-depth description about mapping, mapping with speed and safety controller see ”How to make a map” at 6.2.1, mapping with speed in ”Limitation” at 6.2.2 and safety_controller in ”safety stops” at 6.4.

[55]

[1]`https://github.com/MichaelBosello/f1tenth-RL/blob/master/f1tenth-rl/car/safety_control.py`

Chapter 4

Planning

4.1 Initial plan

The initial plan for the project that was given by our employer, was first to make a map while using the LIDAR sensor attached in front of the vehicle. Then, the group wanted to develop a ML algorithm to make the vehicle drive by itself on the map designed by the group. Initially, the group wanted to use the concept of picture analysis in order to work with the plan and the map.

4.2 Revised Plan

The initial plan had to be revised due to disparities in knowledge, technical difficulties with the car and a lack of time. Since the group has differences in the field of ML and the concept of picture analysis, the group decided to change the main objectives because it would have taken to much time to complete the main objective with ML and the concept pf picture analysis, the group realized that, with not so much experience in the field of ML and the concept of picture analysis the group would have used to much time to learn a new skill set. In an effort to finish the project, the group decided to change the objectives to letting the car drive by itself through a mapped area or track, along with a path planning algorithm. Achieving this the group had to learn how to use the ROS packages and Hector SLAM: obtaining a satisfactory level of understanding took a certain amount of time.

4.3 Algorithm selection and path planning

Algorithms are mathematical formulas coded in order to solve tasks for example in path planning, sorting, and numerous other tasks. The algorithms attempt to solve their problems in an efficient manner, depending on how they are constructed, the speed of the process may vary. If a large number of tasks need to be completed, the task speed can increase exponentially. The majority of algorithms, at least in sorting, have a worst, average and best case senior formula so it can calculate how long it will take to solve the problem.

4.3.1 Selecting an algorithm.

To select an algorithm, we first need to do research about which path planning algorithms exist, and their use cases.

4.3.2 Initial plan

Our original plan was to use path planning to solve our objective to make the vehicle drive on the map we have made. Here we will look into which path planning algorithm that will be the easiest to be implemented and the one that have the fastest way to go to there goal.

4.3.3 Revised plan

We decided to revise our plan due to several issues related to making a map, where the vehicle did not show the right position and kept mapping over whats all ready been mapped. The goal shifted from implementing a path planning algorithm to set the foundation on implementing a path planning algorithm. Here we will make the necessary research on how to implement and which algorithm that could be a good starting point.

Chapter 5

First time setup

In this chapter First time setup it will be written on how to install Ubuntu on a host and pit computer, and how to install ROS melodic and its necessary packages to start mapping with Hector Slam with visuals in Rviz.

5.1 Installation

When the car was delivered it was almost fully built without an operating system. We followed the guide at f1tenth.org/build which contains instructions on how to install almost everything needed for F1tenth system with LIDAR UTM-30LX and Nvidia Jetson Xavier NX.

5.1.1 Install Ubuntu on host and pit computer

First we had to install Ubuntu on a pit computer. This computer is where we are going to control the f1tenth car wirelessly. We had to download the OS image from Ubuntu.com, and burn that image over to a flash drive that is at least has a size of 8GB. Next was to install the OS, this step took some time, depending on what computer it is installed on.
The next step is to install Ubuntu on the f1tenth car. Now it is needed to plug in the flash drive with Ubuntu installed, when powering on the Nvidia Jetson computer it should automatically start. Finally it is possible to follow the guide on screen to complete the install of Ubuntu 18.04 on the host computer.

5.1.2 Installation of ROS Melodic

Since this system is using Ubuntu 18.04 it needs ROS1 and Melodic on it. Following the guide at F1tenth.org - Build - Configure F1TENTH System 2.6 when installing ROS on this host system.
Everything from now on will be mostly be accomplished through terminal/console in Linux.
First we need to make sure it will accept software from packages.ros.org.

```
sudo sh -c ’echo ”deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main” > /etc/apt/sources.list.d/ros-latest.list’
```

Add a new apt key:

```
sudo apt-key adv –keyserver ’hkp://keyserver.ubuntu.com:80’
```

–recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

Updating the Debian packages index:
sudo apt update

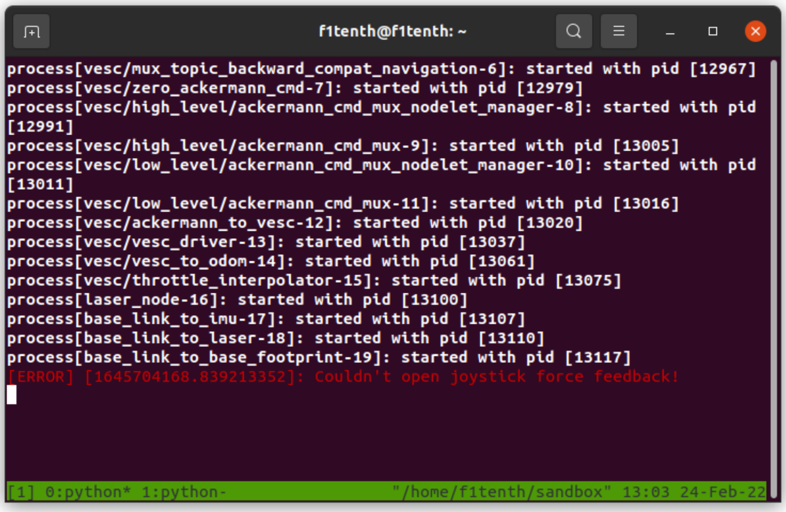
Next is the installation of ROS Desktop package:
sudo apt install ros-melodic-desktop-full

5.1.2.1 Logitech F710 controller driver

Now ROS is installed, the installation of the driver for the controller Logitech F710 can begin. First step is to clone their files from github.
git clone https://github.com/jetsonhacks/logitech-f710-module
Next is to go into the cloned folder and start ./install-module script.
cd logitech-f710-module
./install-module.sh

5.1.3 Installation of the F1tenth - System

First it is needed is to create a folder which will be the workspace folder, inside that workspace folder create a another folder named src. After that click into the src folder and clone the Github repository for F1tenth - system and make sure to use branch melodic from *https://github.com/f1tenth/f1tenth_system*. When this is done, go back to the workspace and in the terminal write *catkin_make* to compile and build the f1tenth system. Next is to source to the folder by typing *source devel/setup.bash* in the terminal.
To make it run make sure roscore is running in a terminal, and to make this run with manual control, with controller type in a another terminal *roslaunch racecar teleop.launch*. Now it should be possible to drive the f1tenth car with a controller.



```
process[vesc/mux_topic_backward_compat_navigation-6]: started with pid [12967]
process[vesc/zero_ackermann_cmd-7]: started with pid [12979]
process[vesc/high_level/ackermann_cmd_mux_nodelet_manager-8]: started with pid
[12991]
process[vesc/high_level/ackermann_cmd_mux-9]: started with pid [13005]
process[vesc/low_level/ackermann_cmd_mux_nodelet_manager-10]: started with pid
[13011]
process[vesc/low_level/ackermann_cmd_mux-11]: started with pid [13016]
process[vesc/ackermann_to_vesc-12]: started with pid [13020]
process[vesc/vesc_driver-13]: started with pid [13037]
process[vesc/vesc_to_odom-14]: started with pid [13061]
process[vesc/throttle_interpolator-15]: started with pid [13075]
process[laser_node-16]: started with pid [13100]
process[base_link_to_imu-17]: started with pid [13107]
process[base_link_to_laser-18]: started with pid [13110]
process[base_link_to_base_footprint-19]: started with pid [13117]
[ERROR] [1645704168.839213352]: Couldn't open joystick force feedback!

[1] 0:python* 1:python-                 "/home/f1tenth/sandbox" 13:03 24-Feb-22
```

Figure 5.1: If it says ”Couldn’t open joystick force feedback” it means all is ready to go.

5.1.4 Installation of Hector_Slam

For the mapping process to start working, it first needs to be installed on the car. It need to clone this repository to get the files for hector slam. One must be sure when cloning to use the branch *melodic-devel* in *https://github.com/tu-darmstadt-ros-pkg/hector_slam.*
The second step is to copy the cloned folder to the workspace/src folder and paste it in there. The next step is to get back to the workspace and in a terminal type *catkin_make* and when it is done, type in terminal *source devel/setup.bash* to source the terminal. The final step is to run Hector Slam, first make sure roscore is running by typing *roscore* in a terminal. In the terminal that is already sourced from before type *roslaunch hector_slam_launch tutorial.launch*
Now it should open rviz and start mapping when all is ready to go.

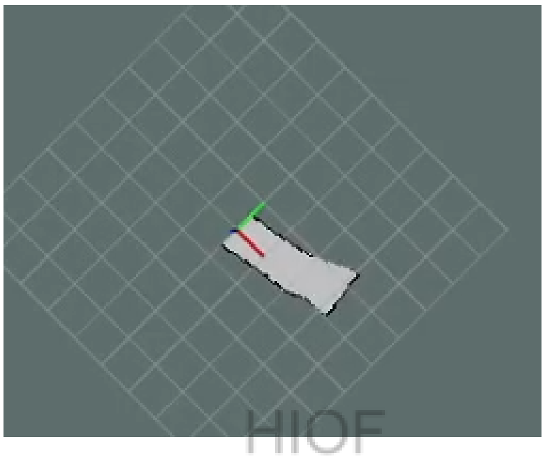


Figure 5.2: Initial scan with Hector_Slam in Rviz

5.2 Problems encountered

There were encountered some installing problems, because of a misunderstandings with the documentation or the lack of documentation in ROS and melodic packages. We have found solutions on how to fix these problems, and it will be stated how.

5.2.0.1 SSH

The first problem we encountered was during the setup phase of the car, revolving around using SSH to connect to the car. This was mostly due to the provided documentation referring to the car system as ”f1tenth” where our system was called ”car”.

5.2.0.2 VESC

By using the documentation, we believed that we had to get the VESC program to work on the car, this seems more like a misunderstanding from our part as this was encountered during us still struggling with using SSH to connect to the car.

5.2.0.3 Catkin_make

Catkin_make is used to create a workspace for the car and allow it to be controlled both manually and autonomously, as well as allowing use of the LIDAR sensor. This failed to create a workspace due to the documentation downloading the most recent version of ROS 1, which in this case was melodic for us. Fixed by specifying which version of ROS 1 to use.

5.2.0.4 LIDAR

The system was not recognizing the LIDAR sensor, this was again encountered due to us progressing without getting the previous step (catkin make) to work. The sensor was recognized once we solved the issue related to catkin make, as well as connecting the the power cables of the sensor to the power supply.

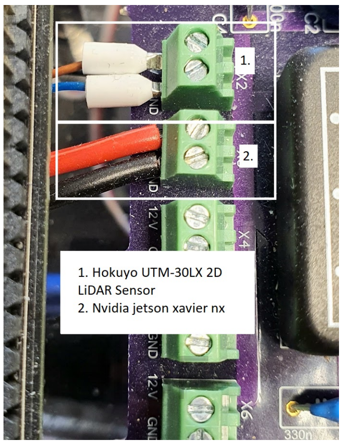

1.
2.
1. Hokuyo UTM-30LX 2D LiDAR Sensor
2. Nvidia jetson xavier nx
X2
X4
X6
12V
GND

Figure 5.3: Remember to connect the tiny brown and blue cable

5.2.0.5 Inability to turn left/right

The inability to turn the car left or right, was again due to the documentation referring to an older version of VESC, as the documentation had VESC mk4 and our car has VESC mk5. In the following document however there was a note clarifying that if the user was using VESC mk5 it would be necessary to follow this guide at *https://github.com/f1tenth/vesc_firmware* instead.

5.2.0.6 Mapping in only start position

While mapping the car would register as not moving in Rviz software, so the mapping process would still take the data from the LIDAR sensor but put all data in start position. This would result in putting all the data on top of each other and creating a splash map that is not recognizable. This was fixed with correcting the file mapping_default.launch in the hector_mapping/launch folder. In this file two arguments for base_frame and odom_frame was changed into base_link. We also had to change a node on the bottom of the same file, had to create a new node that was adjusted after changes with base_link.

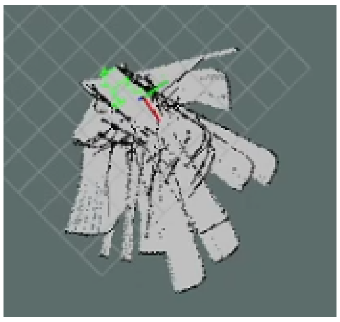


Figure 5.4: Overlapping from Rviz

5.2.0.7 Mapping while moving

The car would not properly execute the mapping process and instead create entirely different paths that should not even be possible. After multiple attempts we reduced the speed of the car to the minimum and while the car did not like the experience, we were able to successfully map the course we were testing in.

