

Andreas Jonsebråten
Vegar Karlsen
Glenn R. Varhaug

Graph-based Path Planning in a Simulated Rough Terrain

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Irja Gravdahl
May 2023



Norwegian University of
Science and Technology



KONGSBERG

Andreas Jonsebråten
Vegar Karlsen
Glenn R. Varhaug

Graph-based Path Planning in a Simulated Rough Terrain

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Irja Gravdahl
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics



DEPARTMENT OF ENGINEERING CYBERNETICS

IELET2920 - BACHELOR THESIS AUTOMATION

Graph-based Path Planning in a Simulated Rough Terrain

Authors:

Andreas Jonsebråten

Vegar Karlsen

Glenn R. Varhaug

May, 2023

Abstract

This thesis details the implementation of the path planning framework, GBPlanner2, for a simulated version of the Lone Wolf All-Terrain Vehicle (ATV). Furthermore, it covers the implementation of via-point functionality to this framework. The specifications, design and implementation of five novel Gazebo worlds are described, as well as how the simulated Lone Wolf ATV was integrated with GBPlanner2 and the Gazebo worlds.

Lone Wolf is a collaborative project by Kongsberg Defence & Aerospace (KDA), where student interns work towards creating a fully autonomous ATV. This thesis builds on this project by implementing path planning as this is a crucial part of making the vehicle fully autonomous. Path planning functionality is realized by using GBPlanner2 as a framework. GBPlanner2 is an open-source library created by NTNU Autonomous Robots Lab (ARL) and includes mapping, path planning and motion planning for autonomous ground- and air vehicles. GBPlanner2 has no support for via-point path planning, this is therefore implemented, and a GUI for selecting via-points for path planning is implemented in RViz.

Five novel worlds were designed in Blender based on specific criteria decided in order to properly test the path planning algorithm in action. Four of the five worlds are made from the ground up, while the fifth world is made using height-data from KDA's test site for the ATV, Heistadmoen. After designing the worlds in Blender, they were imported into Gazebo, where trees and buildings could be added.

The resulting product is a functioning set of worlds for testing the ATVs path planning system. The path planning, including via-point functionality, works as intended. Because of the built-in support for motion planning in GBPlanner2, a crude motion planner is also included.

Sammendrag

Denne rapporten beskriver implementasjonen av baneplanleggingsrammeverket, GBPlanner2, for en simulert versjon av Lone Wolf ATVen. Rapporten dekker også implementasjon av viapunkt-funksjonalitet. Spesifikasjoner, design og implementasjon av fem nye verdener for bruk i Gazebo blir beskrevet, samt hvordan den simulerte versjonen av Lone Wolf ATVen ble integrert sammen med Gazebo og GBPlanner2.

Lone Wolf er et samarbeidsprosjekt ved Kongsberg Defence & Aerospace (KDA) der studenter gjennom sommerjobb hos KDA jobber for å lage en helautonom ATV. Denne rapporten bygger videre på dette prosjektet ved å implementere baneplanlegging, da dette er en viktig del av å gjøre kjøretøyet helautonomt. Funksjonalitet for baneplanlegging er realisert ved bruk av GBPlanner2 som rammeverk, et bibliotek som er utviklet av NTNU ARL. Biblioteket inkluderer kartlegging, baneplanlegging og banefølgning for bakke- og luftfartøy. GBPlanner2 har ikke støtte for viapunkter for baneplanlegging, og dette har derfor blitt implementert. I tillegg har et grafisk brukergrensesnitt for valg av viapunkter til baneplanleggingen blitt implementert i RViz.

Fem nye verdener ble designet i Blender basert på spesifikke kriterier som ble bestemt for å kunne teste baneplanleggingsalgoritmen i praksis. Fire av de fem verdenene ble laget fra bunnen av, mens den femte verdenen ble skapt ved bruk av høydedata fra Heistadmoen som er testområdet til KDA. Etter at verdenene ble designet, ble de importert til Gazebo hvor trær og bygninger ble lagt inn.

Det resulterende produktet er et fungerende sett med verdener hvor man kan teste ATVens baneplanleggingssystem. Baneplanleggingen, inkludert viapunkt-funksjonaliteten, fungerer slik som ønsket. GBPlanner2 sin innebygde banefølgning er også inkludert, slik at ATVen kan kjøre banen autonomt.

Preface

This thesis serves as the final project before completing a bachelor's degree in Electrical Engineering, specializing in robotics and automation, at the Norwegian University of Science and Technology (NTNU). The project has given us valuable experience with a highly relevant project. The project has been challenging, exciting and rewarding.

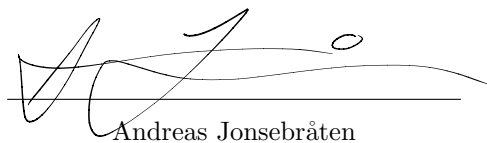
We would like to express our sincere appreciation to our supervisor at NTNU, Irja Gravdahl, for the plethora of feedback she has given us, the insight she has provided, and the dedication she has shown us.

We want to convey our thanks to Roger Werner Laug for providing us with this project and for giving us the insight and resources necessary to succeed. Furthermore, we would like to thank Chris André Brombach for his technical guidance during the project.

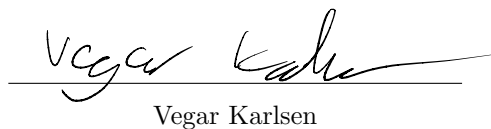
We would like to acknowledge and thank Sigrid Mellemseter who helped us catch up on the current status of the Lone Wolf project, and Andreas Traa Utkilen who held an introductory course on ROS for us.

Lastly, we would like to express our thanks to Mihir Dharmadhikari and Nikhil Khedekar at NTNU Autonomous Robots Lab for their invaluable insight into GBPlanner2, and the continued correspondence that answered any questions we had.

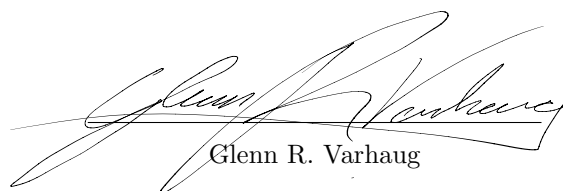
Trondheim, May 21, 2023



Andreas Jonsebråten



Vegar Karlsen



Glenn R. Varhaug

Contents

1	Introduction	1
1.1	Background	1
1.2	Project Assignment	2
1.3	Thesis Statement	2
1.4	Scope and Delimitations	3
1.5	Contributions	3
1.6	System Overview	4
1.7	Report Structure	5
2	Theory	7
2.1	Mapping	7
2.1.1	SLAM	7
2.1.2	Voxblox	8
2.2	Path Planning	8
2.2.1	What is Path Planning?	8
2.2.2	Graphs and Trees	9
2.2.3	Local vs. Global Path Planning	10
2.3	Path Planning Algorithms	10
2.3.1	Sampling-Based Algorithms	10
2.3.2	Node-Based Optimal Algorithms	12
2.4	Modeling	13
2.4.1	Blender	14
2.4.2	Gazebo	14
2.5	Visualization	14

2.5.1	RViz	14
2.6	ROS	14
2.6.1	ROS Components	15
2.6.2	ROS 1	15
2.6.3	ROS 2	15
3	Potential Solutions	17
3.1	Navigation 2	17
3.1.1	Structural Overview	17
3.1.2	Advantages	18
3.1.3	Constraints	18
3.2	GBPlanner 2	19
3.2.1	Structural Overview	19
3.2.2	Advantages	21
3.2.3	Constraints	21
3.3	Conclusion	21
4	Implementation	23
4.1	Background	23
4.2	ATV	23
4.3	Modeling	23
4.3.1	Blender	24
4.3.2	Gazebo	25
4.3.3	Simulated Worlds	25
4.3.4	Work Flow	30
4.4	Via-Point Implementation	31
4.4.1	Selecting Via-Points	31
4.4.2	GbPlanner Control Panel	31
4.4.3	Planner Control Interface	32
4.4.4	Global Planner	32
4.4.5	Via-Point Overview	33
4.5	System Overview	33

4.6	Modifications	34
4.6.1	Adding a New World	34
4.6.2	Change Settings in the Current World	35
4.6.3	Changing Maximum Inclination	35
4.6.4	Changing Preferences in RViz	36
5	Simulation	39
5.1	Launching the Simulator	39
5.1.1	How to use the Path Planning System in RViz	39
5.2	Test Procedure	40
5.2.1	Path Planning Using Via-Points	40
5.2.2	Path Planning Without Using Via-Points	44
5.2.3	Fully Explored World	46
5.3	Summary	47
6	Discussion	49
6.1	Further Developments	49
6.1.1	Known Issues	49
6.1.2	Suggested Developments	50
6.2	The Lone Wolf Project	51
7	Conclusion	53
	Bibliography	57
	Appendices	63
A	Bachelor Assignment	63
B	Poster	67

List of Figures

1.1	Lone Wolf ATV in the simulated version of Heistadmoen	1
1.2	Kongsberg Gruppen logo	2
1.3	General system overview	4
2.1	Simple continuous path planning visualization	9
2.2	Example of a graph and a tree	9
2.3	Overview of sampling-based algorithms	11
2.4	RRT searching approach	11
2.5	RRG search example	12
2.6	Overview of node-based optimal algorithms	13
2.7	Dijkstra's algorithm and A* search example	13
2.8	ROS data flow	15
3.1	Nav2 structure overview	18
3.2	2D Costmap	19
3.3	GBPlanner2 structure overview	20
4.1	Criteria for worlds	24
4.2	maze_world in Blender	25
4.3	simple_world in Blender	26
4.4	normal_world in Blender	26
4.5	complex_world in Blender	27
4.6	heistadmoen in Blender	27
4.7	heistadmoen in Gazebo	28
4.8	heistadmoen in RViz using Voxblox	28

4.9	<code>heistadmoen</code> in RViz using point cloud	29
4.10	World design work flow	30
4.11	Via-point numbering	31
4.12	<i>GbPlanner Control</i> panel	32
4.13	Via-point communication	33
4.14	General complete system overview	34
4.15	Gazebo user interface	35
4.16	Point cloud visualization	36
4.17	Voxblox visualization in RViz	37
4.18	Point cloud visualization in RViz	37
5.1	Rviz path planning buttons	40
5.2	Path planning in <code>maze_world</code>	41
5.3	Path planning in <code>simple_world</code>	41
5.4	Path planning in <code>normal_world</code>	42
5.5	Path planning in <code>complex_world</code>	42
5.6	Path planning in <code>heistadmoen</code>	43
5.7	Path planning in <code>heistadmoen</code>	43
5.8	Close up of via-point numbering on path	44
5.9	Path planning in <code>heistadmoen</code> without via-points	45
5.10	Path planning in <code>heistadmoen</code> without via-points, close up	45
5.11	<code>heistadmoen</code> when fully explored.	46
5.12	<code>normal_world</code> when fully explored.	47
6.1	Via-point path crossing	50
7.1	All five worlds in Blender	53
7.2	All five worlds in RViz	54
7.3	Graphical User Interface in RViz	54
7.4	Path planning in <code>heistadmoen</code> with and without the use of via-points	55

Listings

4.1	Where to find the <i>worlds</i> - and <i>models</i> -folder.	35
4.2	How to launch a new world through the terminal.	35
4.3	How to change spawn height in <code>gbplanner.launch</code>	35
4.4	How to change maximum inclination in <code>gbplanner_config.yaml</code>	36
5.1	How to launch the simulator in the terminal.	39
5.2	How to launch the simulator with a specific world.	39

List of Algorithms

1	Calculate path using via-points	33
---	---	----

Abbreviations

ARL Autonomous Robots Lab. i, ii, 19, 51

ATV All-Terrain Vehicle. i, ii, 1–5, 17, 21, 23–27, 30, 33, 35, 36, 39, 40, 44, 46, 47, 49–51, 53, 54

BlenderGIS Blender Geographic Information System. 24

GBPlanner2 Graph-Based Exploration Planner 2. i, ii, 3–5, 17, 19, 21, 22, 31–33, 40, 46, 49–51, 53, 54

GPS Global Positioning System. 2, 50

GUI Graphical User Interface. i, 23, 33, 36, 54

IMU Inertial Measurement Unit. 2

KDA Kongsberg Defence & Aerospace. i, ii, 1, 2, 57

KOG Kongsberg Gruppen. 2, 57

LiDAR Light Detection And Ranging. 2, 4, 7, 8, 23

Nav2 Navigation 2. 17, 18, 21

NTNU Norwegian University of Science and Technology. i, ii, 19, 23, 51

PCI Planner Control Interface. 19, 20, 31, 32

ROS Robot Operating System. 5, 7–9, 14, 15, 17, 19, 21, 33, 40, 50, 51

RRG Rapidly-exploring Random Graph. 11, 12, 21

RRT Rapidly-exploring Random Tree. 10, 11

SLAM Simultaneous Localization And Mapping. 2, 7, 8, 53

UAV Unmanned Aerial Vehicle. 8

Chapter 1

Introduction

In this chapter the necessary background information about the Lone Wolf project will be presented along with the specific project assignment and thesis statement for this bachelor's thesis. Furthermore, the limitations for the project assignment will be described, followed by a definition of the contributions to the Lone Wolf project provided in this project. Lastly there will be a brief summary of the report structure in order to guide the reader to specific chapters and sections in the report, with descriptions of the contents of each. Figure 1.1 shows the simulated ATV in a simulated version of Heistadmoen.