Chapter 6

Methodology

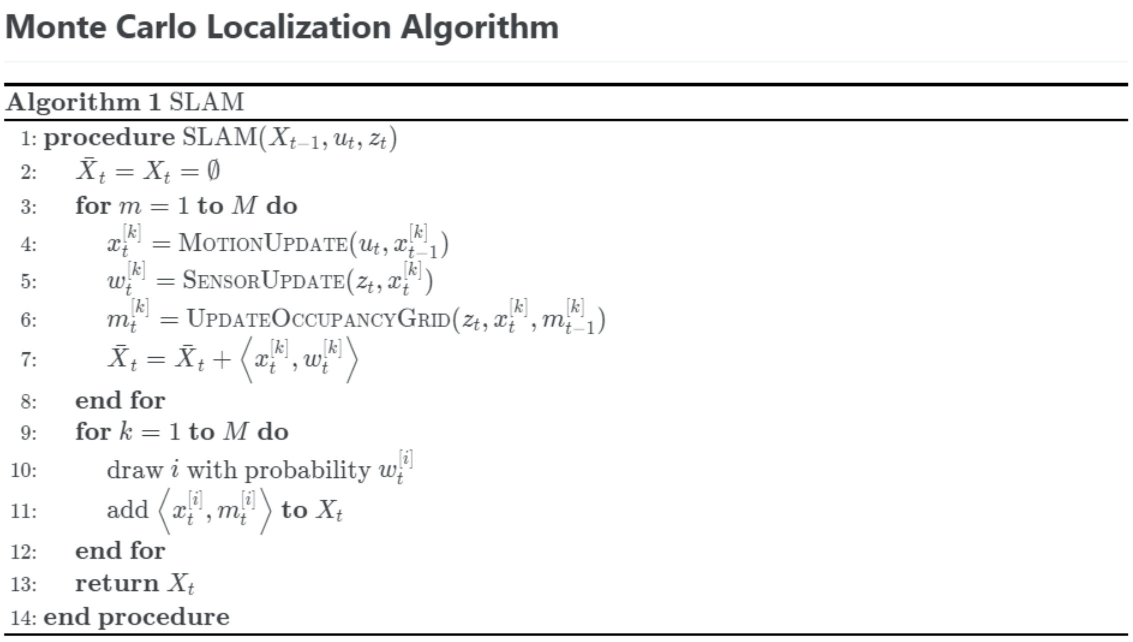
We will in this chapter present a description in more detail the methodology behind whats to be implemented and how the algorithms we have chosen work. The chapter will also contain details about Monte Carlo Localization algorithm, ray casting, A*, D*, Dijkstra's algorithm and LIDAR. We will end the chapter with implementation where we will be writing a bit about localization, safety stops and self driving with path planning.

6.1 Description of methods

Here we will describe the methods that are being used in the project in a deeper manner, but also why these methods are being used here and the thought process behind them. All the described methods will in be implemented or written how in theory we want to use it, or certain aspect how it could work for our benefit later in chapter implementation.

6.1.1 Monte Carlo Localization algorithm

The Monte Carlo Localization algorithm(MLC), is when the algorithm are given a map, MLC is used to estimate the position and orientation of the robot in the map using a particle filter. MLC is estimating three values, X, Y and the angular orientation theta. As the figure 6.2 shows.



Monte Carlo Localization Algorithm

Algorithm 1 SLAM

```
1: procedure SLAM($X_{t-1}, u_t, z_t$)
2:     $\bar{X}_t = X_t = \emptyset$
3:     for $m = 1$ to $M$ do
4:         $x_t^{[k]} = \text{MotionUpdate}(u_t, x_{t-1}^{[k]})$
5:         $w_t^{[k]} = \text{SensorUpdate}(z_t, x_t^{[k]})$
6:         $m_t^{[k]} = \text{UpdateOccupancyGrid}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$
7:         $\bar{X}_t = \bar{X}_t + \left\langle x_t^{[k]}, w_t^{[k]} \right\rangle$
8:     end for
9:     for $k = 1$ to $M$ do
10:        draw $i$ with probability $w_t^{[i]}$
11:        add $\left\langle x_t^{[i]}, m_t^{[i]} \right\rangle$ to $X_t$
12:    end for
13:    return $X_t$
14: end procedure
```

Figure 6.1: figure is taken from https://fjp.at/posts/localization/mcl/

Sudo code from figure shows an example to set up the the MLC algorithm.

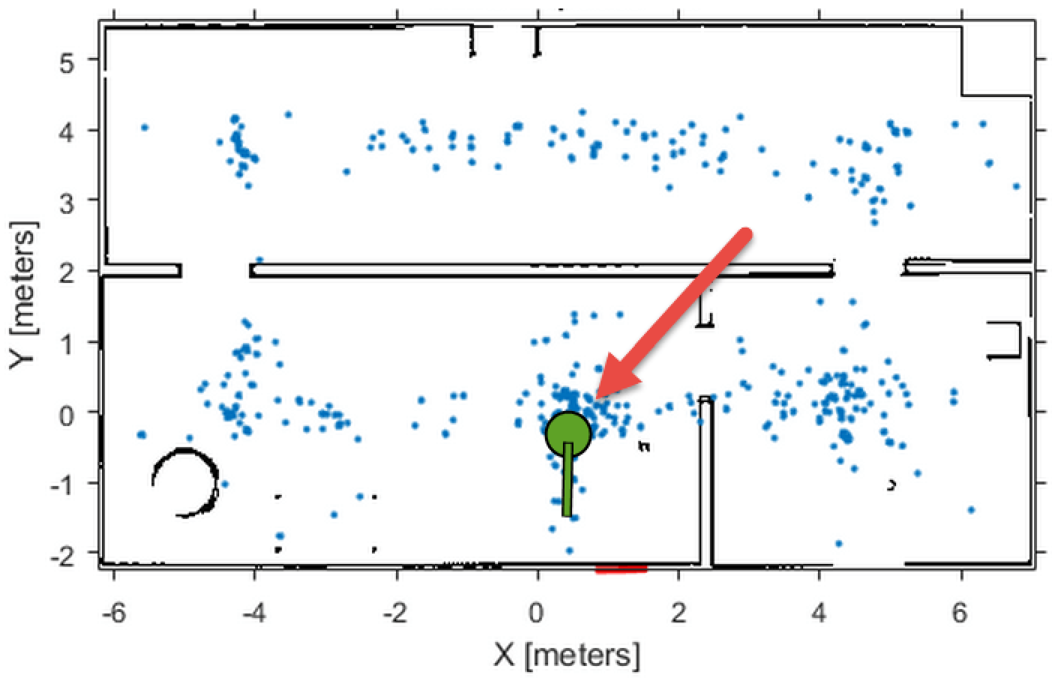


Figure 6.2: Picture is from https://se.mathworks.com/help/nav/ug/monte-carlo-localization-algorithm.html

In this project Monte Carlo Localization Algorithm(MLC) is currently used within a particle filter to check where the f1tenth car is on a map. With that working it should be easier for a self-driving car to understand where it is and find a fast path to the destination. How particle filter with MLC is used is written more about in the Localization chapter at 6.2.

[95]

6.1.2 Ray casting

[77] [52] [24]

Ray casting is a basic form of ray tracing, Scott Roth introduced the term "Ray casting" while he was working at General Motors Research Lab between 1978 to 1980. Ray casting is for 2D and 3D where it casts rays along where the viewpoint is seeing, and it determines if it is just empty space or an object or wall. If ray casting is used with 3D it can "see" color of the object and how much brightness it is, those values are important for the final image after rendering what ray casting has seen.

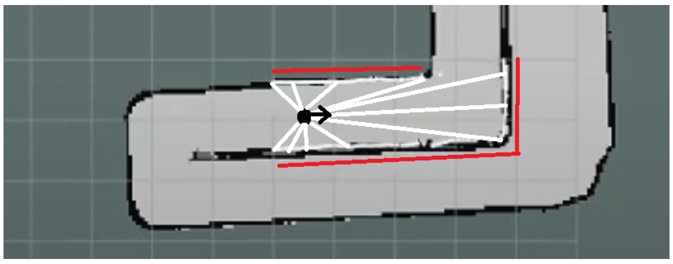


Figure 6.3: Initial scan with Ray casting

Ray casting is used like shown in figure 6.3 in this project. This sensor is using its laser to send ray casting in a 270 degree angle, and receive feedback from the sensor. It is used in the mapping process and the particle filter, since it is reading laser data and converting it to a map or map location. [24] [52] [77]

6.1.3 A*

A* is a heuristic search function that intends to improve upon the Dijkstra algorithm. The main goal for the algorithm is to find the shortest path from node A (start node) to B (end / goal node). Before a search is initialized we must first decide on how far apart all the nodes are by setting appropriate cost (distance) values. For all nodes horizontal and vertical we could say that the cost is 1, and the diagonal cost would then be sqrt(2) or approximately 1.4 (diagonal length of a square). To make it easier by only working with clean integers we can multiply these values by 10, so the horizontal/vertical and diagonal node costs are 10 and 14 instead of 1 and 1.4, respectively, as shown in the figures below. As mentioned briefly we appointed certain cost values for nodes depending on their distance from the starting node. When we begin a search, we start by evaluating all the nearby nodes from our starting node. The distance from the current node to the starting node is known as the G cost (upper left corner) and initially will be either 10 (horizontal/vertical) or 14 (diagonal). We also have an H cost (heuristic) which is the distance from the current node to the end/goal node (upper right corner). This means that the closer we get to our goal, the lower the H cost and the higher the G cost. Lastly, there is also the F cost which is simply the total cost, in other words G cost + H cost. The F cost is the primary value that is used when calculating the shortest path. Quite simply, the A* algorithm will always try to go for the node with the lowest F cost.

In total, we have three different values per node:

G cost = distance to the starting node

H cost = distance to the end node

F cost = G cost + H cost

No obstacles

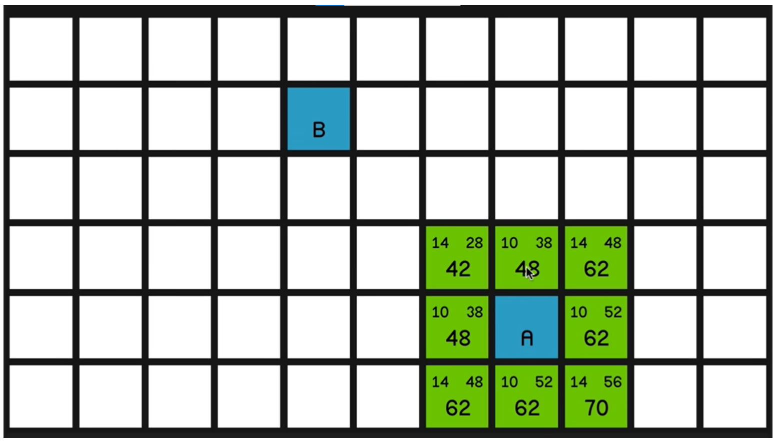
We begin the search at our starting node (A), and start by evaluating the nearby nodes. The program will then look for the node with the lowest F cost first, and mark that node with the CLOSED state. The program will then evaluate nearby nodes from this closed node the same way as before and simply follow a new node with the lowest F cost. This will continue until the goal node (B) is reached.

Obstacles

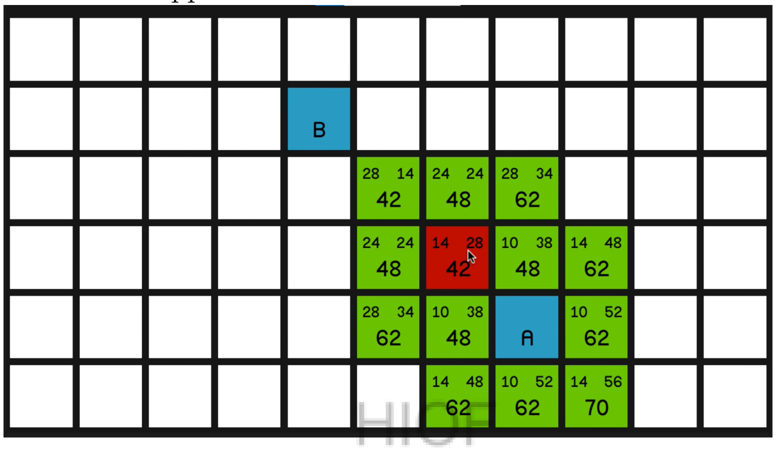
If known obstacles are present, some areas will work a little bit differently. The beginning is the same where we evaluate the nearby nodes surrounding our starting node. As per usual the program will attempt to follow the nodes with the lowest F cost until our goal is reached, but there are a few differences in what will happen.
Since we cannot pass or go through obstacles we also cannot evaluate any nodes that are mapped as obstacles since they are not eligible for travel. This can cause a lot of backtracking for the algorithm as we now have to try and move either sideways or backwards to circumvent the obstacle nodes. Ultimately this can also lead to a lot of individual nodes being evaluated before we actually reach or goal.

Exceptions

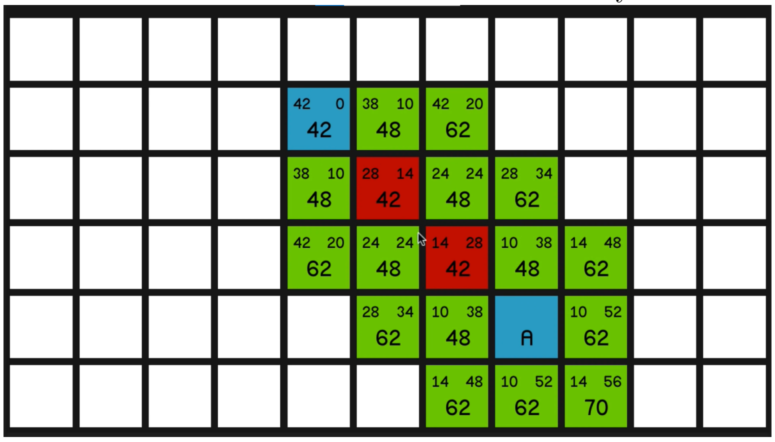
In the event that two nodes share the same F cost, the program will then evaluate the H cost and therefore pick the node that is closest to the end node. If there are nodes that happen to have identical cost values the program can simply pick between these identical nodes at random. The figure 6.4 below show a standard progression path for the A* algorithm where there are no obstacles on the map.



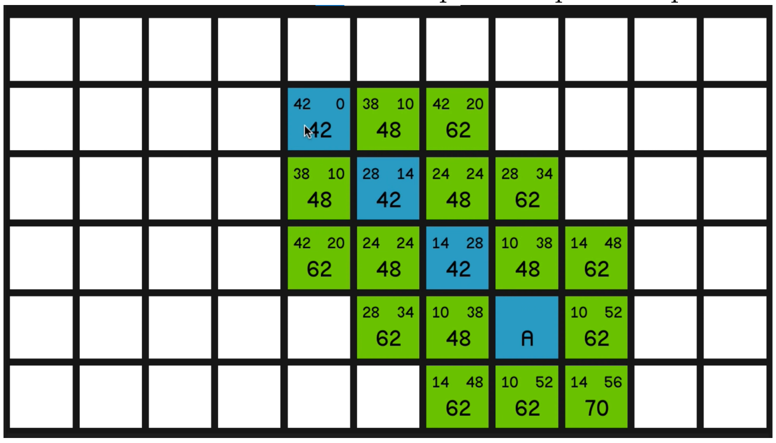
Upper left node has the lowest F cost



Mark it as CLOSED and evaluate its nearby nodes



Look for the lowest F cost and repeat the previous process



Finally reach our end goal, node B

Figure 6.4: Source: https://www.youtube.com/watch?v=-L-WgKMFuhE

[4] [56]

6.1.4 D*

The name D* comes from the term Dynamic A*, and is an algorithm that works almost just like A*. Essentially it is the product of taking a heuristic search algorithm like A* and adding incremental search functionality. This added capability of incremental searching can speed up repeated searches by using experience from previous search problems. Generally this leads to better performance over A* since the program can keep information and use it to replan routes. Other differences between D* and A* is that the edge weight (cost) between nodes can change during run time. D* also works backwards, by moving from the end node to the start node instead and then follow a back trace to the end goal once the start node is reached. Aside from these differences, D* follows the same expansion pattern like A* by evaluating nearby nodes, moving to the lowest F-cost node and then evaluate nearby nodes again.

[18] [9]

6.1.5 Dijkstra's Algorithm

Dijkstra's algorithm is one of the earlier forms of path finding algorithms, with the goal of finding the shortest path from point A to point B. It does this by scanning for nodes and checking the distance between every node, this leads to an end result where every possible path that leads from A to B has been checked for distance, and based on numerical values defines the shortest path. Each node is assigned a default value of infinity, hence why the algorithm checks every possible node rather than finding the most immediate path. This also leads to constant addition of every node on every potential path, the lowest of those end values being deemed the shortest path.

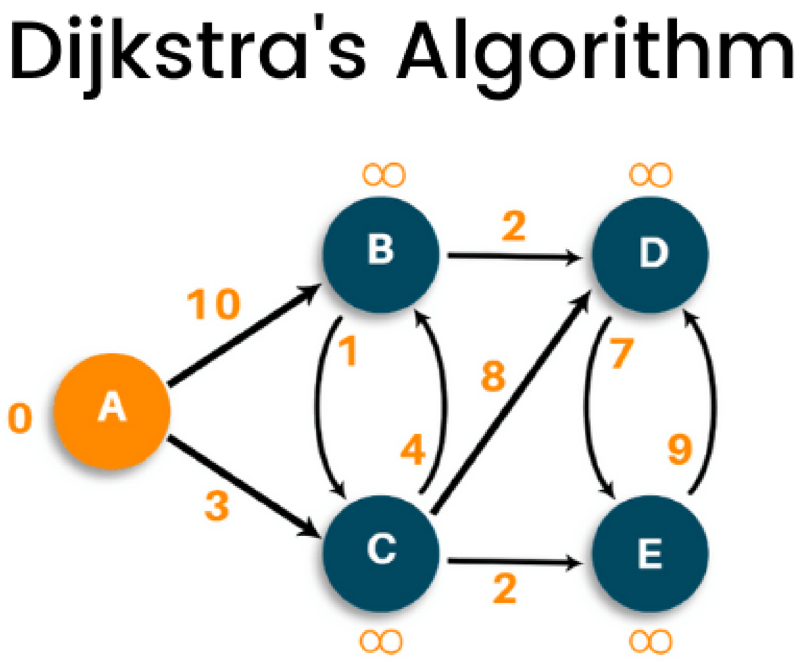
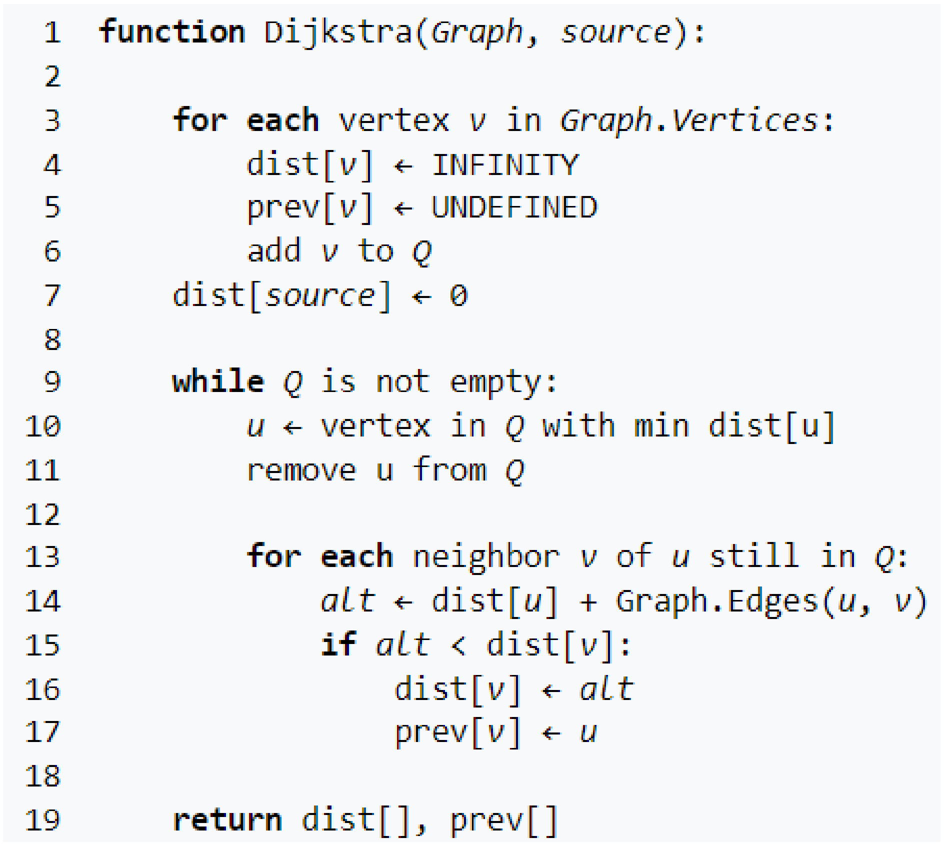