1.1 Background



Figure 1.1: Lone Wolf ATV in the simulated version of Heistadmoen. The large tree is a distinct landmark, and the white box is a building.

"Lone Wolf is a multi-disciplinary project in Kongsberg Defence & Aerospace (KDA). The project designs and builds an autonomous All-Terrain Vehicle (ATV) that can avoid obstacles" [1]. Since 2019, Lone Wolf has been developed by students working as summer interns at KDA. Recently, there has also been increasing attempts at opening the project for bachelor's and master's projects. The goal of the project is to create a fully autonomous ATV. In order to achieve this, a multitude

of complex systems need to be implemented and integrated. Some of these systems include control algorithms for navigation, safety features for operating the ATV around humans, sensors for gathering data for the onboard computer, and algorithms for autonomous decision-making.



Figure 1.2: Kongsberg Gruppen logo [2].

Kongsberg Defence & Aerospace is one of four business areas within Kongsberg Gruppen (KOG) [3]. Kongsberg Gruppen is a corporation delivering advanced technological solutions required to operate under extremely challenging conditions, figure 1.2 shows KOGs logo. Kongsberg Defence & Aerospace focuses on the defence- and aerospace industry, delivering technologies ranging from surveillance and space to remote weapon stations and missile systems [4].

As of the summer of 2022, the ATV is able to be operated remotely. Mounted on the ATV is a waterproof computer which controls the throttle, steering and breaking. Furthermore, a LiDAR, camera, IMU and a GPS are mounted on the vehicle. The LiDAR- and IMU-data is used as input to a SLAM-algorithm which maps the environment surrounding the ATV and precisely estimates the position of the vehicle in relation to the generated map. In order to test new implementations on the ATV, a Gazebo simulator is available with a 3D model of the vehicle which has been imported into the simulated environment, see section 2.4.1 for a description of Gazebo.

This thesis continues the work of previous students, with the project assignment, described in section 1.2, being a necessary goal in order to reach the main project goal as described above.

1.2 Project Assignment

The original project assignment proposed by KDA can be found in appendix A. This assignment is very broadly defined, and therefore needs to be narrowed down to a more precise thesis statement. The key elements of the project assignment are terrain-based path planning, Gazebo worlds, and via-points. In order to define the assignment as complete, these aspects have to be implemented. Furthermore, implementation of a method for the ATV to explore its surroundings in order to expand the local map generated by the onboard SLAM algorithm is desirable, but not necessary for completion.

1.3 Thesis Statement

Based on the specified project assignment from section 1.2, the thesis statement is as follows; *Implement a system for path planning in a simulated terrain-based environment with support for adding via-points to the desired path. Furthermore, design and implement multiple worlds into the Gazebo simulated environment.*

1.4 Scope and Delimitations

The various keywords in the thesis statement from section 1.3 can be interpreted in different ways depending on the readers familiarity with the subject matter. This section will therefore present the delimitations of the project in order to reduce the risk of misinterpretation of the projects scope.

As described in sections 1.2 and 1.3, the project aims at:

- Designing custom worlds for simulation in accordance to the criteria described in section 4.3 and figure 4.1.
- Designing a custom world that closely resembles the real-life test location, Heistadmoen.
- Implementing the custom worlds into Gazebo.
- Implementing a system for global path planning on the simulated ATV, see section 4.4.4.
- Implementing via-point functionality to the path planning system.

The following aspects are not covered by this thesis, and are as such outside the scope of this assignment:

- Implementation of motion planning; that is, planning of the specific movements necessary for the robot to traverse the path.
- Implementation of local path planning for handling dynamical changes in the environment, see section 2.2.1.

The scope and delimitations are guidelines for the reader to understand what has been the focus of this thesis; they are not, however, strict rules as to what will be included in the finished product. Motion planning will for example not be a prioritized implementation, but is in fact included in this thesis as the library used for path planning includes built-in motion planning as described in section 3.2.1.

1.5 Contributions

This section will clarify which parts of the project that can be contributed to the authors. The work of engineers rely on reiteration of previous work in order to suit the specific needs of the project at hand. Chapter 4 Implementation, describes all contributions from this project, and the main contributions are listed below:

- Design of five novel Gazebo worlds.
- Importation of the aforementioned worlds into the GBPlanner2 framework.
- Importation of Lone Wolf ATV-model into the GBPlanner2 framework.
- Setup of simulation environment for GBPlanner2's path planning algorithm.

- Implementation of via-point functionality into GBPlanner2's path planning framework.
- Implementation of a novel interface in RViz for use of via-point functionality.

1.6 System Overview

This section will serve as a general overview of the structure and composition of the project as a whole. For a more in-depth description of each component, see the specific chapter or section.

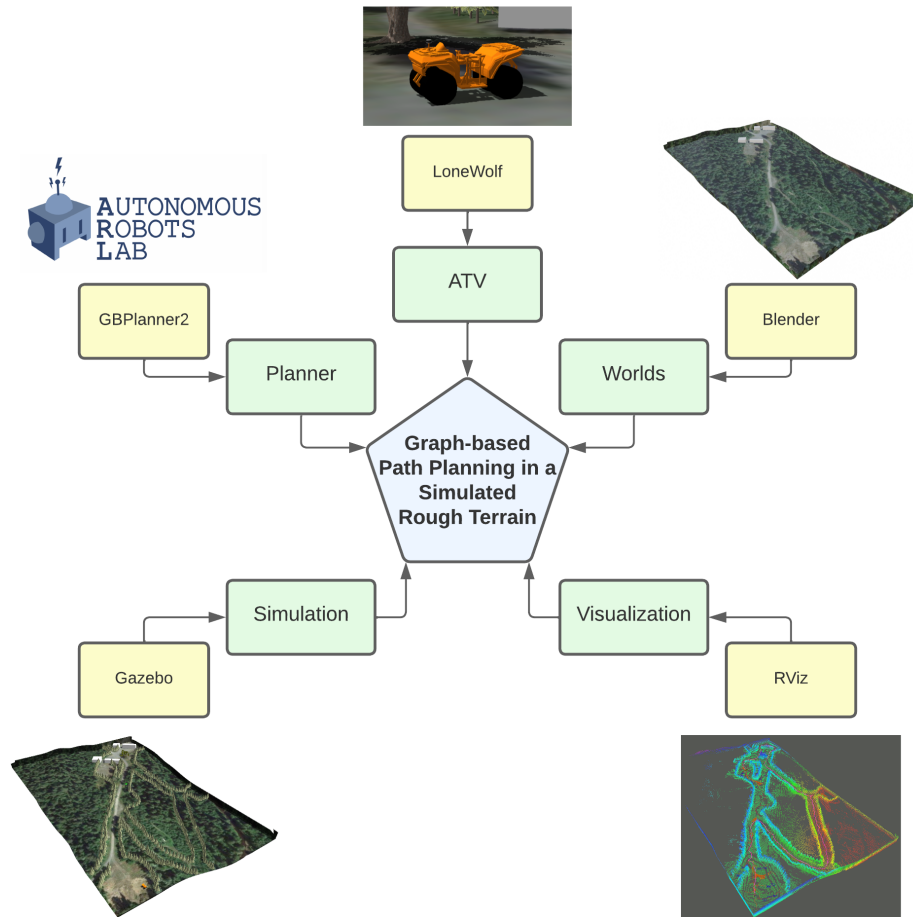


Figure 1.3: General system overview

Figure 1.3 is an illustrative map of the key components this project is built upon. The figure provides some insight into how the different parts are connected, together with a descriptive picture or logo. The key components are as follows:

- **ATV:** A simulated representation of the physical Lone Wolf ATV.
- **Worlds:** Five worlds designed and created using Blender.
- **Visualization:** Displays the planned path and LiDAR-data using RViz.
- **Simulation:** Simulate the ATV and custom worlds using Gazebo.
- **Planner:** Using GBPlanner2 as a system for global path planning and via-point functionality.

1.7 Report Structure

The report is structured into seven chapters including the introduction. This section will briefly describe the contents of each chapter.

Chapter 2 Theory, describes the theoretical background from which the product has been developed. A large part of the theory chapter is dedicated to path planning theory, as well as descriptions of the modeling and simulation tools used, the visualization tool, the Robot Operating System (ROS) and the methods used for mapping the ATV's surroundings.

In chapter 3 Potential Solutions, the two main potential solutions are presented before discussing their respective advantages and disadvantages. This chapter ends with a conclusion and reasoning for continuing with GBPlanner2 as the preferred framework for solving the project assignment.

Chapter 4 Implementation, covers the entire implementation process. This includes integrating the ATV-model into GBPlanner2, modeling of Gazebo worlds, setup of GBPlanner2, implementation of via-point functionality, and finally how to modify the system by adding new worlds and changing the settings.

In chapter 5 Simulation, the finished product will be presented. This chapter will include demonstrations and visualizations of how the path planning functions, how the worlds look, and the procedure for testing in the simulated environment.

Following this, the project will be discussed in chapter 6. The resulting project will be described, along with the known issues that need to be handled in the future. There will be suggestions about further developments on the Lone Wolf project, and the finished product will be put into perspective with the entire Lone Wolf project.

Finally, a short conclusion and summary is made in chapter 7.

Chapter 2

Theory

This chapter covers all the required theoretical aspects necessary to understand the implemented solution to the thesis statement. From the thesis statement in section 1.3, the project can roughly be divided into two separate areas; path planning and simulation. Both these subjects are broad and require a certain level of knowledge in order to properly understand the project at hand. This chapter's goal is therefore to give the reader an introduction to the key elements of the thesis. The theoretical groundwork has been divided into five sections which all cover key elements of the solution.

Section 2.1 introduces general information about the mapping strategy, SLAM, that is used by the mapping library, Voxblox, in GBPlanner2. Then, sections 2.2 and 2.3 will give a detailed introduction to path planning, covering aspects such as what path planning is, how it works, and key differences between path planning algorithms. Section 2.4 will lay the theoretical groundwork for the modeling of worlds by introducing the main software used to create the worlds, namely Blender and Gazebo, sections 2.4.1 and 2.4.2 respectively. Following this, section 2.5 introduces the visualization tool, RViz, used for both mapping and path planning. Finally, ROS will be presented in section 2.6, introducing both ROS 1 and ROS 2, as well as the difference between these.

2.1 Mapping

Mapping in robotics is related to computer vision and cartography. The goal is for a robot to construct a map and localize itself within it by using different sensors, and then represent this sensor data in either a metric or topological map. The differences between these two representation types is that the metric framework considers a 2D space in which it places objects, while the topological framework considers places and relations between them. The latter is most commonly used when dealing with path planning since distances in the map are more precisely calculated. [5]

2.1.1 SLAM

"Simultaneous Localization And Mapping (SLAM) is a method used for autonomous vehicles that lets you build a map and localize your vehicle in that map at the same time" [6]. Mapping of an area can be done by attaching a Light Detection And Ranging (LiDAR) sensor to the vehicle.

LiDAR is a device that uses lasers to scan its surroundings by measuring the time it takes for the reflected laser to return to the receiver [7]. SLAM takes the data collected by the LiDAR and generates a point cloud which depicts the vehicles surroundings. This can be done in both 2D and 3D, whereas the 3D point cloud is mostly used for UAVs and autonomous driving. [6]

2.1.2 Voxel

Voxel is a volumetric mapping library which is integrated tightly with ROS. It is based mainly on Truncated Signed Distance Fields (TSDF), a "3D voxel array representing objects within a volume of space in which each voxel is labeled with the distance to the nearest surface." [8]. A voxel is in 3D similar to what a pixel is in 2D [9]. When using Voxel for planning purposes, the general idea is to have two nodes running simultaneously, one for mapping which processes point cloud data, and one for planning. [10]

2.2 Path Planning

Path planning seems like a simple concept in practice; however, this is not the case for a computer. The issue of optimal path planning is a challenging problem that is still being researched today [11], [12], [13]. Moving the focus away from *optimal* path planning and instead focusing only on path planning, a multitude of algorithms are available depending on the specific requirements at hand.

In this section, a more thorough description of path planning will be provided along with introductions to some of the most relevant algorithms for the current application available today.

2.2.1 What is Path Planning?

Path planning is the geometric problem of finding a collision-free path from a start position to an end position. To acquire this, a representation of the environment is required. The exact representation needed varies based on the approach used to calculate the path [14], however, the representation always needs to include some information of where obstacles are located and where there is traversable space. [15]

Assuming a mathematical representation of the environment, \mathcal{C} , that is divided into two sets; \mathcal{C}_{free} that describes the open feasible area, and $\mathcal{C}_{obstacle}$ that describes the non-feasible area. The path planning problem can be reduced to finding a path, $\mathcal{S}(p)$, where $p \in \mathcal{C}_{free}$ such that moving along the path will take you from p_{start} to p_{goal} . This concept is illustrated in figure 2.1. [15]

In terms of optimization, the most common approach is to find the path that will give the shortest total distance. However there are many other optimization parameters that could be taken into consideration such as turning angle, safety, and smoothest movement. Additional parameters would also increase the complexity level of the path planning problem.