Figure 6.5: Source: https://favtutor.com/blogs/dijkstras-algorithm-cpp

The figure is a demonstration of how the algorithm calculates the shortest path, assigning each node a new value determined by the nodes previous to the unchecked ones and applying that value to the additional distance. This is also the reason it keeps checking most nodes on its way of finding the shortest path, only ever eliminating a path if paths meet up at a shared node, determining from there which path is the shortest up to that point.



```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Figure 6.6: Source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Pseudocode here is just an example to show and explain how one would potentially set up the algorithm and how to use it effectively

[101] [72]

6.1.6 LIDAR

LIDAR is a tool for checking the range from the sensor to a object, it will measure by emitting light signals to an object and calculate the distance based on when the signal returns. This project is using Hokuyo UTM-30LX sensor to get the LiDAR data. This sensor is using the 2D type of scanner to determine the length to a object or wall. It also has a scanning angle on 270 degrees, so it will receive data from that.

[76]

6.2 Implementation

In this section we will go over what's been implemented on the vehicle, our findings when it comes to mapping and safety stops. Last part of the implementation will go over in theory how path planning would have worked on the vehicle if we would have managed to reach our goals. Our safety implementation will be included in the list of code and the start of our code with path planning using A*.

6.2.1 How to make a map

There are a few different ways to go about making a 2D map. Here, for instance some of these methods are SLAM or hector SLAM for making a 2D map. In the case of this project, the method with hector LIDAR looks to be a better option and easier, because hector SLAM has no need for odometry to make a map. It also uses low computation resources. Hector SLAM is launched as a ROS node where it publishes current pose data and a map. [37]

A few guides have been made on how to start the process with mapping in either SLAM or with hector SLAM, in this project we are using the git repository from f1tenth_gtc_tutorial that is made by Varundev Sukhil and Madhur Behl. First, we needed to start by downloading it and start catkin_make for compiling it and check if every thing is correct. When it is finished compiling, we had to write in the terminal the ROS command for it. After this, it could start and as well as Rviz, and automatically start mapping if everything is as it should be. This is written more about in chapter 4.1.4 Installation of Hector_Slam.

6.2.2 Limitations

There are some limitations when using hector SLAM and making a map while the F1/10 vehicle is driving in relative high speed for the LIDAR sensor. Sudden movements and turning often break the map and the program loses its position on the map. The vehicle seems to drive off to the distance, or keep mapping on top of what already has been mapped. See figure 6.7 for example.

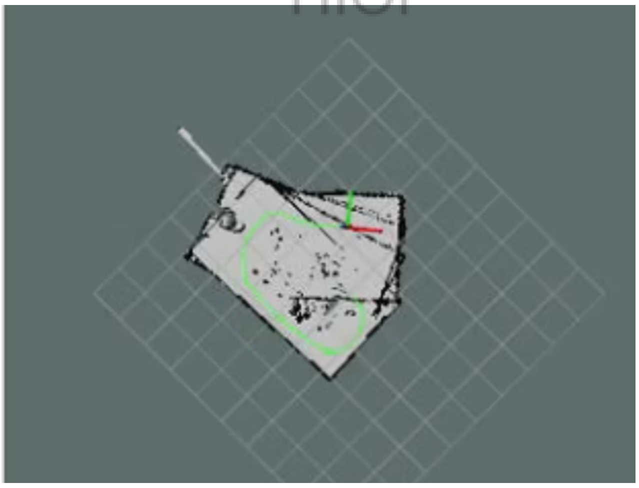


Figure 6.7: Not accurate when mapping in higher speed

6.2.3 Working map

When the vehicle is driving relatively slow in corners and when the vehicle is turning, hector SLAM will complete the 2D mapping of the area the vehicle is being driven. See the figure 6.8 for an example of a complete map. The longer time the LIDAR uses the more accurate the mapping will be, so even if the car is struggling from the low speed it is driving, the map will have more details from its surroundings. Figure 6.8 shows the group working with a map.

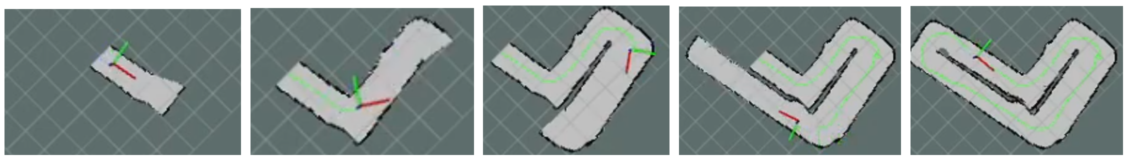


Figure 6.8: Mapping in progress

6.3 Localization

For localization we are going to use f1tenth's method on how to use a particle filter. This method is using the MCL (Monte Carlo algorithm) to localize its whereabouts on a 2D map. First it needs to know where the initial position on the map is, and it will use the LIDAR sensor data to check if the scan is the same as the initial pose. When driving, it will continuously compare with the map and LIDAR data if it's in the right place. When driving with manual control the localization will look like the figure below, it will check for walls and compare if it's in the same location on the map. The sensor is casting rays to get feedback so the sensor can compare it with the data it receives from the map. Figure 6.9 and 6.10 show the position on the map the group implemented.

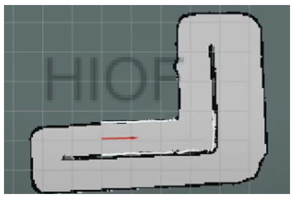


Figure 6.9: Initial pose with particle filter

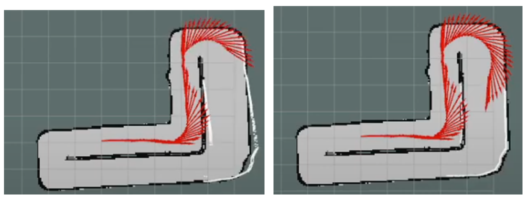


Figure 6.10: Trying to find position (white line out of place) and found position

6.4 Safety stops

The vehicle will try to drive on a map that's already been made. The program will utilize the LIDAR and it's laser scanning to measure distances to walls and objects, where it will take certain safety precautions. If the vehicle comes too close to a wall, the car will turn a different direction or stop, and if there is an object that blocks the way the vehicle will either turn, reverse or stop. From here, it will then try to find a new route with path planning or just simply stop if it believes there is no route to the goal. This is made to ensure that the vehicle is not going to have any accidents with crashing where the worst out come is that our vehicle will be too damaged to drive. As the figure 6.11 shows.

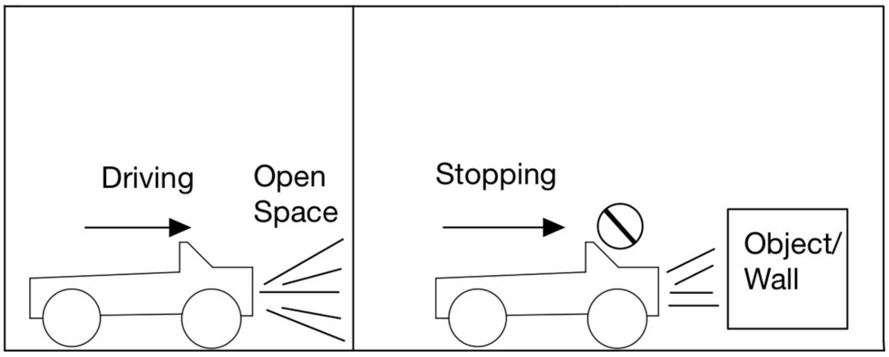


Figure 6.11: Stopping when a object or wall is to close

6.5 Self driving with path planning

The car is going to drive by itself with a path planning algorithm and a map, and it needs a start position and an end position. With that given in Rviz with position arrows it will calculate the shortest path with the algorithm A* and simultaneously avoid crashing into walls that is recorded in a map and distance measured by the LIDAR 2D sensor. It should always calculate the shortest path, as if the area it is scanning suddenly is open and could receive a shorter path than normally. In the figure 6.12 it is possible to see step by step that it is checking the map, therefore it will know where the shortest path will be. When the car gets the inputs on where start and end position is it should try to drive the route that path planning has given to the car.
In Appendix A the group tried implement the path planning with safety stop is listed. This is made to set as foundation if the group wants to develop it at a later point.

6.5.1 Implementing A* and D*

With the research we did from section 6.1.3 and below to 6.1.6. the group wanted to try to implement and test to see which path planning algorithm would have worked best for our use.

6.5.2 Implementing and testing one global planner in simulation

We tried to implement a global planner [1], and adjust it with our f1tenth system. It should help the car to find a optimal path from location A to B, when driving by itself. When the global planner started to run, it looked like figure 6.12 where it send out a lot of green dots to check what was inside the map and what is outside of the map. After the check it need to receive a start position and where the destination should be. In the end the car should be able to drive by itself inside the blue area in last map at figure 6.12.

[1]`https://github.com/lsa-pucrs/global_planner_move_base`

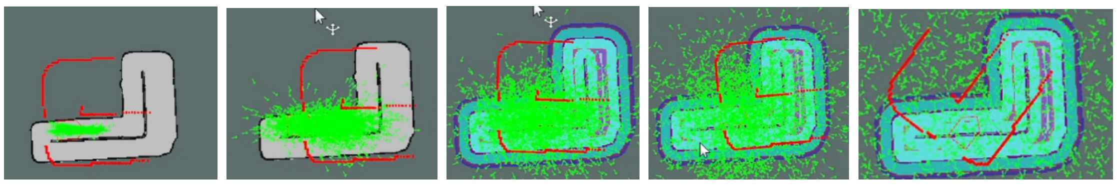


Figure 6.12: Path planning: Checking after shortest path with a existing map

Chapter 7

Discussion

This chapter covers the discussions of the project. Where section 7.1 talks over what the objectives where stated initially for the thesis. Chapter ends with what we plan on working on if further development in section 7.6.

7.1 Goals

During this section, we will discuss what our goals were in the introduction. We discussed how we solved the problems, what we did, and whether we arrived at the goal.

7.2 Choosing a programming language

The Ros software supports C++ (roscpp), Python (rospy), and LISP (roslisp) : those are the three main client libraries provided by ROS. We also have experimentalROS libraries for other languages, but we primarily concentrate on the three main ones. From these three, we were able to find the most documentation and information on roscpp and rospy. The Roslisp language, on the other hand, does not contain as much information, but it is also not as commonly used as the other languages.

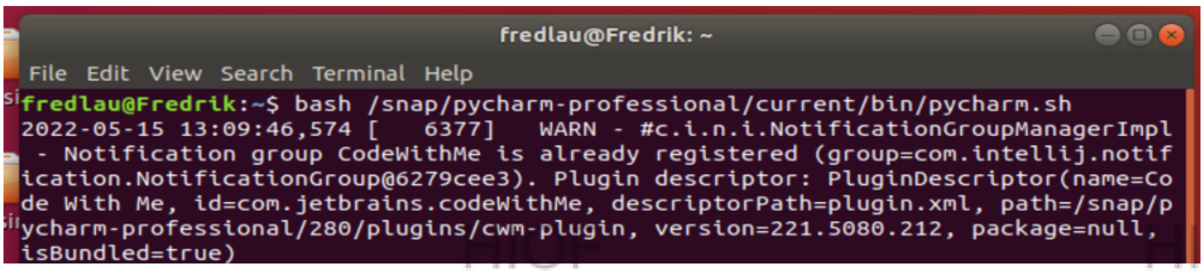
We chose Python for the main reason that two members of the group had prior experience with the language. Furthermore, Python is a more intuitive language to understand for a person who does not know the language. C++ is a highly specialized and complex programming language. Furthermore, none of the group members possessed a lot of prior knowledge regarding C++.

Program Language	**ROS Main LIB**	**ROS EXP LIB**	**Group Knowledge**
Python	Rospy	X	✓
C++	Roscpp	X	X
Java	X	Rosjava	✓

Table 7.1: Caption

7.3 Impression of ROS

In the beginning, learning how to use ROS is a steep learning curve : documentation and error messages can be hard to understand. While the error messages were not always the easiest to understand, some of the documentation from ROS did not always give out helpful answers or sometimes lacking explanations. Issues we did encounter were often issues others had encountered before, but the option where other had solved were not always the most intuitive and sometimes lacked content on how to understand and solve the issue. Where forums often was the best option it also lead to more research and testing other options from all kinds of sources.
To use rospy, and the python 2.7 packaging without using the terminal and Pycharm we encountered a problem or several problems that ROS and text editor never seemed to work together no matter what we did. After looking into countless forums both for ROS and google, one of the solutions that seemed to work was to open a Pycharm through a special file that connected to Rospy and the ROS libary that was installed. This way we could connect to the subscribers and publishers. The command in figure 7.1 shows one way that worked for the group, to access to the Melodic libraries.



```
fredlau@Fredrik:~$ bash /snap/pycharm-professional/current/bin/pycharm.sh
2022-05-15 13:09:46,574 [   6377]   WARN - #c.i.n.i.NotificationGroupManagerImpl
 - Notification group CodeWithMe is already registered (group=com.intellij.notif
ication.NotificationGroup@6279cee3). Plugin descriptor: PluginDescriptor(name=Co
de With Me, id=com.jetbrains.codeWithMe, descriptorPath=plugin.xml, path=/snap/p
ycharm-professional/280/plugins/cwm-plugin, version=221.5080.212, package=null,
isBundled=true)
```

Figure 7.1: One way to have access to the ROS library with Pycharm

ROS may also be easier and a lot better if working with a AMD64 system over an ARM64 system, since an AMD64 system is used a lot more, and therefore it will also have a lot more documentation. Not all packages in ROS is supported by ARM64, so many questions asked on the internet would not be to any help because it is not supported. Even though that ROS has a steep learning curve, and the system can be a bit confusing in the beginning with not always the best documentation. ROS is good for what it has been made for; utilizing smart ways to make it for robotics and its use cases.
[36]

7.4 Mapping

This project needs a map, since it would be difficult to self-drive with f1tenth without a map as F1tenth is not built to drive by itself without one. With a map it will be much faster to localize itself in an area as well. In this project we are using hector slam to make a 2D map because it was one of the methods that was recommended to use with the f1tenth system. Mapping can be useful to a lot other than a f1tenth project, it can be used to help others find the location they are looking for. The same map can be used for other projects that need a map to localize its position, because the map can be converted to other files than only pgm files.

7.4.1 Hector Slam vs Visual Slam

In this project we will use Hector Slam instead of Visual Slam since we are using a 2D LIDAR scanner that Hector Slam is using, instead of the camera which Visual Slam will use. Visual slam is not supported with the use of melodic, and not supported with Ubuntu 18.04 which Nvidia Jetson Nx is using. Visual slam is only supported with Ubuntu 10.04 and that is using ROS fuerte and is also only an experimental research code, so it is not updated to support any of the newer ROS or Ubuntu versions. In the documentation page for vslam, there is a warning that recommends other more supported methods to map an area.

[2] [5]

7.5 Autonomous driving

- How we are going to make it drive autonomously
When we looked into how we were to solve that the car was going to drive itself autonomously, we ended up with two ideas. One was to use machine learning that would make the robot teach itself how to solve the course over many attempts. The other one was the use of path finding algorithms.

- Why we choose the option we did
When we ended up having to make a decision it was based on how easily the group was able to learn the necessary skills. We only had one member of our group that had any prior knowledge to machine learning. With only one person knowing machine learning in the group and the time we had to do the project we didn't see it as a feasible approach to solve the task. Therefore we settled on using path-finding algorithms with a combination of pre-made maps. Everyone has prior knowledge of programming some sort of an algorithm even though its not necessarily as complex.

- What required it to work (as in algorithm and packages)
To make the vehicle be able to drive by itself by the use of path finding algorithms. We have to use an algorithm and a pre-made map. The pre-made map we have mapped out our self by driving the the car at slow speeds in a set track. Then we implement the algorithm A*(a star) so it can calculate the fastest and best way to drive that path.

- Conclusion on why that it was the best option.
Based on the group's lack of machine learning knowledge and research done on potential path finding algorithms, we agreed on using A*(a star) due to it being more advanced than Dijkstra's algorithm but still relatively simple to set up. Another potential algorithm we decided to test out was D*(d star), which has multiple versions of itself and still relatively simple in terms of logic. - This is just pure theoretical

7.5.1 Safety mechanism

Safety mechanisms are protocols or indirect orders for the project in response to reducing the chances of dangerous or destructive mistakes. A mechanism we were able to test in terms of automated driving is having the car stop immediately once it senses its within X amounts of meters of an obstacle. We were only testing it on lower or more controllable speeds but if this was our main way of safety mechanism in higher speed environments, there would need to be more room to allow the car to come to a full stop. Higher speed puts forth the idea that hitting the breaks could lead to drifting and sliding, so the threshold would need to be bigger or add additional instructions depending on the situation. Such as turning left or right depending on the surrounding area. Ultimately it would require more testing and different terrain to work out an average response threshold for forcing the car to stop based on its current speed. Alternatively there's also allowing the car the freedom of turning in an attempt to avoid the obstacle, this will require even more testing so we are able to learn how hard the car can turn at different speeds, as a hard turn at high speeds bring in the chance of the car rolling over.

7.5.2 Algorithms

In order to identify and solve a driving path for the vehicle, an algorithm is necessary. As we examined algorithms, we wished to choose one that was the fastest and most reliable. For this study, we chose to look at four algorithms: Dijkstra, a-star, d-star, and d-star lite. As one of the first pathing algorithms developed, Dijkstra is also the slowest of the bunch. As the successor of Dijkstra, A-Star is an improved version of Dijkstra, D-Star is an approved version of A-Star, and D-Star Lite is an improved version of D-Star. Our conclusion was that the easiest one to implement first would be the a-star because it was easier to understand and if implemented, we could add the two others at a later date.

7.5.2.1 Algorithm comparisons

Generally when searching for path finding algorithms, it is more likely to come across Dijkstra, A*, D* and D* Lite among a few other additions. These algorithms are usually either heuristic search algorithms or incremental search algorithms, but there are some that try to combine both functionalities. This way we essentially end up with three different groups of functionalities than an algorithm may implement: Heuristic search (I.e A*), Incremental search (DynamicsSWSF-FP) and Incremental heuristic search (I.e D* Lite and LPA*).
When deciding on the pathfinding algorithm we would start with there were a few different aspects we had to consider with each algorithm:
Simplicity: is the general concept and behavior for the algorithm easy to understand or grasp?
Implementation: is a typical implementation small in scale, flexible and simple to get started with?
Performance: does the algorithm perform well within our use case of autonomous vehicles?
Documentation: is there sufficient documentation, instructions, guides and info about the algorithm?
Popularity: is the algorithm frequently discussed within the context of pathfinding problems?
These are some of the points we ended up discussing frequently when we started researching the different algorithms. Our first impressions also played a bit of a role in deciding on a suitable algorithm, but these would be less obvious or less tangible points than the ones stated above. Although all of these points played some role in our choice, performance is

still arguably the most important factor.
Dijkstra's has a fairly long history with its origins rooted in mathematics, and is therefore also frequently discussed since it sees usage in other fields as well. As a result it's quite easy to find plenty of documentation and various explanations on how to implement Dijkstra's on various platforms. There are also online resources available like videos and graph tools that make it easier to understand how it works. Unfortunately, performance is an area in which Dijkstra's will struggle. Put simply, an algorithm like A* will only evaluate nodes that look favorable. Dijkstra's on the other hand will generally end up evaluating all connected nodes until the goal is found. The figure 7.2 shows[1] all the traversed nodes (blue) and nodes to be evaluated (green) before the end goal was found (red square).

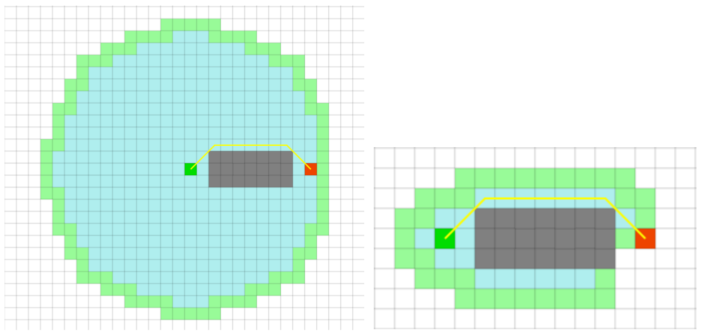
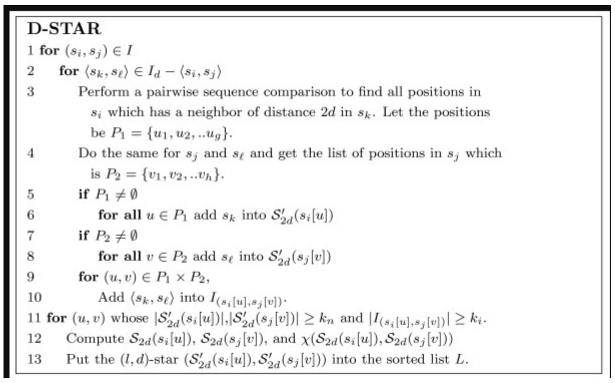


Figure 7.2: Dijkstra's (left) compared to A* (right)

A* is an algorithm that is frequently discussed regarding path finding, and also has a fair amount of documentation available. Additionally, it also tends to boast favorable performance and many implementations are small in scale and easy to understand with few lines of code. To us it seemed to score quite well in most aspects, which is why A* was the algorithm we initially wanted to get started with.
Once we had successfully implemented A* and done some testing with the vehicle, our next goal would have been to move on to either D* or D* Lite. We thought it would make for a smoother transition since these two have a lot less surrounding discussion and available information online about them. D* is also in fact based on A*, but with incremental search functionality added. All considered it seemed like getting some experience with A* first would be useful before attempting either D* or D* Lite. The advantage of incremental searching is that it allows the vehicle to learn from previous experiences, and therefore speed up future searches that are similar to previous ones. This is not something that A* does which means that D* and D* Lite will outperform A* where a lot of repeated searches occur, such as obstacles frequently appearing.
D* Lite behaves similarly to D*, but with a few changes to the heuristic search that is supposed to make it more efficient. Performance wise it matches D* and sometimes even beats it, and there are also more guides and resources available on D* Lite than the original D*. The original D* might in fact be considered to be a little outdated since the creator himself, Stentz, is using D* Lite more on recent projects. Even though the original D* has been around longer than D* Lite it is surprisingly difficult to find guides or other information about it. To top it of, D* Lite is also meant to be easier and simpler to understand and implement than D*. With all of this in mind it might be more productive

[1] https://qiao.github.io/PathFinding.js/visual/

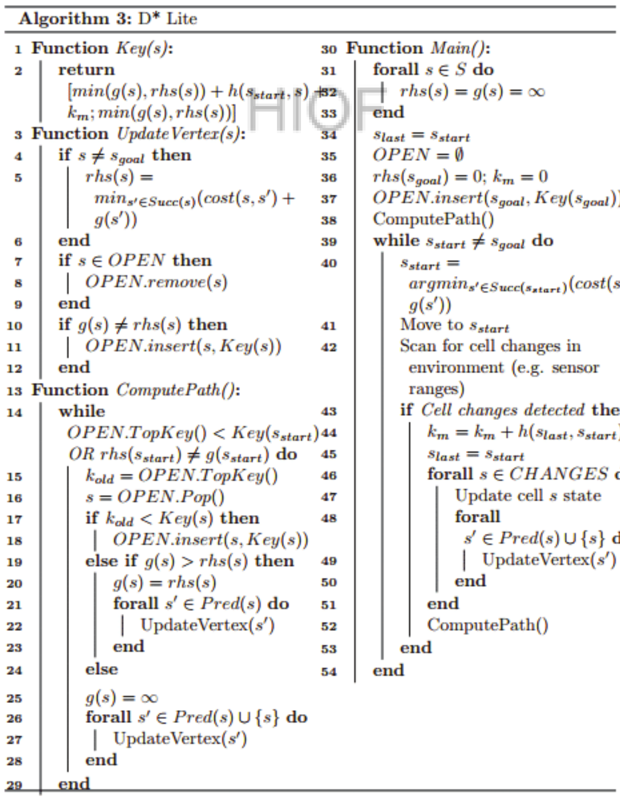
to skip the original D* altogether, and simply focus on D* Lite after A*.
Figure 7.3 and figure 7.4 below are two pseudo-code examples of the original D* and the D* Lite algorithms. Although this D* example has fewer lines of pseudo code, an actual D* implementation would still generally require more lines of code compared to D* Lite.



```
D-STAR
1 for $(s_i, s_j) \in I$
2     for $\langle s_k, s_\ell \rangle \in I_d - \langle s_i, s_j \rangle$
3         Perform a pairwise sequence comparison to find all positions in
            $s_i$ which has a neighbor of distance $2d$ in $s_k$. Let the positions
            be $P_1 = \{u_1, u_2, ..u_g\}$.
4         Do the same for $s_j$ and $s_\ell$ and get the list of positions in $s_j$ which
            is $P_2 = \{v_1, v_2, ..v_h\}$.
5         if $P_1 \neq \emptyset$
6             for all $u \in P_1$ add $s_k$ into $\mathcal{S}'_{2d}(s_i[u])$
7         if $P_2 \neq \emptyset$
8             for all $v \in P_2$ add $s_\ell$ into $\mathcal{S}'_{2d}(s_j[v])$
9         for $(u, v) \in P_1 \times P_2$,
10            Add $\langle s_k, s_\ell \rangle$ into $I_{(s_i[u], s_j[v])}$.
11 for $(u, v)$ whose $|\mathcal{S}'_{2d}(s_i[u])|, |\mathcal{S}'_{2d}(s_j[v])| \geq k_n$ and $|I_{(s_i[u], s_j[v])}| \geq k_i$.
12     Compute $\mathcal{S}_{2d}(s_i[u])$, $\mathcal{S}_{2d}(s_j[v])$, and $\chi(\mathcal{S}_{2d}(s_i[u]), \mathcal{S}_{2d}(s_j[v]))$
13     Put the $(l, d)$-star $(\mathcal{S}'_{2d}(s_i[u]), \mathcal{S}'_{2d}(s_j[v]))$ into the sorted list $L$.
```

Figure 7.3: Source: https://www.researchgate.net/figure/The-D-STAR-algorithm_fig1_6689399

Pseudocode of D*

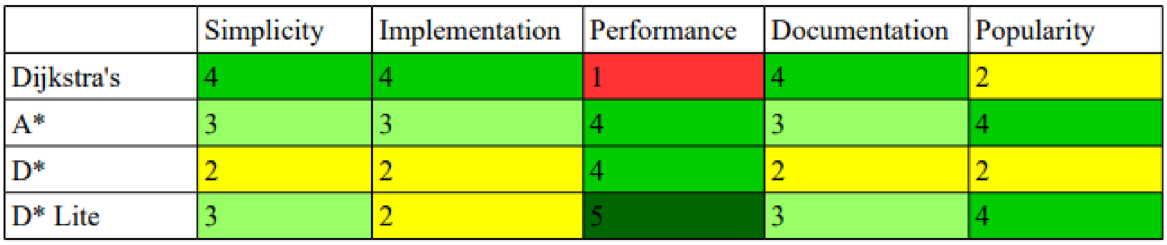