For a machine, the path $\mathcal{S}(p)$ cannot be continuous, and is therefore represented as a set of points. The same principle as in the continuous case still applies.

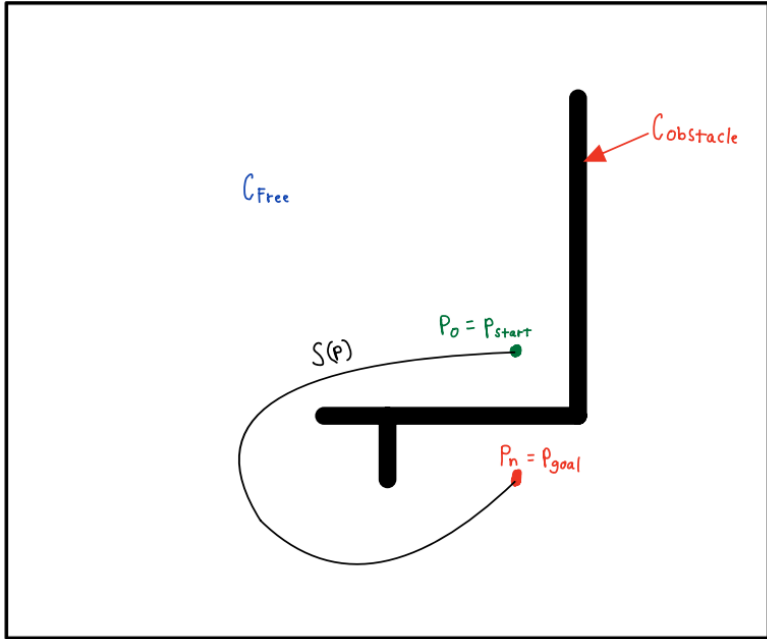


Figure 2.1: Simple continuous path planning visualization

2.2.2 Graphs and Trees

Many path planning algorithms use graphs as their environmental representation. A graph is a collection of vertices and edges. Each edge connects two vertices and implies that it is possible to move between the vertices without colliding with any obstacle or violating any other constraints [15]. In the literature, vertices and nodes are often used interchangeably, however in this thesis vertices are used to avoid confusion with ROS-nodes, see section 2.6.1.

A tree is a graph where each vertex only has one path leading to it. This means that starting from the root vertex of the tree there is only one path in the tree going to each vertex. [15]

Figure 2.2 shows an example of what a graph and a tree could look like. In the figure, the starting vertex is colored dark grey, and the goal vertex is colored green. The figure demonstrates that for the graph there are multiple paths that will lead to the goal vertex, whereas for the tree there is only one possible path.

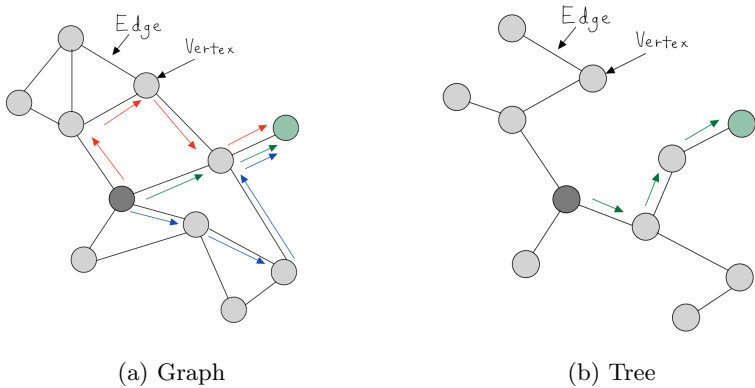


Figure 2.2: Example of a graph and a tree

2.2.3 Local vs. Global Path Planning

To solve the path planning problem, a map of the environment is required. The map needs to include an accurate spatial representation of where objects and obstacles are located, as well as the robots position within the map, see 2.1.1. Based on the known information in this environmental description, path planning can be divided into two methods; local path planning, and global path planning [16]. When planning locally, the robot is considering new information acquired by its sensors when generating the path. The robot is constantly updating its environmental representation while in motion. This means that a robot can update the path, or generate a new one as a response to environmental changes [17]. When planning globally, the environmental map is known prior to generating the path [16]. This means that the robot is considering a static environment, and does not take into consideration any dynamical changes.

2.3 Path Planning Algorithms

Path planning algorithms can be sorted into five different categories; sample-based algorithms, node-based optimal algorithms, mathematical model-based algorithms, bio-inspired algorithms and multifusion-based algorithms. The algorithms are distinguished according to their unique properties, and the approach used to calculate the path [14]. This chapter will include a description of the first two categories. They are amongst the most researched approaches in the field of path planning, and is most relevant for this thesis. The remaining three categories are outside the scope of this thesis, for a description of these, see [14].

2.3.1 Sampling-Based Algorithms

Sample-based algorithms work by sampling the environment as vertices. These algorithms only require an environmental representation of \mathcal{C}_{free} and $\mathcal{C}_{obstacle}$ before sampling vertices in \mathcal{C}_{free} , see section 2.2.1. The algorithms then typically either map \mathcal{C}_{free} by building a graph, or search randomly in order to find a path. As shown in figure 2.3, the sampling-based algorithms can be further divided into two groups; passive and active algorithms. The algorithms that map the environment are classified as passive algorithms, these algorithms find multiple valid paths and require post-processing to find the optimal one. The active algorithms are capable of finding the best feasible path by its own procedure, requiring no post-processing. [14]

In figure 2.3, examples of both passive and active sample-based algorithm can be seen. The *etc.* refers to similar or improved alternatives to the algorithms mentioned within the box.

As seen in figure 2.3, Rapidly-exploring Random Tree (RRT) is an example of an active sample-based algorithm. The algorithm works by randomly creating vertices in the obstacle-free region of the map. It then connects the new vertices to the closest already-existing vertex. This will form a random tree expanding from the starting position and the algorithm will keep expanding the tree until there is a vertex within a predetermined distance to the goal position. The path can then be found by starting at the goal vertex and going backwards until the starting position is reached. Figure 2.4 shows an example of an RRT search procedure. [18], [19]

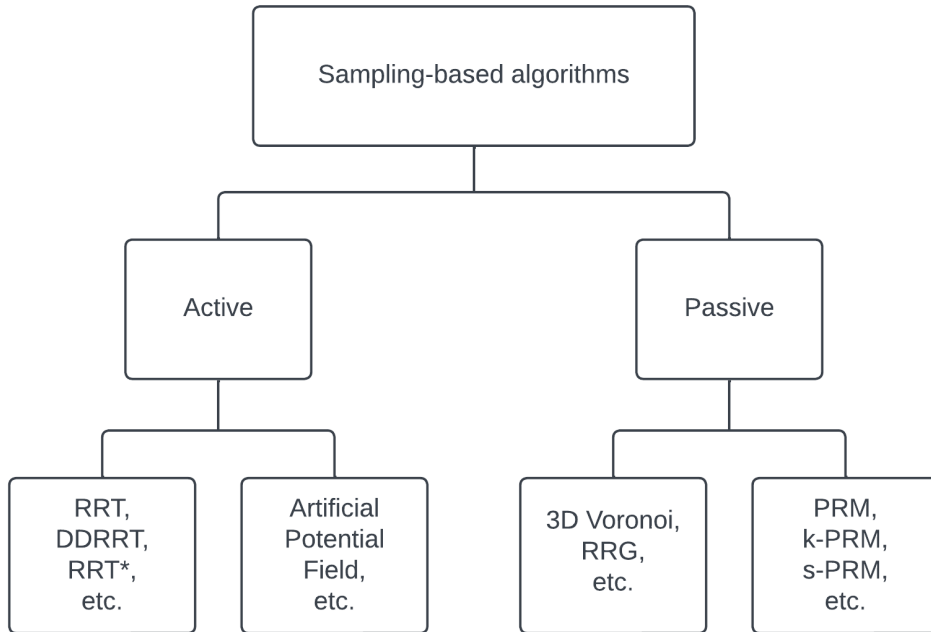


Figure 2.3: Overview of sampling-based algorithms. [14]

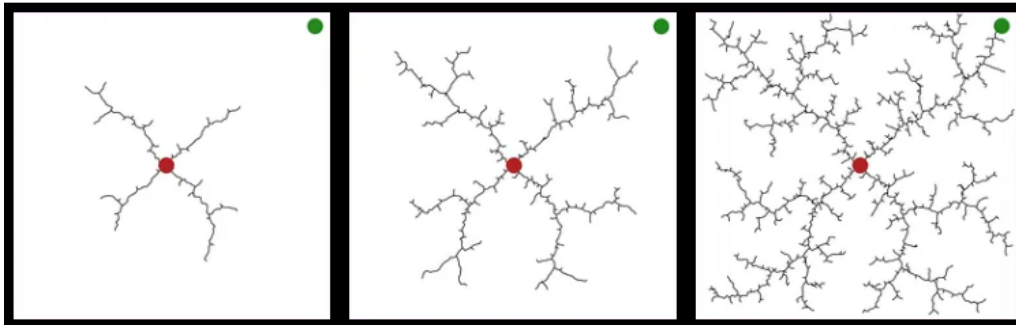


Figure 2.4: RRT searching approach. Inspired by [18], [19].

An example of a passive sample-based algorithm is Rapidly-exploring Random Graph (RRG). This algorithm uses the same principles as RRT but the goal is to expand a graph over the entire obstacle-free region. The algorithm is therefore well-suited for exploration purposes as it creates a graph of feasible paths over the entire space. Unlike RRT, RRG connects the newly sampled vertices to all the nearby vertices within a pre-specified radius of the new vertex which gives a collision-free path. This serves two purposes; firstly, it maps all feasible paths for a robot to traverse. Secondly, it keeps track of this area such that it knows where it has explored and where it has not [20]. The algorithm also works great when used in collaboration with multiple robots to explore an area.

In figure 2.5, an example of an RRG search with multiple robots can be seen. Each color in the figure represent a sub-graph made from an individual search. The figure demonstrates how the sub-graphs are connected to form a combined global graph.

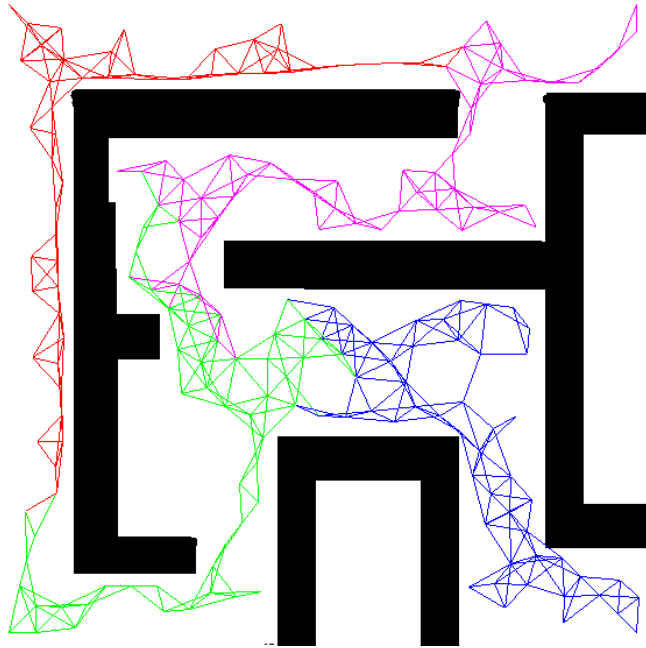


Figure 2.5: Example of RRG search. [20]

2.3.2 Node-Based Optimal Algorithms

Node-based optimal algorithms search through a set of already-established vertices in the environment. This means that it needs a graph or another environmental representation where information about obstacles and openly accessible space is already processed. The algorithms also have in common that they are always able to find an optimal path [14]. These algorithms are great in combination with passive sample-based algorithms as they can be used to find a path in the graph that is built, see section 2.3.

Figure 2.6 shows a few examples of node-based optimal algorithms. The *etc.* refers to similar or improved alternatives to the algorithms mentioned within the box.