```
Algorithm 3: D* Lite
Function Key(s):
    return
      $[min(g(s), rhs(s)) + h(s_{start}, s) + k_m; min(g(s), rhs(s))]$
Function UpdateVertex(s):
    if $s \neq s_{goal}$ then
        $rhs(s) = min_{s' \in Succ(s)}(cost(s, s') + g(s'))$
    end
    if $s \in OPEN$ then
        $OPEN.remove(s)$
    end
    if $g(s) \neq rhs(s)$ then
        $OPEN.insert(s, Key(s))$
    end
Function ComputePath():
    while $OPEN.TopKey() < Key(s_{start})$ OR $rhs(s_{start}) \neq g(s_{start})$ do
        $k_{old} = OPEN.TopKey()$
        $s = OPEN.Pop()$
        if $k_{old} < Key(s)$ then
            $OPEN.insert(s, Key(s))$
        else if $g(s) > rhs(s)$ then
            $g(s) = rhs(s)$
            forall $s' \in Pred(s)$ do
                UpdateVertex($s'$)
            end
        else
            $g(s) = \infty$
            forall $s' \in Pred(s) \cup \{s\}$ do
                UpdateVertex($s'$)
            end
    end
Function Main():
    forall $s \in S$ do
        $rhs(s) = g(s) = \infty$
    end
    $s_{last} = s_{start}$
    $OPEN = \emptyset$
    $rhs(s_{goal}) = 0; k_m = 0$
    $OPEN.insert(s_{goal}, Key(s_{goal}))$
    ComputePath()
    while $s_{start} \neq s_{goal}$ do
        $s_{start} = argmin_{s' \in Succ(s_{start})}(cost(s_{start}, s') + g(s'))$
        Move to $s_{start}$
        Scan for cell changes in environment (e.g. sensor ranges)
        if Cell changes detected then
            $k_m = k_m + h(s_{last}, s_{start})$
            $s_{last} = s_{start}$
            forall $s \in CHANGES$ do
                Update cell $s$ state
                forall $s' \in Pred(s) \cup \{s\}$ do
                    UpdateVertex($s'$)
                end
            end
            ComputePath()
        end
    end
```

Figure 7.4: Source: https://www.researchgate.net/figure/Pseudo-code-of-D-Lite-algorithm_fig9_327985957

Pseudocode of D* Lite

We think that the A* algorithm is a good place to start with plenty of available guides and documentation. Moving on to D* Lite however could be a natural stepping stone, as the incremental search functionality might be a necessity for truly efficient path finding. The figure below shows the experience we had and the impressions we got of the five aspects mentioned above regarding each algorithm. This is meant to showcase more of a subjective impression, and not necessarily a scoring table where highest collective score wins. For instance, performance would typically be a very highly valued aspect, whereas popularity might not be as important.



	Simplicity	Implementation	Performance	Documentation	Popularity
Dijkstra's	4	4	1	4	2
A*	3	3	4	3	4
D*	2	2	4	2	2
D* Lite	3	2	5	3	4

Figure 7.5: Algorithm impression scores

[66] [71] [72]

7.6 Future work

In this section of the paper we will write about what comes next in development and in what order we prioritize to work further on in this project. Work further on the machine learning aspect and compare that to path planning we all ready have implemented in the project. 3D map will be the next in line step and how make a map in a higher speed compared to LIDAR mapping in low speed.

7.6.1 Imitation learning

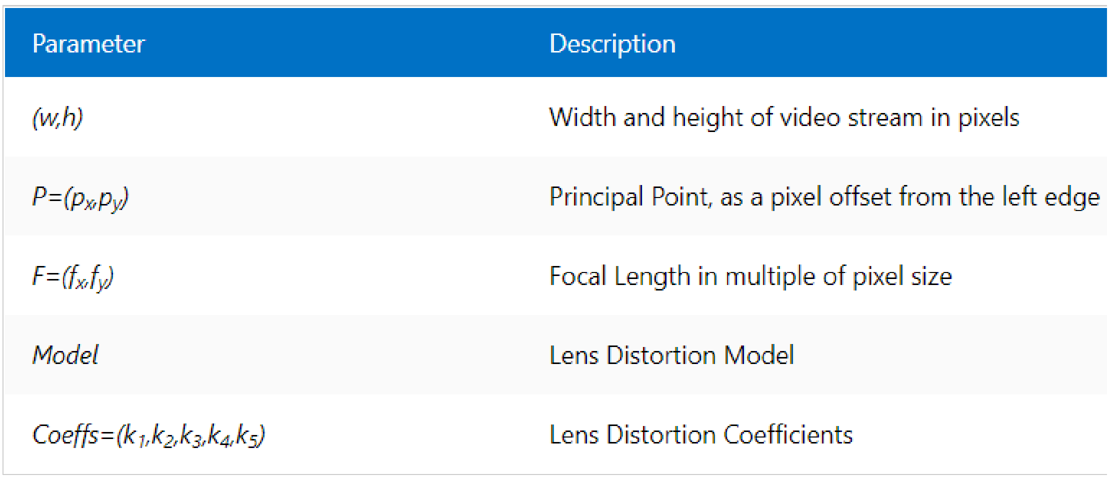
To proceed in development with machine learning, where the vehicle will fully drive by it self. To achieve this is to use a method called sim2real and it utilize a imitation learning, where an expert can demonstrate to the algorithm a certain behavior and it will try to learn the same. In our instance we would have used the joystick to demonstrate how to drive, and then the program will try to learn thru imitating the demonstration. If the hardware on the car does not support training in real time it is a way to train in while it is being simulated and then transfer what the robot learn in the simulation, and then it will try to drive the same course as in the program. [48]

7.6.2 3D mapping and car driving in high speed

With the limitation from LIDAR scanning not managing high speed,making a map with the camera and trying to make 3D map the goal would be to try to have higher speed while the process is being mapped.
[48]

7.6.3 3D mapping

In order to achieve 3D mapping, 3 coordinates are required as input and these are X, Y and Z. X referring to right/left length, Y referring to downwards length and Z referring to forward length.



Parameter	Description
(w,h)	Width and height of video stream in pixels
$P=(p_x,p_y)$	Principal Point, as a pixel offset from the left edge
$F=(f_x,f_y)$	Focal Length in multiple of pixel size
Model	Lens Distortion Model
$Coeffs=(k_1,k_2,k_3,k_4,k_5)$	Lens Distortion Coefficients

Figure 7.6: Source:https://dev.intelrealsense.com/docs/projection-texture-mapping-and-occlusion-with-intel-realsense-depth-cameras

These are the parameters for the sensor we were given in this project, which are all needed for accurate 3D mapping. The sensor should then be able to retrieve the necessary information and form a 3D render of its current point of view. As well as understand in greater depth how far away an object is from the sensor or if the car points upwards or downwards. The sensor should also be able to see colors of objects and the surrounding area within its field of view, this can potentially be used to assign additional orders or functions. An example of this could be to stop for a couple of seconds or slow down if it comes within a certain range of a specific color. Another potential in having a 3D sensor working with 2D sensor is cross referencing, leading to reduced mistakes or mishaps. A 3D sensor would also allow for updates on whether the car is going up or down, making up for potential mistakes the 2D sensor might make. [34]

7.6.3.1 Resources

GitHub: https://github.com/IntelRealSense
This is a repository published officially by Intel whom are the creators of the 3D sensor given to us, future work on the 3D sensor will most likely use this repository to setup and use the sensor as it has an in-depth installation guide.

7.6.4 Algorithms and scripting

Initially we had landed on the idea of trying out either A*, D* or D* Lite for our first script. They seemed like appealing algorithms to start with as they were fairly popular and had simpler pseudocode compared to other pathfinding algorithms. All these algorithms

are also somewhat similar in behavior, so any further attempts in the future would most likely involve testing all of them for performance and ease of use.
The general process of a possible implementation would first begin by fetching coordinate data from the LIDAR sensor. These coordinates could then be used in our script or program to make a node object. A node would have to consist of x and Y coordinates as well as some other properties such as G, F and H cost so that the program could properly evaluate nodes. After finding the node with the lowest F cost we could return its coordinates to the robot through the use of a ROS method, such as SendingSimpleGoals.
Actual behavior of the vehicle would depend on the algorithm we pick. A* for instance does not technically account or allow for re planning of paths like D* does, so our assumption is that A* would find a fully cleared path before the vehicle drives. In that case it's possible that an entire 2D map and its coordinates would have to be loaded into the program. All these coordinates could then be made into node objects to be further evaluated by the program (algorithm). Finally, the algorithm would then find a full path of coordinates that we could then return to the vehicle which it would then attempt to follow.

Chapter 8

Conclusion

8.1 Main purpose

We decided to use Python due to most of the group having experience with Python as a coding language. Python is also widely supported within the sources we used in our autonomous car project. C++ was also considered as an option but was dropped due to the group having little to no experience using the language. ROS also supports a version of Python called ”rospy”, which in short is a Python library for ROS.
Robot Operating System(ROS) is our main system for connecting and allowing our car to drive and understand autonomous orders. We learned early on that it is a challenge to learn and use, our greatest challenge was understanding error messages. But our first challenge in using ROS came in the ways of accurately mapping an area or track in our case.
We required a map for our car to have the ability of autonomous driving. This was achieved by using Hector SLAM with the LIDAR sensor, giving us a 2D map. This did however pose a challenge due to the speed of the mapping, driving too fast always resulted in the map breaking up and the car losing itself. Our most consistent way of creating a map was driving as slow as possible, but large open spaces still pose great difficulty for the sensor. After successfully creating a map we looked to the next step which revolves around autonomous driving, or making the car successfully drive through the map on its own.
The group decided to use a path finding algorithm due to only one of our members having knowledge on Machine Learning. Thanks to an easily accessible track we agreed to use A*(a star) but also discussed the use of D*(d star) because of it having a multitude of variations. With the group agreeing on which sort of path finding algorithm to use in the beginning, a new topic arose, that of safety mechanisms and how we can prevent damage to the car.
These are protocols and instructions we use on the car in preparations for autonomous driving. Having the car stop once it comes within a certain defined distance of an object, defined in meters. Resulting in the car stopping as soon as something comes to close, this did not go much further in terms of testing but ideas such as turning left or right depending on location was discussed. But in the end we where able to accomplish some of our goals, but we did commit research into the next steps in the further developing the project.
Listed below are a few of the goals we would like to achieve or what we as a group felt was the next logical step for development. Next phase in development was to achieve implementing imitation learning which revolves around the car being able to drive by itself

and make its own decisions in situations where driving from A to B could pose potential dangers. Another being 3D mapping which would allow the car to learn more from its environment, one major one being to spot uphill and downhill positions or simply knowing if what is in front of the vehicle when it goes up or down depending on the situation.
Finally we can conclude that our project can be used as a foundation for the next project of making the car drive by itself either using the path planning method or using ML with Imitation learning.

Bibliography

[1] Likhachev and Carnegie Mellon University. “Real-time planning and re-planning ii: Planning with freespace assumption, agent-centered search.” (Sep. 25, 2010), [Online]. Available: https://www.cs.cmu.edu/~motionplanning/lecture/agentcenteredsearch.pdf.

[2] LizMurphy. “Vslam.” (Dec. 18, 2012), [Online]. Available: http://wiki.ros.org/vslam (visited on 05/13/2022).

[3] T. . Contributor. “Write once, run anywhere (wora).” (Mar. 14, 2013), [Online]. Available: https://www.techtarget.com/whatis/definition/write-once-run-anywhere-WORA (visited on 05/17/2022).

[4] Lague. “A* pathfinding (e01: Algorithm explanation).” british. (Dec. 16, 2014), [Online]. Available: https://www.youtube.com/watch?v=-L-WgKMFuhE (visited on 05/11/2022).

[5] StefanKohlbrecher. “Hector$_slam - roswiki$.” (Apr. 17, 2014), [Online]. Available: http://wiki.ros.org/hector_slam (visited on 05/13/2022).

[6] TullyFoote. “Rviz/displaytypes/grid - ros wiki.” (Jan. 7, 2014), [Online]. Available: http://wiki.ros.org/rviz/DisplayTypes/Grid (visited on 04/11/2022).

[7] A. . Ademovic. “An introduction to robot operating system: The ultimate robot application framework.” (Mar. 4, 2016), [Online]. Available: https://www.toptal.com/robotics/introduction-to-robot-operating-system (visited on 05/11/2022).

[8] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, “Chapter 7 - autonomous guided vehicles,” in *Wheeled Mobile Robotics*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds., Butterworth-Heinemann, 2017, pp. 387–418, ISBN: 978-0-12-804204-5. DOI: https://doi.org/10.1016/B978-0-12-804204-5.00007-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012804204500007X.

[9] PRG UMD Teaching. “Lecture 7: Rrt and d star.” (Sep. 21, 2017), [Online]. Available: https://www.youtube.com/watch?v=TNajwFQP-Ys (visited on 05/09/2022).

[10] F. . Toydemir. “Top five use cases of...” (Nov. 6, 2017), [Online]. Available: https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/ (visited on 02/02/2022).

[11] Xu. “Pathfinding.js.” (Apr. 24, 2017), [Online]. Available: https://qiao.github.io/PathFinding.js/visual/ (visited on 05/09/2022).

[12] E. . Ackerman. “Learn to program self-driving cars (and help duckies commute) with duckietown.” (Aug. 20, 2018), [Online]. Available: https://spectrum.ieee.org/learn-to-program-self-driving-cars-and-help-duckies-commute-with-duckietown#toggle-gdpr (visited on 02/15/2022).

[13] T. . Contributor. “Simultaneous localization and mapping.” (Jul. 30, 2018), [Online]. Available: https://www.techtarget.com/whatis/definition/simultaneous-localization-and-mapping (visited on 05/14/2022).

[14] M. . Heller. “What is cuda? parallel programming for gpus.” (Aug. 30, 2018), [Online]. Available: https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html (visited on 02/02/2022).

[15] B. . Limone. “Global robot path planning vs local path planning | energid.” (Nov. 18, 2018), [Online]. Available: https://www.energid.com/blog/what-is-global-path-planning-how-does-it-compare-to-local-path-planning (visited on 04/22/2022).

[16] Marguedas. “Melodic - ros wiki.” (Aug. 14, 2018), [Online]. Available: http://wiki.ros.org/melodic (visited on 04/11/2022).

[17] P. . Marin-Plaza. “Global and local path planning study in a ros-based research platform for autonomous vehicles.” (Feb. 22, 2018), [Online]. Available: https://www.hindawi.com/journals/jat/2018/6392697/ (visited on 04/22/2022).

[18] MIT OpenCourseWare. “Advanced 1. incremental path planning.” (Oct. 26, 2018), [Online]. Available: https://www.youtube.com/watch?v=_4u9W1xOuts (visited on 05/09/2022).

[19] “Rviz - ros wiki.” (May 16, 2018), [Online]. Available: http://wiki.ros.org/rviz (visited on 02/11/2022).

[20] R. . Siddiqui. “Path planning using potential field algorithm - rymsha siddiqui.” (Jul. 29, 2018), [Online]. Available: https://medium.com/@rymshasiddiqui/path-planning-using-potential-field-algorithm-a30ad12bdb08.

[21] undefined. “[ros in 5 mins] 025 - what is rviz?” (Jul. 29, 2018), [Online]. Available: https://www.youtube.com/watch?v=yLwr5Zhr_t8 (visited on 02/19/2022).

[22] ——, “[ros in 5 mins] 028 - what is gazebo simulation?” (Aug. 4, 2018), [Online]. Available: https://www.youtube.com/watch?v=mranHM9wn0g (visited on 02/01/2022).

[23] J. . Brownlee. “Introduction to the python deep learning library tensorflow.” (Dec. 19, 2019), [Online]. Available: https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/ (visited on 02/03/2022).

[24] Computer Hope. “What is ray casting?” (Aug. 2, 2019), [Online]. Available: https://www.computerhope.com/jargon/r/ray-casting.htm (visited on 05/09/2022).

[25] IntelliVision. “Intelligent video analytics.” (Oct. 30, 2019), [Online]. Available: https://www.intelli-vision.com/intelligent-video-analytics/ (visited on 05/16/2022).

[26] Javanmardi, GU, Javanmardi, and Kamijo. “Autonomous vehicle self-localization based on abstract map and multi-channel lidar in urban area.” (Apr. 1, 2019), [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0386111217301206 (visited on 04/20/2022).

[27] Linux. “What is linux?” (Aug. 23, 2019), [Online]. Available: https://www.linux.com/what-is-linux/ (visited on 02/01/2022).

[28] melwin. “Rviz/displaytypes/robotmodel - ros wiki.” (Oct. 11, 2019), [Online]. Available: http://wiki.ros.org/rviz/DisplayTypes/RobotModel (visited on 04/07/2022).

[29] redhat. “What is an ide?” (Jan. 8, 2019), [Online]. Available: https://www.redhat.com/en/topics/middleware/what-is-ide (visited on 05/12/2022).

[30] Udemy. “[udemy course] what is tf in ros?” (Jun. 30, 2019), [Online]. Available: https://www.youtube.com/watch?v=A0ocyo9Ihwc&ab_channel=AnisKoubaa (visited on 04/11/2022).

[31] Villinger and Avast. “What is ram and why is it important?” (Nov. 7, 2019), [Online]. Available: https://www.avast.com/c-what-is-ram-memory (visited on 05/13/2022).

[32] AnswerRocket. “Top 8 ai frameworks and machine learning libraries | answerrocket.” (Aug. 13, 2020), [Online]. Available: https://www.answerrocket.com/ai-libraries/ (visited on 02/02/2022).

[33] A. . Automaticaddison. “What is the difference between rviz and gazebo? – automatic addison.” (Jun. 24, 2020), [Online]. Available: https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/ (visited on 04/13/2022).