Dijkstra's algorithm is an example of a node-based optimal algorithm for finding the shortest path from a start position to an end goal. The algorithm works by examining all nearby vertices starting from the source vertex at the start position, until it reaches the end goal. During each iteration, the algorithm finds the shortest path connecting all vertices to their non-examined neighbor vertices, adding these vertices to be examined next. This results in a shortest-path tree expanding from the source to all examined vertices, and the result is the path going to the goal position. [22]

A* (pronounced "A-star") is an improvement on Dijkstra's algorithm that combines shortest path searching and heuristic searching. Unlike Dijkstra's algorithm, A* takes into consideration the distance from nearby vertices to the end goal when choosing which vertex to examine next. This makes the algorithm much more efficient than Dijkstra's since it searches vertices in the direction of the goal only, instead of expanding equally in all directions. The heuristic distance can also be modified or changed which makes the algorithm more optimizable for specific use cases. [23]

In figure 2.7 an example of a path found using Dijkstra's algorithm and A* in the same environment can be seen. In both figures, the red vertex is the start position and the purple vertex is the goal position. The other colored vertices symbolize how the algorithms have searched to find the

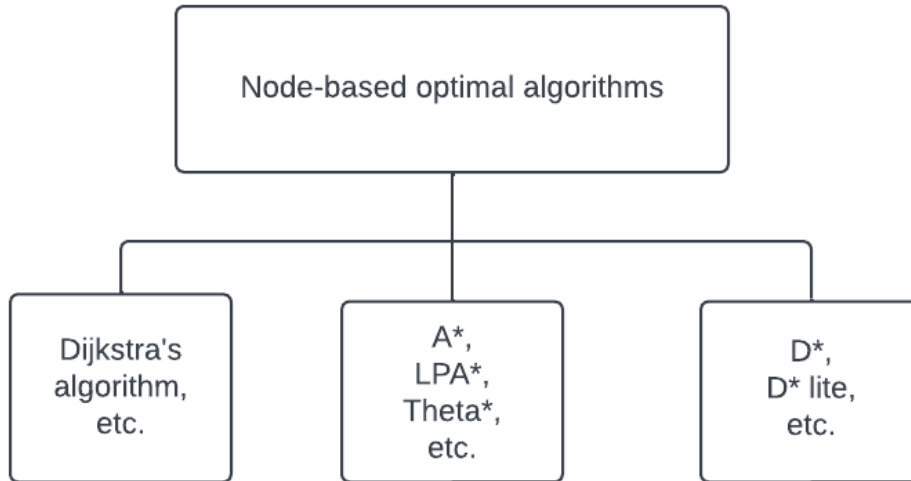


Figure 2.6: Node-based optimal algorithms. [14]

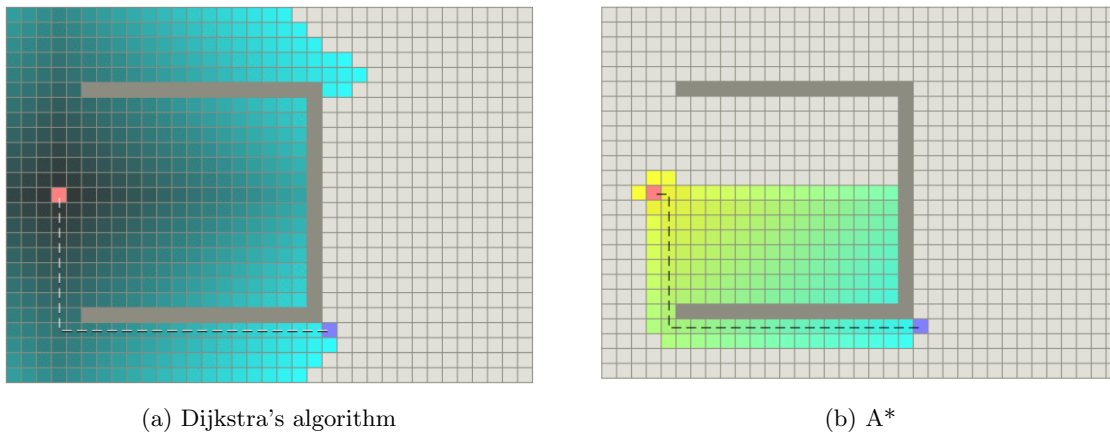


Figure 2.7: Dijkstra's algorithm and A* search example. [21]

solution. In figure 2.7 (a) the shading represents the order the algorithm have searched, going from dark shading to light. In figure 2.7 (b) yellow is used to show vertices that are far from the goal, and blue is used to show vertices that are far from the starting vertex. This then shows the balance A* uses when considering which vertex to examine in the search procedure compared to Dijkstra's algorithm. [21]

2.4 Modeling

Modeling is used in robotics to enable easy testing opportunities without requiring a physical setup for testing. When testing in a virtual environment, having a realistic representation of the environment, robot, and the physics that govern both, is essential. This ensures reliability of the testing, which in turn can be generalized to the real-life use case.

2.4.1 Blender

Blender is a free and open-source 3D graphics software. It is, among other things, suitable for modeling, animation, simulation and game creation [24]. Blender can be used to design and model objects or terrain with several export file-options such as Collada, Wavefront and STL. This allows the user to export a model into a range of different software applications.

2.4.2 Gazebo

”Gazebo is a collection of open source software libraries designed to simplify development of high performance applications” [25]. It is a simulator that is commonly used in robotics for testing and designing robots. It also allows the user to import a 3D model of a world which can be designed further in Gazebo with different static elements such as trees, buildings and cars.

2.5 Visualization

Visualizing the robots behavior when simulating gives the user useful data in an understandable format. This enables the user to make informed assessments about the robot’s progress and how it could be improved [26].

2.5.1 RViz

RViz is a 3D visualization software which visualizes the robots perception of its world using sensor data. RViz stands for ROS Visualization, and its main job is to show the world from the robots point of view. This also means that if there are any issues relating to the encoding of sensor data, the visualization in RViz will reflect these [27]. In contrast, Gazebo will visualize the world as it is, independently of the robot.

2.6 ROS

”Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications” [28]. It is an open-source framework with tools, libraries and capabilities which are crucial for developing applications for robots. ROS has two versions, ROS 1 and ROS 2. Each of these versions of ROS have a list of distributions, also called distros. A distribution is a stable version of ROS where future development of that specific distribution is limited to ”bug fixes and non-breaking improvements” [29].

Certain key components of ROS are common in both ROS 1 and ROS 2. These include, but are not limited to; nodes, topics and services. These components will be presented as they are relevant for the implemented solution.

2.6.1 ROS Components

Nodes are modular units within the ROS framework which have a single, specific purpose. The nodes can communicate by sending or receiving data from other nodes via topics, services, actions and parameters. [30]

Topics facilitate communication between nodes by serving as a channel for messages to pass through. Whenever a node needs to send a message, it publishes the message onto a specific topic, and then any node subscribing to that topic will receive the message. Communication via topics can therefore be one-to-one, one-to-many, many-to-one and many-to-many. [31]

Services are another form of communication between nodes. Whereas topics work using a publisher/subscriber model, services use a call-and-response model [32]. Services thus send data only when called upon to do so. A service call can be made by any number of service clients, however each service will be handled by only one service server.

Figure 2.8 demonstrates how data flows using nodes, topics and services.

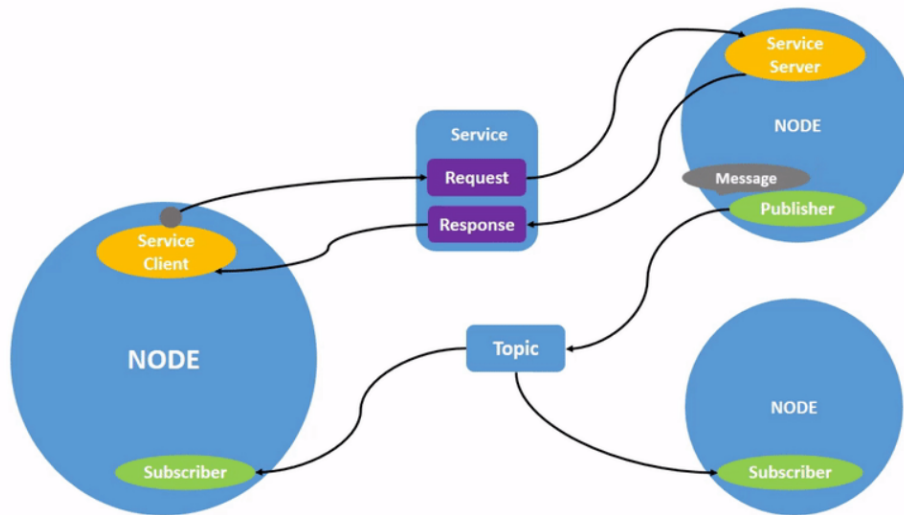


Figure 2.8: System overview for communication between ROS nodes via topics and services. [30]

2.6.2 ROS 1

ROS 1 was released in January 2010 and its latest distribution, ROS 1 Noetic, has an end of life date in May 2025, meaning there will be no more updates after this. ROS 1 uses a Master-Slave architecture with no support for real-time systems. [33]

2.6.3 ROS 2

The latest distribution for ROS 2 is Humble. ROS 2 uses Data Distribution Service (DDS) which provides higher efficiency and reliability, low latency, and scalability. It has a greater support for different languages than just C++ and Python, such as C# and Java. [33]

Chapter 3

Potential Solutions

Two potential solutions were considered as alternatives of how to implement a path planning system onto the ATV. These solutions are based around large, open-source frameworks that are called Navigation 2 (Nav2) and Graph-Based Exploration Planner 2 (GBPlanner2), sections 3.1 and 3.2 respectively. In considering these frameworks, the project assignment from section 1.2 and appendix A, and the thesis statement described in section 1.3 covered the necessary criteria for guiding which solution was best suited to the project's needs. The following sections will describe the structure of both frameworks, as well as the advantages and disadvantages of each. Lastly, a conclusion and reasoning for choosing GBPlanner2 as the preferred solution framework will be presented.

3.1 Navigation 2

Nav2 is the successor of the ROS Navigation Stack [34]. It is a ROS library which allows robots to complete tasks in different types of environments through perception, planning, control, visualization and localization.

3.1.1 Structural Overview

The Nav2 architecture follows the principle of having a server for each important component in the system, see figure 3.1. At the top level of the architecture, the *BT Navigator Server* can be found. This server provides a scalable framework for defining the behavior of the system. In simpler terms, the *BT Navigator Server* controls the behavior of the other servers. The *Planner Server* is responsible of generating a feasible path, and it uses a global path planning approach to do so. The *Controller Server* is responsible for motion planning, and its main task is to make the robot follow a path. It also handles the local path planning. [35]

Each server has numerous plugins to choose from which dictate how the server handles its tasks. Plugins in this context means different approaches to solve the same problem, for example the *Planner Server* has multiple path planning algorithms available that changes the approach used to find a path. This makes the library highly customizable.

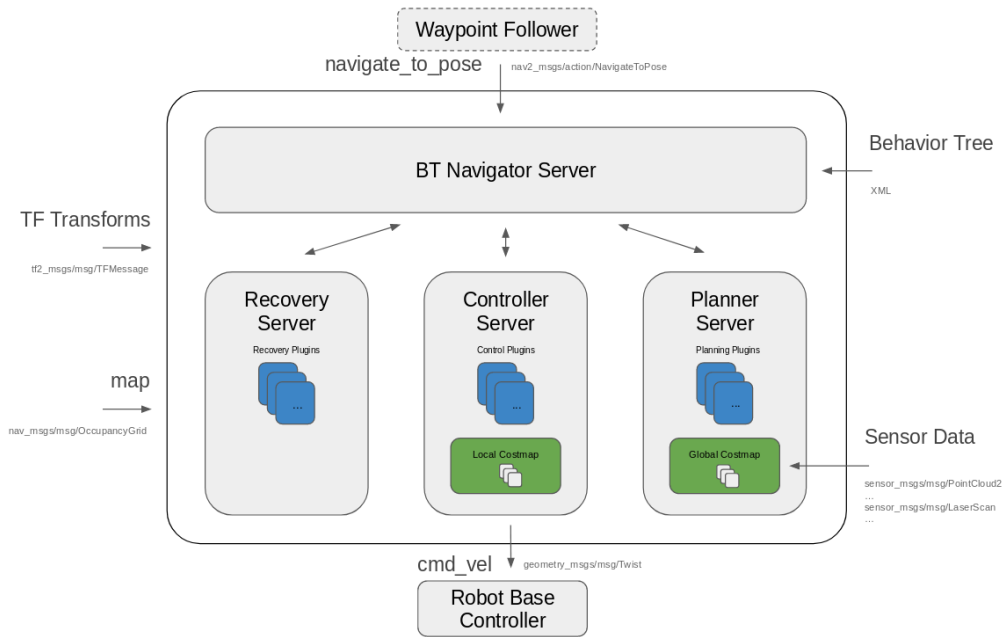


Figure 3.1: Nav2 structure overview [36]

3.1.2 Advantages

Nav2 is a well documented library with a vast amount of information related to it. It has been well tested and provides an easy-to-use example guide when installing [37]. Some of the advantages with Nav2 is the ability to load and store maps, localize the robot in the map, plan a path around objects, and convert sensor data into a costmap representation, see section 3.1.3. [34]

With regards to this project, Nav2 allows for path planning to compute the shortest path, compute a complete coverage path, and compute paths along predefined routes. It also has the ability to change between different path planning algorithms, such as NavFn Planner [38], Theta Star Planner [39], Smac Hybrid-A* Planner, Smac Planner 2D and Smac Lattice Planner [40]. These algorithms have different strengths, meaning some are best suited for legged robots, and some are better suited for wheeled robots [41].

3.1.3 Constraints

Since Nav2 is well documented, it also contains a lot more information than is usually required. This makes the process of finding relevant or specific information on a subject tedious and time-consuming. Nav2 is best suited for flat surfaces with only minor changes in terrain, and is often used for indoor applications [42].

Another issue when dealing with Nav2, is that information regarding the robots environment is represented in a 2D costmap. A costmap is a 2D grid of cells represented by a value from unknown, free or occupied cost. The problem with 2D costmaps is that information regarding the 3D space gets lost because of the loss of one dimension. This will in return not allow for sufficient path planning in terrain-based environments [43]. Figure 3.2 demonstrates what a 2D costmap can look like.

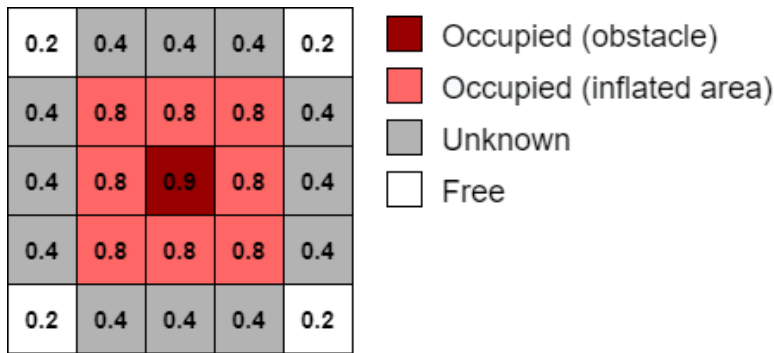


Figure 3.2: 2D Costmap. [44]

3.2 GBPlanner 2

GBPlanner2 is a ROS package developed primarily for autonomous exploration of subterranean environments. The library is developed and maintained by NTNU Autonomous Robots Lab (ARL).

3.2.1 Structural Overview

GBPlanner2 can be divided into important key components relevant to this project, as shown in figure 3.3. The components represent ROS-nodes, see section 2.6.1, each with a unique task. The first component, *GBPlanner Control Panel* is responsible for providing a user interface from which the operator can control the system. The second key component is the *Planner Control Interface (PCI)*. This node controls the entire system at a higher level by communicating with the other nodes and delegating tasks. The next key component is the *Planner*. This is where the path planning happens, and it has the option to do both local and global path planning. Another key component is the *Path Tracker*. This node handles the motion planning and its main purpose is to generate the movement required to follow a given path. The last key component is the *Controller manager* which is responsible for controlling the robot. [45]

3.2.1.1 GBPlanner Control Panel

The *GBPlanner Control Panel* is built as an extension to RViz. It contains the option to initialize five different features; *Start planner*, *Stop planner*, *Go home*, *Initialize* and *Plan to waypoint*. Each feature has its own button within the RViz interface. When one of the buttons is pressed, a request is sent to the *PCI* to request the corresponding feature. [45]

3.2.1.2 Planner Control Interface

The *PCI* can be interpreted as the brain of the system. It keeps track of important system variables and decides what the system should do. One of the key aspects of the *PCI* is the request handler. This is a loop that constantly checks whether a feature has been requested from the control panel. Furthermore, it performs safety checks, such as checking if the robot is currently in a position where the requested feature is possible to execute. The *PCI* operates with a prioritized order, such that the most important features are executed first. [45]

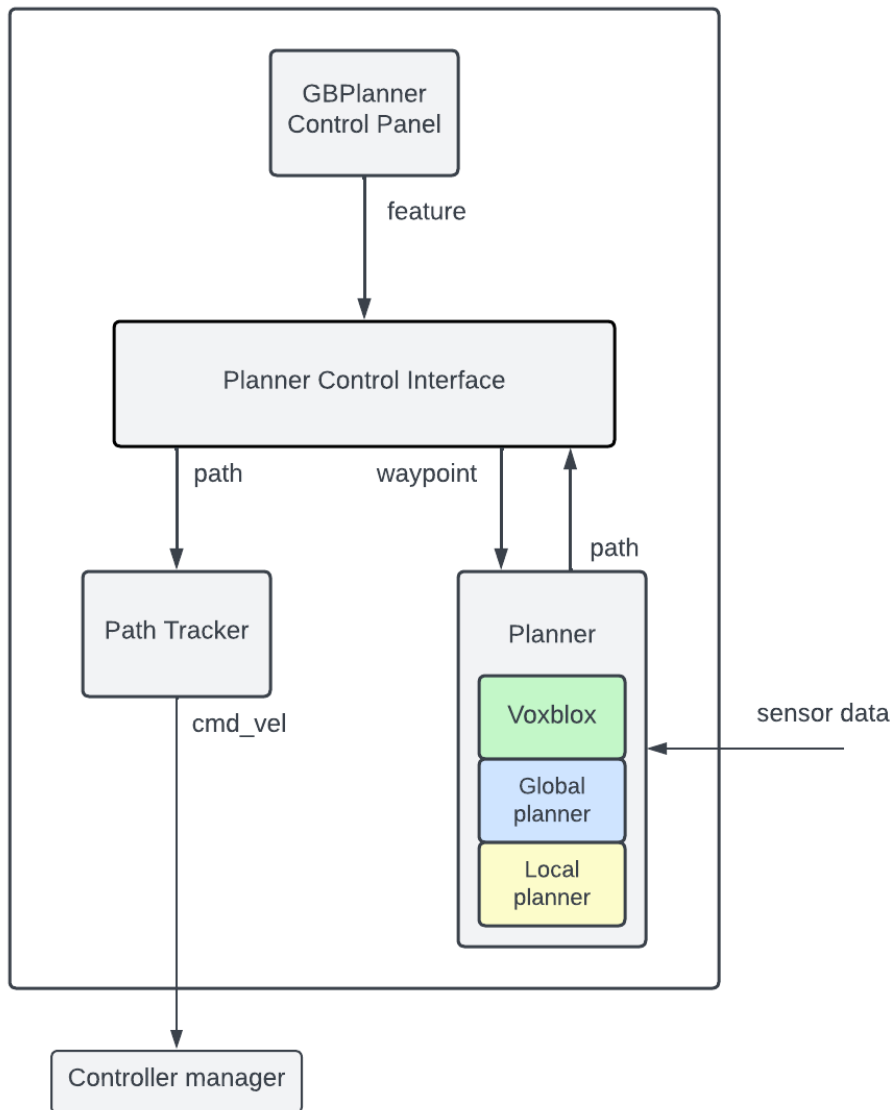


Figure 3.3: GBPlanner2 structure overview

The main communication between the *PCI* and *Planner* is when the *PCI* requests a path. It then specifies if the local or global planner is to be used. The local planner is used when the robot is in exploration mode, which is initialized with the *Start planner* button. The global planner is used when *Go home* or *Plan to waypoint* is initialized. If *Plan to waypoint* is chosen, a waypoint must also be provided. The waypoint is set in the RViz interface. *Go home* will generate a path to the starting position of the robot. [45]

3.2.1.3 Planner

The *Planner* node receives sensor data, and maps the environment using Voxblox, see section 2.1.2. By using Voxblox, the robot's surroundings are mapped by marking occupied voxels. In this way the *Planner* knows that non-marked voxels are traversable, and marked voxels are considered as obstacles [45]. As mentioned, the *Planner* node can either plan using a local or a global approach.

The local path planner is an explorational planner and uses a combination of RRG, see section 2.3.1, and Dijkstra’s algorithm, see section 2.3.2. It samples random vertices in the open region of the map within a specified radius around the robots current position. It then uses RRG to build a local graph connecting the vertices. After this, it finds the vertex which gives the robot the highest explorational gain. It uses Dijkstra’s algorithm to search through the graph and find the shortest path to get to that vertex. When the robot reaches this vertex it repeats the procedure, expanding the global graph by adding the recently explored local graph. [46]

When planning globally, the *Planner* considers the global graph, and uses Dijkstra’s algorithm to find the shortest path to the goal position. [46]

3.2.2 Advantages

GBPlanner2 is a well documented library. The developers have produced two articles that describe the work and theory behind the library [46], [47], and the open-source code is provided with great documentation on Github [45]. The library is relatively small compared to Nav2. This makes it easier to find relevant information, and to understand how the code works.

The library is mainly developed for exploration in subterranean environments. This means that it is designed to work in uneven terrain with elevation differences. It uses the 3D environmental representation tool, Voxblox, to map the environment, which is more suitable for a terrain-based environment than other 2D approaches. The library also comes with a built-in simulation of a vehicle that is relatively similar to an ATV, called smb. GBPlanner2 provides motion planning and low-level control for this vehicle, which is transferable to the Lone Wolf ATV.

3.2.3 Constraints

Some of the constraints of GBPlanner2 is that there are fewer customization options available compared to other bigger libraries such as Nav2. There is only one path planning algorithm provided, and in order to use other algorithms they would have to be implemented. It is also developed using ROS 1, which is not optimal with respect to the Lone Wolf project, as it uses ROS 2. The developers are planning on migrating over to ROS 2, but is currently limited by important dependencies such as Voxblox still being available in ROS 1 only.

3.3 Conclusion

Nav2 was the initial choice of path planning framework, as it contained many of the desired functionalities relevant for this project. This included the vast amount of information available about the library, the amount of customization options available, and its compatibility with ROS 2. However, as the main objective of this thesis was to implement a framework for path planning in a terrain-based environment, Nav2 had to be discarded as a potential solution because of its lack of support for 3D costmaps.

GBPlanner2 as an alternative, is a relatively small and new library. It also contains many of the desired functionalities, however it is less customizable, and as it uses ROS 1, ROS Bridge will

possibly be required in order to combine it with the Lone Wolf project, see section 6.2. GBPlanner2 uses Voxblox, as mentioned in section 3.2.2, which enables mapping and path planning of the 3D environment. This was the main reason for why GBPlanner2 was chosen as the framework to be implemented.

Chapter 4

Implementation

This chapter covers the purpose and approach for implementing the ATV in different worlds with various terrain. The goal of the implementation is to simulate how the ATV, together with a path planning system, handles different terrain-based environments. It is also desirable to visualize the simulation in a Graphical User Interface (GUI). The full code covering the implemented solutions for this thesis can be found on Github [48].

4.1 Background

In order to create, simulate and visualize the worlds the ATV traverses, different types of software were utilized. Blender was used for the design and modeling of the different worlds. Gazebo was used for tasks involving simulation of the ATV, and RViz was used for visualizing the point cloud created by the vehicle's onboard LiDAR. This lays the foundation for implementing a path planner for a terrain-based environment.

4.2 ATV

The design and affiliated files of the ATV used in this project is a true representation of the actual vehicle. This model comes from a previous bachelor's thesis written by students at the Norwegian University of Science and Technology (NTNU) in 2022 [49].

A copy of the smb-vehicle's, see section 3.2.2, simulation files were copied, and the ATV was then implemented by replacing the relevant files. This included .stl files of the ATV body and wheels, as well as the .urdf files of the ATV and wheel descriptions. Furthermore, the launch files of the smb vehicle were modified to launch the ATV.

4.3 Modeling

For testing the ATV in various terrain and environments, different worlds to simulate the vehicle in are needed. A world in this context is an enclosed area where nothing exists outside of what

can be seen. There have been previous attempts to create terrain-based worlds in Gazebo for the Lone Wolf project, however this was unsuccessful. In addition to a world that closely resembles the actual test-site for the physical ATV at Heistadmoen, Kongsberg, four other worlds were created; `maze_world`, `simple_world`, `normal_world` and `complex_world`. The five worlds were designed based on a set of criteria specifying their complexity, see figure 4.1.

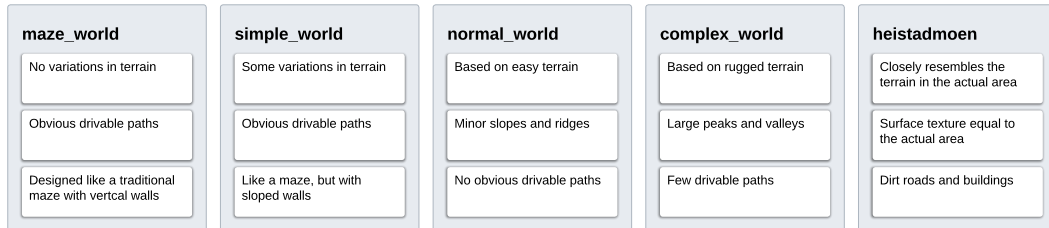


Figure 4.1: Criteria for each world.

Blender and Gazebo was used in order to create the different worlds so that the ATV had an environment to drive and explore in. Blender was used for the 3D modeling and design of the different worlds, while Gazebo was used as the simulator in which the ATV could drive within these worlds.

4.3.1 Blender

When researching which modeling software to be used for creating the different worlds, several options appeared sufficient for the task. Software such as Blender, Unity and Fusion 360, all have the capability to design terrain-based environments. Gazebo uses Blender as an example-tool for creating worlds, as it uses the preferred file format, Collada. Collada is a file format which allows the user to export 3D models between different graphics software [50]. For this reason, Blender was a natural choice for as the modeling software. A world could also be created directly in Gazebo, however, it does not allow for creation of varying terrain and landscape.