[34] Dorodnicov, Grunnet-Jepsen, and Wen. “Projection, texture-mapping and occlusion with intel® realsense™ depth cameras.” (Jul. 5, 2020), [Online]. Available: https://dev.intelrealsense.com/docs/projection-texture-mapping-and-occlusion-with-intel-realsense-depth-cameras (visited on 05/10/2022).

[35] f1tenth. “Github - f1tenth/particle$_filteratmelodic$.” (Mar. 2, 2020), [Online]. Available: https://github.com/f1tenth/particle_filter/tree/melodic (visited on 04/04/2022).

[36] Hellsesø, Sjøvold, and Svea-Lochert, “Maneuver autonomous vehicles with arm gestures,” May 28, 2020.

[37] Jönsson and Stenbäck, “Monte-carlo tree search in continuous action spaces for autonomous racing,” Jun. 14, 2020. [Online]. Available: https://f1tenth.org/publications/Monte-Carlo.pdf (visited on 04/12/2022).

[38] OpenCV. “About.” (Nov. 4, 2020), [Online]. Available: https://opencv.org/about/ (visited on 02/03/2022).

[39] TullyFoote. “Tf/tutorials - ros wiki.” (Apr. 28, 2020), [Online]. Available: http://wiki.ros.org/tf/Tutorials (visited on 04/11/2022).

[40] E. . Ackerman. “Learn to program self-driving cars (and help duckies commute) with duckietown.” (Jun. 24, 2021), [Online]. Available: https://spectrum.ieee.org/learn-to-program-self-driving-cars-and-help-duckies-commute-with-duckietown#toggle-gdpr (visited on 02/02/2022).

[41] J. . Brownlee. “Your first deep learning project in python with keras step-by-step.” (Oct. 12, 2021), [Online]. Available: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (visited on 02/02/2022).

[42] E. . Burns. “Machine learning.” (Mar. 30, 2021), [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML (visited on 02/02/2022).

[43] DavidLu. "Urdf/tutorials/building a visual robot model with urdf from scratch - ros wiki." (Oct. 7, 2021), [Online]. Available: http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch (visited on 04/15/2022).

[44] I. . C. . Education. "Machine learning." (Nov. 5, 2021), [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning (visited on 02/02/2022).

[45] Ford Motor Company. "How ford's next-gen test vehicle lays the foundation for our self-driving business." (Dec. 16, 2021), [Online]. Available: https://medium.com/self-driven/how-fords-next-gen-test-vehicle-lays-the-foundation-for-our-self-driving-business-aadbf247b6ce (visited on 05/19/2022).

[46] "Ford, argo ai, and walmart to launch autonomous vehicle delivery service in three u.s. cities | ford media center." (Sep. 15, 2021), [Online]. Available: https://media.ford.com/content/fordmedia/fna/us/en/news/2021/09/15/ford-argo-ai-and-walmart.html (visited on 04/11/2022).

[47] T. . Francis. "What is lane-keeping assist?" (Sep. 14, 2021), [Online]. Available: https://www.autoexpress.co.uk/tips-advice/356028/what-lane-keeping-assist (visited on 05/11/2022).

[48] Lőrincz. "A brief overview of imitation learning - smartlab ai." (Dec. 11, 2021), [Online]. Available: https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c (visited on 05/13/2022).

[49] V. . Mazzari. "Robotic simulation scenarios with gazebo and ros." (Jul. 6, 2021), [Online]. Available: https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/ (visited on 02/07/2022).

[50] nvidia. "Jetson modules." (Dec. 3, 2021), [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules (visited on 02/11/2022).

[51] pybullet. "Bullet real-time physics simulation | home of bullet and pybullet: Physics simulation for games, visual effects, robotics and reinforcement learning." (May 21, 2021), [Online]. Available: https://pybullet.org/wordpress/ (visited on 02/03/2022).

[52] rosettacode. "Ray-casting algorithm - rosetta code." (Dec. 11, 2021), [Online]. Available: https://rosettacode.org/wiki/Ray-casting_algorithm (visited on 05/09/2022).

[53] Wikipedia contributors. "Nvidia jetson." (Jul. 30, 2021), [Online]. Available: https://en.wikipedia.org/wiki/Nvidia_Jetson (visited on 02/11/2022).

[54] C. . Woodford. "Roomba® robot vacuum cleaners." (May 20, 2021), [Online]. Available: https://www.explainthatstuff.com/how-roomba-works.html (visited on 02/02/2022).

[55] M. Bosello, R. Tse, and G. Pau, "Train in austria, race in montecarlo: Generalized rl for cross-track f1<sup>tenth</sup> lidar-based races," in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 290–298. DOI: 10.1109/CCNC49033.2022.9700730.

[56] GeeksforGeeks. "A* search algorithm." (Apr. 13, 2022), [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/ (visited on 05/11/2022).

[57] ——, "History of python." (Feb. 11, 2022), [Online]. Available: https://www.geeksforgeeks.org/history-of-python/ (visited on 03/20/2022).

[58] ——, "Top 10 programming languages to learn in 2022." (Jan. 7, 2022), [Online]. Available: https://www.geeksforgeeks.org/top-10-programming-languages-to-learn-in-2022/ (visited on 03/01/2022).

[59] J. . Hartman. "Jvm | what is java virtual machine its architecture." (Feb. 12, 2022), [Online]. Available: https://www.guru99.com/java-virtual-machine-jvm.html (visited on 03/04/2022).

[60] D. . Johnson. "Keras tutorial: What is keras? how to install in python [example]." (Feb. 12, 2022), [Online]. Available: https://www.guru99.com/keras-tutorial.html (visited on 02/02/2022).

[61] nvidia. "Jetpack sdk." (Feb. 24, 2022), [Online]. Available: https://developer.nvidia.com/embedded/jetpack (visited on 02/11/2022).

[62] ——, "Nvidia deepstream sdk." (Feb. 28, 2022), [Online]. Available: https://developer.nvidia.com/deepstream-sdk (visited on 02/02/2022).

[63] P. . "Automation advantages and disadvantages | what are the top advantages and disadvantages of automation?" (Feb. 14, 2022), [Online]. Available: https://www.aplustopper.com/automation-advantages-and-disadvantages/ (visited on 05/19/2022).

[64] O. Robotics. "Ros introduction (captioned)," Vimeo. (2022), [Online]. Available: https://vimeo.com/osrfoundation/ros (visited on 02/17/2022).

[65] techcrunch. "Techcrunch is part of the yahoo family of brands." (Jan. 13, 2022), [Online]. Available: https://techcrunch.com/2022/01/13/serve-robotics-new-autonomous-sidewalk-delivery-robots-dont-require-human-assist/ (visited on 02/05/2022).

[66] Wikipedia contributors. "A* search algorithm." (May 18, 2022), [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm (visited on 04/11/2022).

[67] ——, "Ai accelerator." (Feb. 2, 2022), [Online]. Available: https://en.wikipedia.org/wiki/AI_accelerator (visited on 02/02/2022).

[68] ——, "Argo ai." (Apr. 28, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Argo_AI (visited on 05/19/2022).

[69] ——, "Automation." (May 16, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Automation (visited on 05/19/2022).

[70] ——, "C++." (Feb. 22, 2022), [Online]. Available: https://en.wikipedia.org/wiki/C%2B%2B (visited on 03/01/2022).

[71] ——, "D*." (Apr. 4, 2022), [Online]. Available: https://en.wikipedia.org/wiki/D* (visited on 04/06/2022).

[72] ——, "Dijkstra's algorithm." (May 10, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (visited on 04/11/2022).

[73] ——, "Dijkstra's algorithm." (May 10, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (visited on 05/09/2022).

[74] ——, “Java (programming language).” (Feb. 23, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language) (visited on 03/01/2022).

[75] ——, “Java virtual machine.” (Feb. 13, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Java_virtual_machine (visited on 03/04/2022).

[76] ——, “Lidar.” (May 11, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Lidar (visited on 05/06/2022).

[77] ——, “Ray casting.” (May 10, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Ray_casting (visited on 05/09/2022).

[78] ——, “Robot operating system.” (Apr. 5, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System (visited on 04/06/2022).

[79] ——, “Robotics.” (May 17, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Robotics (visited on 05/19/2022).

[80] “About.” (), [Online]. Available: https://f1tenth.org/about.html (visited on 02/12/2022).

[81] AMD xilinx. “What’s a som.” (), [Online]. Available: https://www.xilinx.com/products/som/what-is-a-som.html (visited on 05/16/2022).

[82] boston dynamics. “Spot® - the agile mobile robot.” (), [Online]. Available: https://www.bostondynamics.com/products/spot (visited on 02/02/2022).

[83] britannica. “Artificial intelligence | definition, examples, types, applications, companies, facts.” (), [Online]. Available: https://www.britannica.com/technology/artificial-intelligence (visited on 05/14/2022).

[84] ——, “Massachusetts institute of technology | history facts.” (), [Online]. Available: https://www.britannica.com/topic/Massachusetts-Institute-of-Technology (visited on 05/15/2022).

[85] Cyberbotics Ltd. “Webots user guide.” (), [Online]. Available: https://cyberbotics.com/doc/guide/introduction-to-webots (visited on 02/08/2022).

[86] duckietown-foundation. “History – duckietown.” (), [Online]. Available: https://www.duckietown.org/about/history (visited on 02/17/2022).

[87] “F1tenth.” (), [Online]. Available: https://slugbotics.com/projects/formula1tenth/ (visited on 02/14/2022).

[88] gartner. “Definition of object-oriented programming - gartner information technology glossary.” (), [Online]. Available: https://www.gartner.com/en/information-technology/glossary/oop-object-oriented-programming (visited on 05/17/2022).

[89] Gazebo Simulator. “Gazebo.” (), [Online]. Available: https://gazebosim.org/tutorials/?tut=ros_urdf (visited on 04/03/2022).

[90] Gazebosim. “Gazebo.” (), [Online]. Available: http://gazebosim.org/ (visited on 01/22/2022).

[91] GeoSLAM. “What is slam? simultaneous localization and mapping.” british. (), [Online]. Available: https://geoslam.com/what-is-slam/ (visited on 01/21/2022).

[92] “History – duckietown.” (), [Online]. Available: https://www.duckietown.org/about/history (visited on 02/02/2022).

[93] Intel. “What is a gpu? graphics processing units defined.” (), [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html (visited on 05/14/2022).

[94] International Society of Automation. “What is automation? - isa.” (), [Online]. Available: https://www.isa.org/about-isa/what-is-automation (visited on 05/19/2022).

[95] mathworks. “Monte carlo localization algorithm - matlab simulink - mathworks nordic.” (), [Online]. Available: https://se.mathworks.com/help/nav/ug/monte-carlo-localization-algorithm.html (visited on 04/29/2022).

[96] National Ocean and Atmospheric Administration US. Department of Commerce. “What is lidar?” (), [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html (visited on 05/18/2022).

[97] nvidia. “Introduction to jetpack :: Nvidia jetpack documentation.” (), [Online]. Available: https://docs.nvidia.com/jetson/jetpack/introduction/index.html#libraries (visited on 02/11/2022).

[98] ——, “Jetson nano 2gb developer kit.” british. (), [Online]. Available: https://www.nvidia.com/en-gb/autonomous-machines/embedded-systems/jetson-nano/education-projects/ (visited on 02/02/2022).

[99] ——, “Nvidia embedded systems for next-gen autonomous machines.” (), [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/ (visited on 02/11/2022).

[100] PCMag. “Definition of java virtual machine.” (), [Online]. Available: https://www.pcmag.com/encyclopedia/term/java-virtual-machine (visited on 05/13/2022).

[101] programiz. “Dijkstra’s algorithm.” (), [Online]. Available: https://www.programiz.com/dsa/dijkstra-algorithm (visited on 05/09/2022).

[102] “Rviz/displaytypes/grid.” (), [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/rviz(2f)DisplayTypes(2f)Grid.html (visited on 04/11/2022).

[103] stereolabs. “Ros - data display with rviz | stereolabs.” (), [Online]. Available: https://www.stereolabs.com/docs/ros/rviz/ (visited on 02/18/2022).

[104] TensorFlow. “Why tensorflow.” (), [Online]. Available: https://www.tensorflow.org/about (visited on 02/02/2022).

[105] Tutorialspoint. “C++ tutorial.” (), [Online]. Available: https://www.tutorialspoint.com/cplusplus/index.htm (visited on 03/01/2022).

[106] w3schools. “C++ introduction.” (), [Online]. Available: https://www.w3schools.com/CPP/cpp_intro.asp (visited on 03/01/2022).

[107] “Webots: Robot simulator.” (), [Online]. Available: https://www.cyberbotics.com/#cyberbotics (visited on 02/04/2022).

[108] “What is linux?” (), [Online]. Available: https://opensource.com/resources/linux (visited on 02/09/2022).

[109] “What is slam (simultaneous localization and mapping) –.” (), [Online]. Available: https://se.mathworks.com/discovery/slam.html (visited on 02/01/2022).

[110] Wikipedia contributors and Wikipedia. “Python (programming language).” (), [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language) (visited on 03/03/2022).

Glossary

AI An Artificial intelligence(AI), is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. [83]. vi, 10, 11, 17, 18

API An Application Programming Interface (API) is a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.[74]. 11

CPU A Central Processing Unit (CPU), also called a central processor, main processor or just processor, is the electronic circuitry that executes instructions comprising a computer program [70]. 12

CRM A Customer Relationship Management(CRM)is a process in which a business or other organization administers its interactions with customers, typically using data analysis to study large amounts of information.[67]. 10

EXP A Experimental (EXP). 43

GPU A Graphical Processing Unit (GPU) specialized processor originally designed to accelerate graphics rendering. GPUs can process many pieces of data simultaneously, making them useful for machine learning, video editing, and gaming applications. [93]. 10–12

IVA An Intelligent Video Analysis(IVA) products add artificial intelligence to cameras by analyzing video content in real-time, extracting metadata, sending out alerts and providing actionable intelligence to security personnel or other systems.[25]. 11

JVM A Java Virtual Machine(JVM) is the runtime engine of the Java Platform, which allows any program written in Java or other language compiled into Java bytecode to run on any computer that has a native JVM. JVMs run in both clients and servers, and the Web browser can activate the JVM when it encounters a Java applet.[100]. v, 7, 16

LIB A Library (LIB). 43

LIDAR A Light Detection And Ranging(LIDAR)is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth.. v, vii, 1, 3, 5, 8, 17, 19, 23, 28, 31, 38–41, 45, 49, 51, 53

LKAS A Lane-keeping assist systems(LKAS) is designed to help keep a vehicle centered in a detected lane, applying mild steering torque if it determines the vehicle is drifting toward the side of the lane.[80]. 18

ML A Machine Learning(ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so.[42]. 10, 23

OOP An Object Oriented Programming(OOP) is a style of programming characterized by the identification of classes of objects closely linked with the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods.[88]. 16

OS An Operating System(OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.[105]. 9, 25

ROS A Robot Operating System(ROS) is not an actual operating system, but a framework and set of tools that provide functionality of an operating system on a heterogeneous computer cluster. Its usefulness is not limited to robots, but the majority of tools provided are focused on working with peripheral hardware.[7]. vi, vii, ix, 5–9, 15, 23, 25, 26, 28, 39, 43–45, 51, 53

SDK A Software Development Kit(SDK)s a collection of software development tools in one installable package. They facilitate the creation of applications by having a compiler, debugger and sometimes a software framework. They are normally specific to a hardware platform and operating system combination.[75]. 11

SLAM A Simultaneous Localization and Mapping(SLAM) is the synchronous location awareness and recording of the environment in a map of a computer, device, robot, drone or other autonomous vehicle.[13]. v, 5, 7, 8, 39

SOM A System on Module(SOM) provides the core components of an embedded processing system — including processor cores, communication interfaces, and memory blocks — on a single production-ready printed circuit board (PCB).[81]. 11

WORA A Write once, run anywhere(WORA) is a term that refers to a particular program's supposed ability to run on all common OSs (operating systems). The term, sometimes also expressed as write once, run everywhere (WORE), was originally coined by Sun Microsystems in reference to Java.[3]. 16

XML An Extensible Markup Language(XML) is a markup language similar to HTML, but without predefined tags to use. Instead, you define your own tags designed specifically for your needs.[63]. 6

Appendix A

List of code

Below we have tried to implement a path planning algorithm, safety controller and a vehicle controller, where the code work together so we can make sure the vehicle does not crash when it drives by it self with path planning.