For designing `maze_world`, `simple_world`, `normal_world` and `complex_world`, a plane mesh was created and then edited according to the set of criteria shown in figure 4.1, which was decided prior to modeling the worlds. The editing was done using different built-in tools in Blender for manipulating the surface to resemble different types of terrain. A picture containing mountain-like texture was then added onto the edited surface to give it a natural look. For each world, a vertical wall was added at the edges so that the ATV was unable to escape the world.

Since Heistadmoen is a real-life location, it was preferable to create this world as close to the real topography as possible. Blender has an add-on called Blender Geographic Information System (BlenderGIS) which allows the user to create a model based on satellite geodata from all areas on earth [51]. When using this add-on, there was a limit to the zoom-functionality for the desired location, and the generated world ended up being a lot bigger than needed. This resulted in poor computer performance, as well as a lack of details in the terrain. Instead of using BlenderGIS, a satellite picture of the test-site at Heistadmoen was downloaded and added to the plane in Blender. With help from topography maps of the area, the design of the world was done manually using the same procedure as the four other worlds. This included four buildings, a dirt road, uneven

surfaces and general height-differences in the terrain.

4.3.2 Gazebo

Gazebo was used as the simulator for combining the model of the ATV and the different worlds created in Blender. After each world was exported from Blender and imported into Gazebo, the model of the world was made static in order for it to not move while simulating. This is because the model is affected by gravity and external forces by default. Gazebo also has the ability to include built-in static models of trees, cars and buildings. `heistadmoen` was the only world where this was utilized, as the real-life location consists mainly of a forest, with a few buildings and a dirt road with some minor trails leading down to a creek.

4.3.3 Simulated Worlds

`maze_world`, as can be seen in figure 4.2, consists of a flat surface with vertical walls, making it a traditional maze. The purpose of this world is general testing of how the ATV drives, with close to zero room for disturbances to the calculated path or drivable path. The world has a clear distinction between what is a drivable path, and what is not.



Figure 4.2: `maze_world` in Blender.

`simple_world`, seen in figure 4.3, resembles a conventional maze, except this world has an uneven surface with slopes connected to the walls. This adds a layer of complexity for the ATV, but the choice of a feasible path is still clear.

`normal_world`, as can be seen in figure 4.4, no longer contains an obvious choice of a correct path. This world consists of a simple terrain environment with varying differences in height. Almost all areas of the world are drivable, however, there are sections of the terrain with passages that are more easily traversable for the ATV.

`complex_world`, seen in figure 4.5, is a stress-test environment with considerable variations in the terrain. This includes large peaks, rugged surfaces and steep valleys. The world does not contain



Figure 4.3: `simple_world` in Blender.



Figure 4.4: `normal_world` in Blender.

many drivable paths, but will in turn test how the ATV handles extreme terrain.

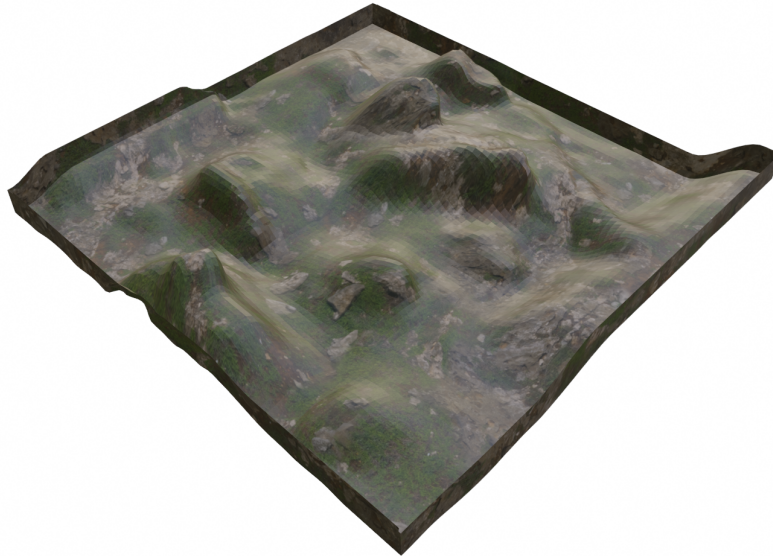


Figure 4.5: `complex_world` in Blender.

Heistadmoen is a location south of Kongsberg and is where the physical ATV is tested. The simulated world of `heistadmoen`, as can be seen in figure 4.6, is similar to the actual terrain in that area, with mostly flat surfaces, a dirt road and some buildings. The purpose of this world is to simulate how the physical vehicle operates in an environment resembling the actual test-site. The buildings were created using simple boxes, while the dirt road is modelled to be somewhat flatter than the adjacent terrain.



Figure 4.6: `heistadmoen` in Blender.

Since `heistadmoen` is the main world in which the simulation will be run, three illustrations showcasing what the world looks like in Gazebo and RViz are shown in figures 4.7, 4.8 and 4.9. The use of point cloud-visualization is explained in section 4.6.4. The real-life location, besides the dirt road, buildings and trails, consist mainly of a forest. Adding such a large amount of trees would require an unnecessary amount of computer power, it was therefore decided to implement the trees adjacent to the dirt road and trails only. This restricts the ATV from going off-road, without sacrificing computer performance.



Figure 4.7: heistadmoen in Gazebo.

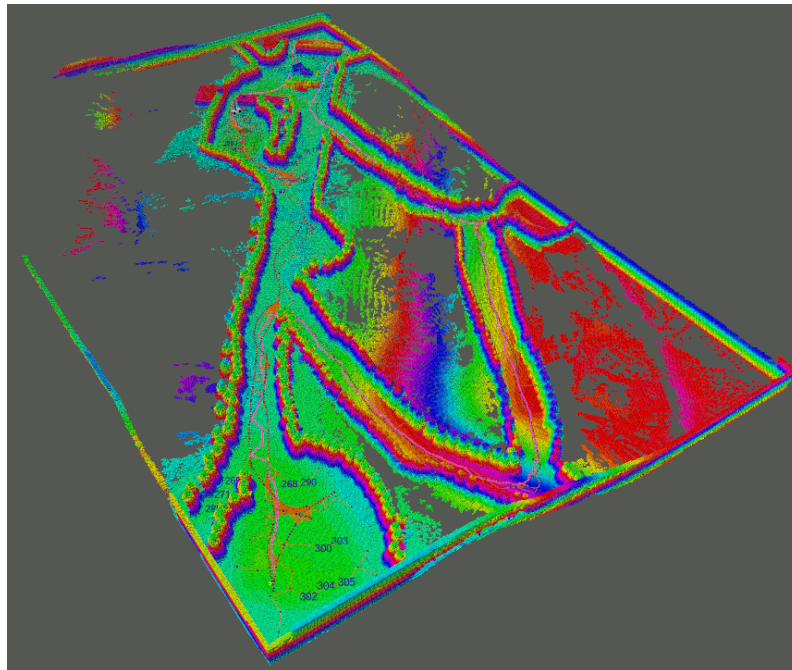


Figure 4.8: heistadmoen in RViz with Voxblox-visualization.

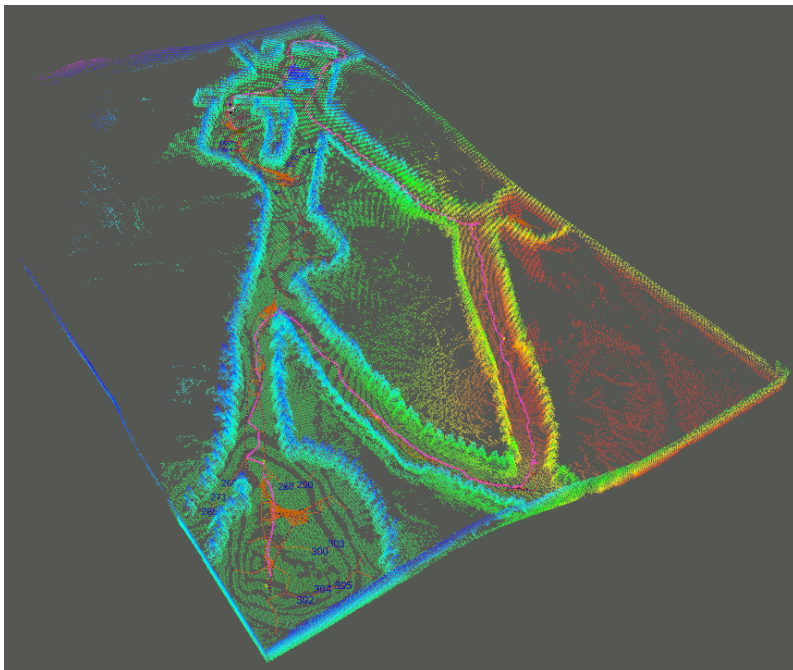


Figure 4.9: heistadmoen in RViz with PointCloud2-visualization.

4.3.4 Work Flow

The process of going from an idea of a world to an actual environment where the ATV can drive consisted of several steps. Before the design process started, a set of criteria describing the appearance and purpose was decided for each world. As shown in figure 4.1, this included the type of terrain that should be present, size and obstacles, as well as how many possible drivable sections the world should contain. As each world had a set of criteria describing it, the five worlds were designed in Blender according to the criteria, with some minor adjustments to create feasible paths throughout the terrain.

The five worlds were then exported from Blender as Collada-files and imported into Gazebo. As mentioned in section 4.3.3, *heistadmoen* was designed further with several trees being implemented in order to represent the real-life location better. The final step was to save each world as a world-file. "The world description file contains all the elements in a simulation, including robots, lights, sensors, and static objects." [52]. This was done in order to use it in a launch-file when starting the simulator. The entire modeling workflow is illustrated in figure 4.10.

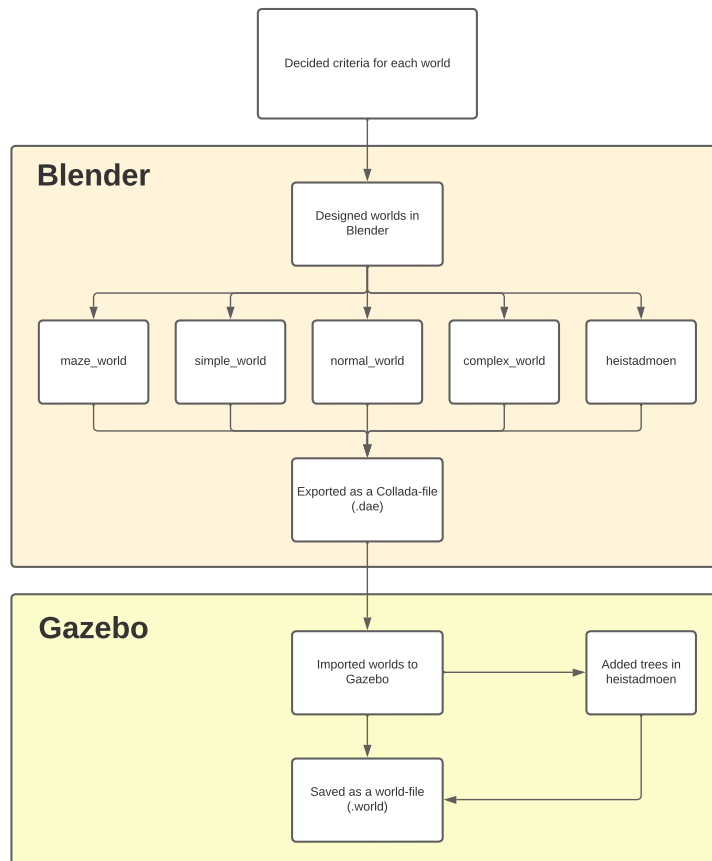


Figure 4.10: Work flow of how the different worlds were created.

4.4 Via-Point Implementation

This section will give a detailed description of how the via-point functionality was implemented, as well as any necessary modifications done to GBPlanner2's original code for it to work.

4.4.1 Selecting Via-Points

In order to select the via-points, the *Publish Point* feature in RViz was used. This was set up to publish the points on the topic `clicked_point`, and a callback function was made in the *PCI* node, see section 3.2.1.2. This callback function was called each time a new point was published on the topic. The purpose of the callback function is to add each new point passing through into a global list which is accessible to the entire *PCI* node. In order to visualize the selected points in RViz, a new visualization topic was created. On this topic, the *PCI* node published all the stored points each time a new point was passed through the `clicked_point` callback function. The topic was configured to visualize the points as yellow numbers in ascending order, as can be seen in figure 4.11.

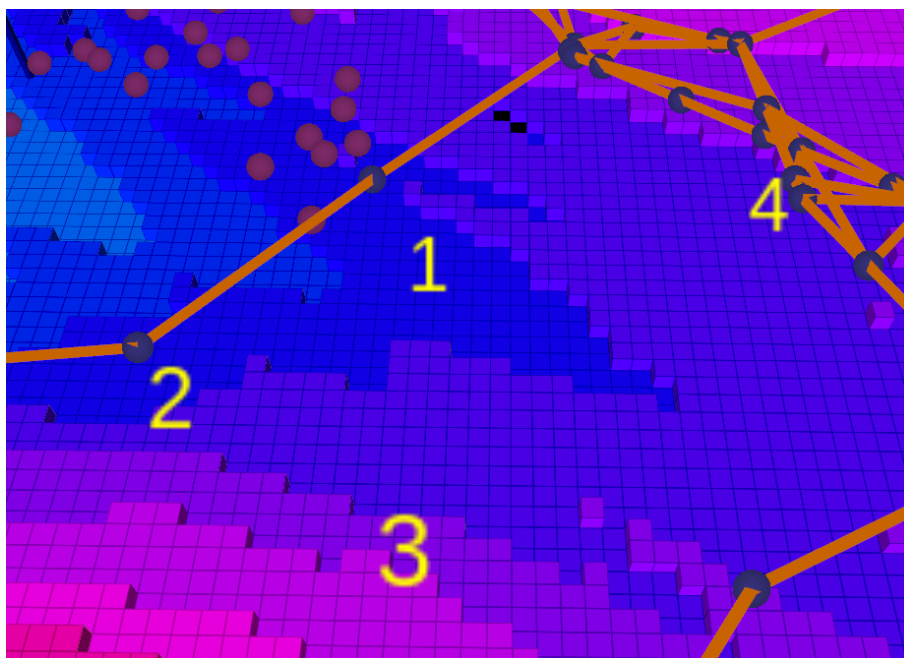


Figure 4.11: Visualizing via-points with numbers in ascending order

4.4.2 GbPlanner Control Panel

In order to initialize the path planner, a new button, *Plan using Viapoints*, was added in the RViz *GbPlanner Control* panel, see figure 4.12. As mentioned in section 3.2.1 the *GbPlanner Control* panel is built as an extension to RViz, by using the `rviz::panel` class to create a panel object [53]. The new button was added by creating a `QPushButton` object and adding it to the panel object, specifying the name and giving it a callback function to be executed when the button is clicked. The callback function sends a service call to the *PCI* node requesting to make a path through all selected via-points and executes the path.

Some of the buttons in the control panel were also renamed to better describe their action after the new feature was added. This includes *Start Exploration* and *Stop*, which was originally called *Start Planner* and *Stop Planner* respectively.

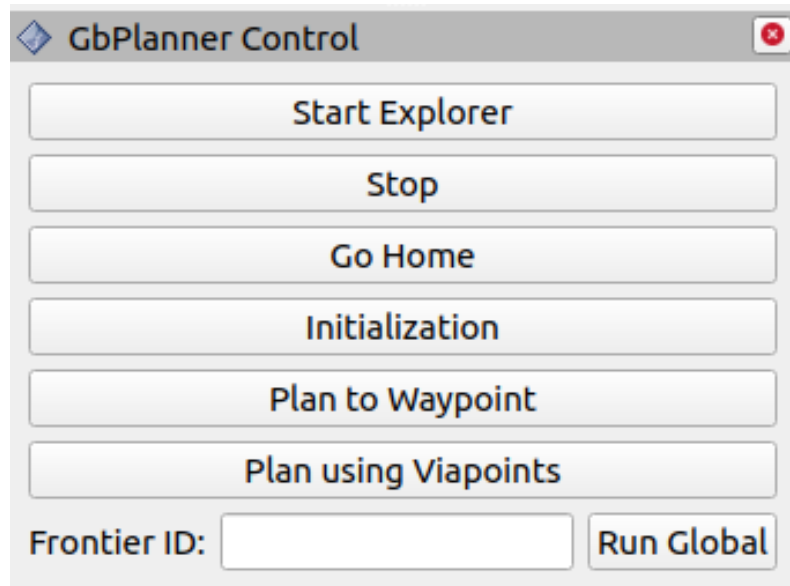


Figure 4.12: *GbPlanner Control* panel [45]

4.4.3 Planner Control Interface

The *Plan using Viapoints* callback request was integrated into the *PCI* node's original request handler at the bottom of the priority list. This was done to ensure that all previous features and safety measures in *GBPlanner2* would work as expected with the same order of priority as before. When the *PCI* node is ready to handle the *Plan using Viapoints* callback request from the *GbPlanner Control* panel, it first requires a path from the *Planner* node, and then sends it to the *Path Tracker* node to be executed.

For the *PCI* node to get the path from the planner, a new service communication was set up between the nodes, see section 2.6.1. The new service type was created to send a list of points, instead of just one, which was the approach used in the original *Plan to Waypoint* feature. Note that *waypoint* is the word used by *GBPlanner2* and means the same as point, which is how its referred to in this thesis.

As mentioned in section 4.4.1 the *PCI* node keeps track of all the added points through the *RViz* interface. When the *Plan using Viapoints* feature is executed, the list of points is cleared such that a new set of points can be specified later. The *Stop* feature was also modified to clear the point list, such that this feature can be used to delete the selected points.

4.4.4 Global Planner

Calculation of the path happens in the *Planner* node. The planning was implemented using *GBPlanner2*'s global planner to calculate a path from one point to the next based on a list of points, and then combining the segments into a total path. The original global planner was only capable

of calculating a path from the current position to a selected point. In order to calculate from point to point, the global planner was modified to instead take in two separate points. The algorithm below shows an in-depth description of how the via-point path generation was implemented.

Algorithm 1 Calculate path using via-points

Require: *waypoints* ▷ A list of points

- 1: $p_0 \leftarrow \text{getCurrentPosition}()$
- 2: $\text{path} \leftarrow \text{getGlobalPathViaPoints}(p_0, p_1)$ ▷ From current position to first point
- 3: **if** $\text{waypoints} \geq 2$ **then** ▷ More than two points
- 4: **for all** p in waypoints **do**
- 5: $\text{tempPath} \leftarrow \text{getGlobalPathViaPoints}(p_{-1}, p)$ ▷ From last point to current point
- 6: $\text{path} \leftarrow \text{combinePaths}(\text{tempPath})$ ▷ Extend path by adding tempPath
- 7: **end for**
- 8: **end if**
- 9: **return** path

4.4.5 Via-Point Overview

Figure 4.13 shows an overview of the GBPlanner2 system, specifically for the via-point feature. In the figure, green is used to symbolize implemented components, modifications to the original code is marked in yellow, and grey is used where the original code was not altered.

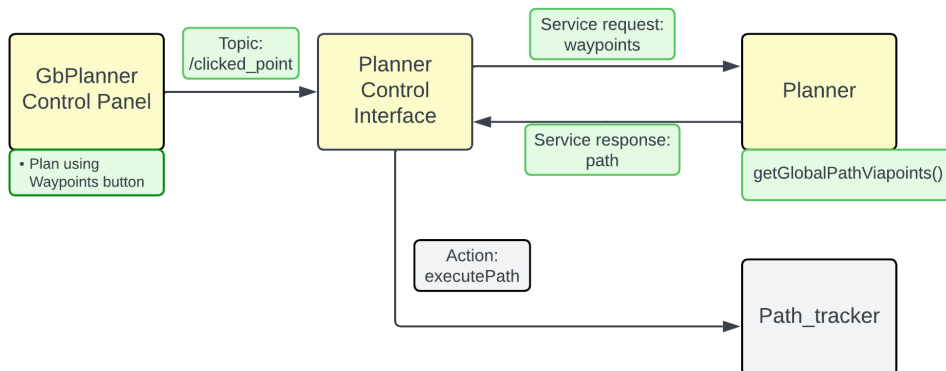


Figure 4.13: Via-point communication.

4.5 System Overview

The system as a whole can be divided into three main parts; Simulation, Planner, and GUI. Simulation consists of two separate models that are spawned in Gazebo. These two models are the ATV and the simulated world. The second part, Planner, is the ROS package GBPlanner2, which takes care of all things relevant to the planning and autonomous driving of the ATV in the simulated world. Lastly, the GUI takes the sensor data collected from the ATV and presents it together with path calculations. This is also where the user controls the vehicle by using the control panel. Figure 4.14 is a rough representation of how the main parts of the system are connected.

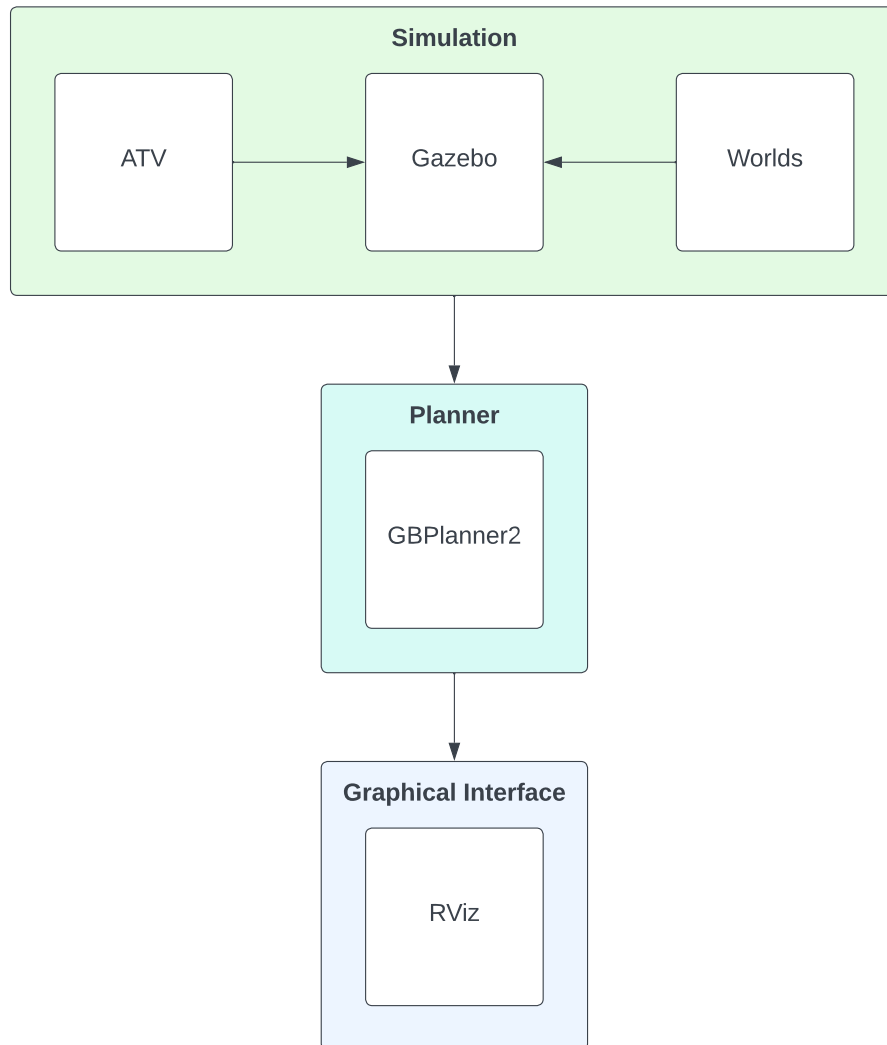


Figure 4.14: A general system overview of the complete system

4.6 Modifications

The purpose of simulating a system is to test real-world scenarios in a safe environment. These real-world scenarios may change over time, either because the robot is upgraded or replaced by a new one, or because the environment it operates in requires different specifications. For that reason, it is beneficial to be able to change elements of the simulation in order to simulate a system as close to reality as possible.

4.6.1 Adding a New World

A new world can be designed either through a modeling software such as Blender, or directly in Gazebo, see section 4.3. The advantage of using a modeling software is that, as mentioned in section 4.3.1, the user can create different types of terrain. If the new world is designed outside

of Gazebo it needs to be exported from that software in a viable file format, for instance STL, Collada or OBJ, with Collada and OBJ being the preferred formats [54]. The model then needs to be imported into Gazebo and saved as a world-file. This file needs to be placed in the *worlds*-folder, while the exported file must be placed in the *models*-folder, see listing 4.1. In order to launch the new world in the simulator, simply write the file-name in the terminal when launching the simulator, as shown in listing 4.2.

```
1 <path_to_lonewolf>/src/sim/lonewolf_sim/atv_gazebo
```

Listing 4.1: Where to find the *worlds*- and *models*-folder.

```
1 roslaunch lonewolf_sim gbplanner.launch world:="new_world.world"
```

Listing 4.2: How to launch a new world through the terminal.

4.6.2 Change Settings in the Current World

The world-model can be edited by either opening it directly in Gazebo, or by editing it in Gazebo after launching the simulator. In Gazebo there are, as mentioned in 4.3.2, several objects that can be added, including trees, cars, robots and buildings. Select the *Insert* tab in the top left corner to access the model database and choose an object. The position and orientation, pose, of each object can be changed through the translate and rotate tool, as well as its position by dragging the object to a desired location [55]. See figure 4.15.

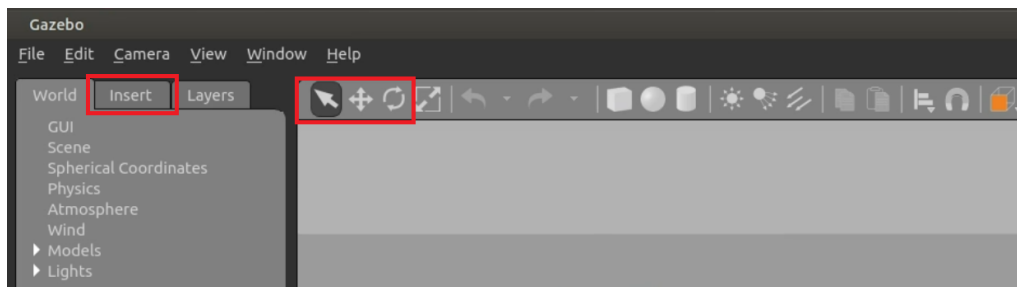


Figure 4.15: Gazebo user interface with *Insert* and move, translate, and rotate buttons highlighted.