```python
#!/usr/bin/env python

# Author: Fredrik Sundre Lauritzen, Magnus Willy Eilefsen and Kristoffer Snopestad Søderkvist
# Date written: 05.05.2022
# It is written where we attempt to make a path planning script
# Which can be used as foundation to make a working path
# We have also used this github repository https://github.com/lsa-pucrs/global_planner_move_base
# for help and understanding.
# The repistory is written by Renan Guedes Maidana and Alexandre Amory.




import math
import rospy
from Queue import PriorityQueue
from nav_msgs.msg import Path, Odometry, OccupancyGrid, MapMetaData
from geometry_msgs.msg import PoseStamped, PoseWithCovarianceStamped

ROBOT_SIZE = 0.1
G_MULTIPLIER = 0.2

init = PoseStamped()    # Initial position
endGoal = PoseStamped()   # Goal position
startPos = PoseStamped()  # Current position


# map

map = OccupancyGrid()
info = MapMetaData()

# Utilities (e.g. flags, etc)
startInitial = 0
endPosGoal = 0

def initCallback(msg):
  global init
  global startInitial

def odomCallback(msg):
    global pos
    pos.pose = msg.pose.pose

def endGoalCallback(msg):
  global endGoal
  global endPosGoal
  endGoal = msg

def mapCallback(msg):
  global map
  map = msg

def infoCallback(msg):
  global info
  info = msg

def pathPlanning():
  # make a publisher
  planPublisher = rospy.Publisher('/path', Path, queue_size=10)


  # make subscribers
  odomSubscriber = rospy.Subscriber('odom', Odometry, odomCallback())
  initSubscriber = rospy.Subscriber('initialpose', PoseWithCovarianceStamped, initCallback)
  endGoalSubscriber = rospy.Subscriber('move_base_simple/goal', PoseStamped, endGoalCallback

  # start node
  rospy.init_node('global_planner', anonymous=True)

  path = search()
  path.frame_id = "map"

  while not rospy.is_shutdown():
    planPublisher.publish(path)


def search():


def poseToGrid(pose):
  # Set meters to grid units
  grid_x = int((pose.pose.position.x - info.origin.position.x) / info.resolution)
  grid_y = int((pose.pose.position.y - info.origin.position.y) / info.resolution)
  poseAux = PoseStamped()
  poseAux.pose.position.x = grid_x
  poseAux.pose.position.y = grid_y
  return poseAux



def setWayPoint(wx, wy):
  ax = PoseStamped()
  ax.pose.position.x = wx
  ax.pose.position.y = wy
  return ax
```

```


if __name__ == "__main__":
  from Saftey_clear import safe
  from Driving import startDrive

  rospy.init_node('test')
  Saftey_clear = Saftey_clear(args.sim)
  saftey_clear = Saftey_clear()

  # have to put other parameters to work.

  try:
    pathPlanning()
  except rospy.ROSInterruptException:
    pass
```

Listing A.1: Path planning

```
#!/usr/bin/env python
# the start on over saftey control to make sure the vehicle does not crash or gets damaged
# this is a no working code.
# Just here to be used as fondation if needed for further development. And it is supposed to be us
# We have also used this github repository https://github.com/MichaelBosello/f1tenth-RL
# for help and understanding .
# The repistory is written by Michael Bosello


import sys
import math
import rospy
from geometry_msgs.msg import Twist, Pose
from nav_msgs.msg import Odometry
from sensor_msgs.msg import LaserScan

LINEAR_VELOCITY = 0.2
ANGULAR_VELOCITY = 0.4
TOLERANCE = 0.3
ROBOT_RADIUS = 0.22
OBSTACLE_THRESHOLD = 0.78
EXIT_STATUS_ERROR = 1
EXIT_STATUS_OK = 0

class safe:
    def __init__(self):
        rospy.init_node('traveler', anonymous=True)
        self.vel_publisher = rospy.Publisher('/r1/cmd_vel', Twist, queue_size=10)
        self.odom_subscriber = rospy.Subscriber('/r1/odom', Odometry, self.odom_callback)
        self.laser_subscriber = rospy.Subscriber('/r1/kinect_laser/scan', LaserScan, self.laser_ca
        self.sonar_subscriber = rospy.Subscriber('/r1/pseudosonar/scan', LaserScan, self.sonar_cal
        # Make the robot hold is pos.
        self.pos = Pose()


    def odom_callback(self, odom):  # will continuously update current position

        self.pos = odom.pose.pose
        self.theta = 2 * math.atan2(self.pos.orientation.z, self.pos.orientation.w)
        self.curr_line = self.slope((self.pos.position.x, self.pos.position.y), self.goal)
```

Listing A.2: Saftey to make sure the vehicle does not crash

```
# the car control written as such it will work with the path planning and the saftey stops,
# Just here to be used as fondation if needed for further development.
# We have also used this github repository https://github.com/MichaelBosello/f1tenth-RL
# for help and understanding .
# The repistory is written by Michael Bosello


import rospy
from ackermann_msgs.msg import AckermannDriveStamped
from rospy.exceptions import ROSException

from threading import Thread
import time
import argparse



wait_publish = 0.005

reduce_speed = 4.5
steering_reduct = 4.5
reveres_reduct = 4.5
light_reduct = 2.4
reveres_sec = 1.5

min_reduct = 5

class Drive():
    def __init__(self, sensors, is_simulator=False):
        self.is_simulator = is_simulator
        if not is_simulator:
            topic = "/vesc/high_level/ackermann_cmd_mux/input/nav_0"
            max_steering = 0.34
            self.max_speed_reduction = reduce_speed
            self.steering_speed_reduction = steering_reduct
            self.backward_speed_reduction = reveres_reduct
            self.lightly_steering_reduction = light_reduct
            self.backward_seconds = reveres_sec

        self.max_speed = rospy.get_param("max_speed", 5)
        self.max_steering = rospy.get_param("max_steering", max_steering)
        self.drive_publisher = rospy.Publisher(topic, AckermannDriveStamped, queue_size=0)
        self.sensors = sensors
        self.stop()
        process = Thread(target=self.drive_command_runner)
        process.daemon = True
        process.start()
        print("max_speed: ", self.max_speed, ", max_steering: ", self.max_steering)
```

Listing A.3: The control of the vehicle