The spawn-point of the ATV might differ in some of the custom worlds, meaning that the height where the ATV spawns depends on the terrain-height of the spawn-point in the specific world. In order to prevent the ATV from falling through the world, the height from which the vehicle spawns can be adjusted in the *gbplanner.launch* file, as shown in listing 4.3.

```
1 <arg name="z" value="2.0"/>
```

Listing 4.3: How to change spawn height in *gbplanner.launch*.

4.6.3 Changing Maximum Inclination

As mentioned in section 4.3.1, each world has a vertical wall at the edges so that the ATV does not fall of. This barrier, together with trees and buildings, works as obstacles in which the vehicle detects non-feasible paths. The planner also allows for a maximum inclination for which the ATV can explore and plan a path. This means that the amount of feasible paths in a steep terrain will

be fewer compared to a flatter terrain. As shown in listing 4.4, the max inclination is set to 25 degrees.

```
1 max_inclination: rad(25.0*pi/180)
```

Listing 4.4: How to change maximum inclination in gbplanner_config.yaml.

4.6.4 Changing Preferences in RViz

The visualization in RViz consists of several topics which, among other things, illustrate the mapping done by the ATV, viable paths, and the current path. It is preset to show Voxelbox, see section 2.1.2, but it can also show a typical point cloud. First, add a new topic by clicking *Add*. Second, choose *By Topic* in the pop-up, and third, click *PointCloud2* under */surface_pointcloud* and accept the new topic by clicking *OK*. This process is shown in figure 4.16. In order to hide the Voxelbox-visualization, simply uncheck *Voxelbox* in the RViz GUI on the left-hand side.

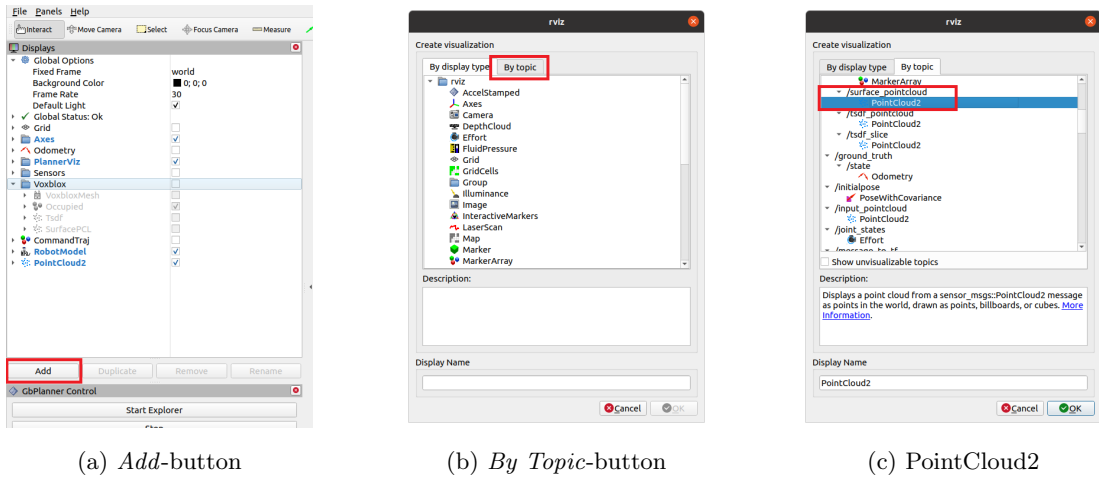


Figure 4.16: Adding point cloud-visualization

Figure 4.17 and 4.18 showcase the differences between Voxelbox and a standard point cloud. Note that the point cloud can be adjusted directly in RViz with different colors, as well as changing size and shape of the points.

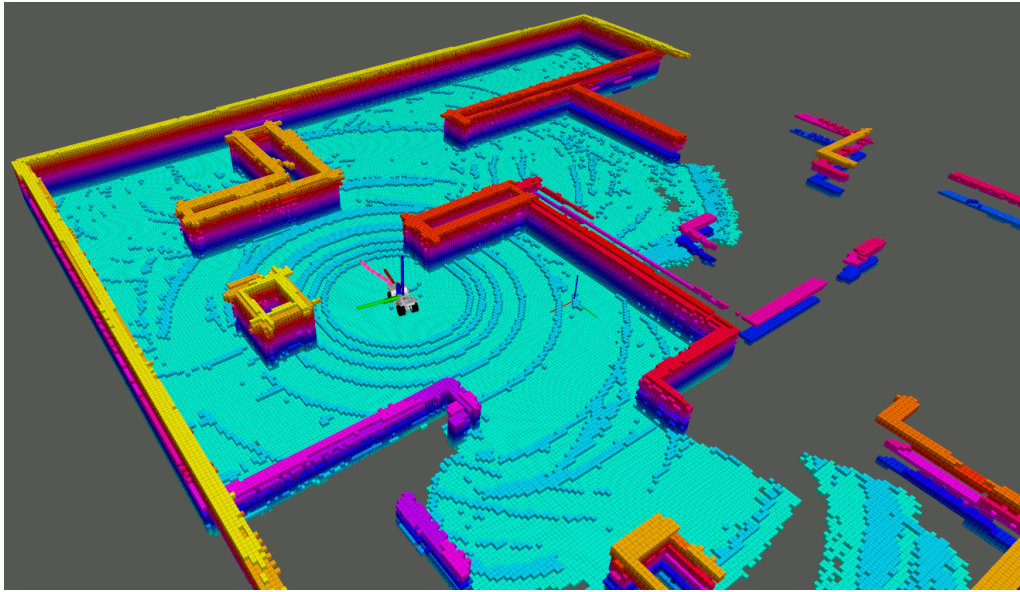


Figure 4.17: Voxblox visualization in RViz

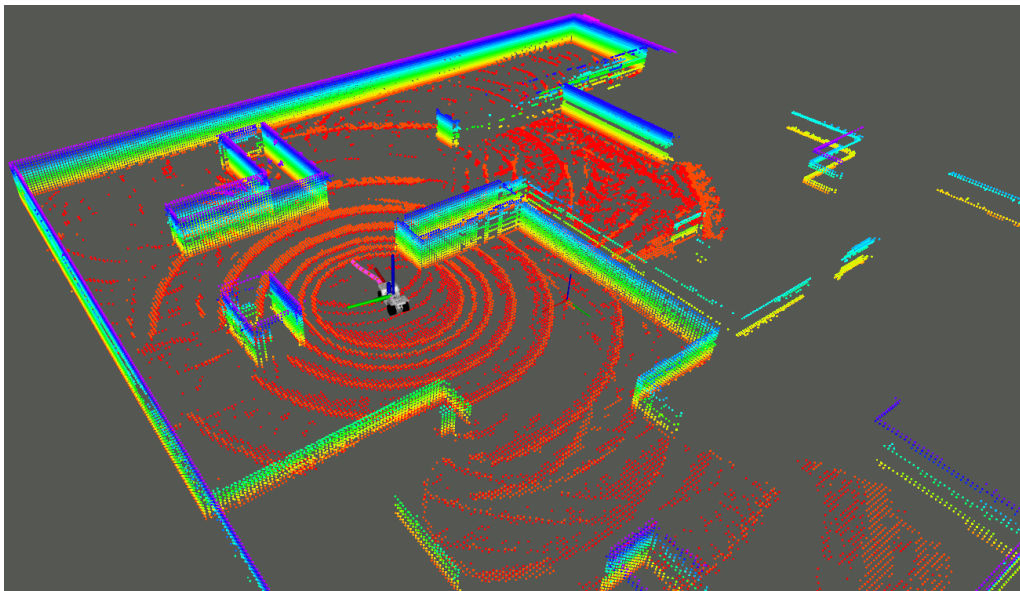


Figure 4.18: PointCloud2 visualization in RViz

Chapter 5

Simulation

This chapter covers the process of how to launch the simulator with the ATV and a custom world. The chapter also covers how the path planning system performs for several of the worlds, together with illustrations from RViz showing the created paths using via-points.

5.1 Launching the Simulator

The simulator is launched by entering the command in listing 5.1 into the terminal. This command can be run with or without specifying a world. If no world is specified, `maze_world` is the default world.

```
1 roslaunch lonewolf_sim gbplanner.launch
```

Listing 5.1: How to launch the simulator in the terminal.

If a specific world is to be launched, an extended command must be entered. The command in listing 5.2 launches `heistadmoen`.

```
1 roslaunch lonewolf_sim gbplanner.launch world:="heistadmoen.world"
```

Listing 5.2: How to launch the simulator with a specific world.

5.1.1 How to use the Path Planning System in RViz

After the simulator has been launched, the ATV has to be initialized. This is done in RViz by clicking the *Initialization*-button. The ATV is now ready to explore its surroundings by clicking *Start Explorer*. See figure 4.12 for the control panel.

In order to plan a path without the use of via-points, click the button labeled *2D Nav Goal* and select a point in the world after it has been explored. Then, click *Plan to Waypoint*. The planner should now create a path, visualized in pink.

When using via-points, the *Publish Point*-button must be clicked. The next step is to click somewhere on the explored map, and a yellow number will show up. In order to select a new via-point,

simply click the *Publish Point*-button again. When the desired amount of via-points are placed, click on the button labeled *Plan using Via-points* in the control panel. Figure 5.1 shows the *2D Nav Goal* and *Publish Point*-buttons.

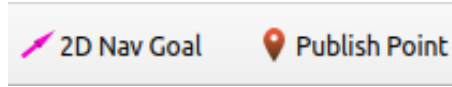


Figure 5.1: *2D Nav Goal* and *Publish Point*-button

5.2 Test Procedure

The purpose of simulating the different custom worlds is to test and gather information on how the ATV handles different terrain-based environments. Since each world has a different terrain design, each world also has a different level of complexity, which in turn allows for a variety of tests when simulating the ATV with its path planning system. The testing is done by using an XBOX-controller to drive the ATV around the obstacles in each world. This is done in order for the ATV to map out the world in the shortest amount of time so that the created paths can be sufficiently visible. Steering the ATV with an XBOX-controller is done via the *teleop_twist_joy* ROS package [56], which is included in GBPlanner2.

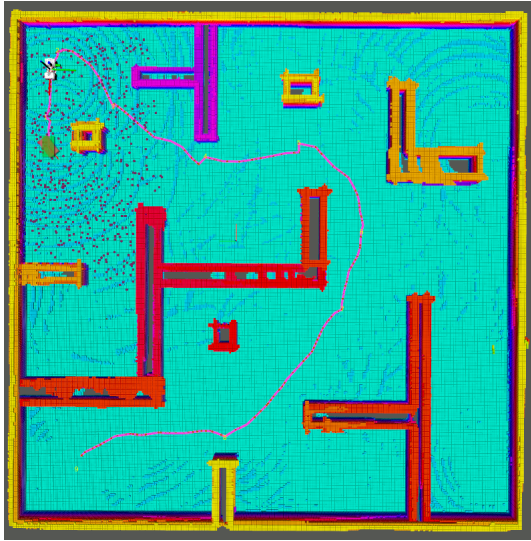
5.2.1 Path Planning Using Via-Points

The illustrations in this section show a fully mapped out world in addition to a planned path. The path is visualized in pink, and the via-points by the yellow numbers. In each world, one long and one short path is calculated using via-points in order to emphasize the intended usage of this system. The Voxblox visualization is RGB color-scaled by the z-axis, which means that changes in color represent changes in terrain-height. Large areas with one single color indicates a flat surface, and large variations in color indicate large variations in terrain-height.

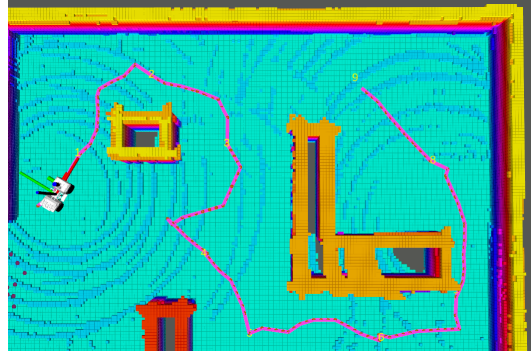
Figures 5.2 and 5.3 show how the path planning system performs in `maze_world` and `simple_world`. As these worlds are mainly flat with vertical walls, the ATV can traverse virtually any surface inside these worlds. The long paths in these two worlds span across large parts of each world, while the short paths visualize how the ATV can manoeuvre around walls and objects.

Figures 5.4 and 5.5, namely `normal_world` and `complex_world`, show how terrain-based environments are visualized in RViz. These worlds contain larger variations in terrain-height, which in turn is visualized by the variations in color. Larger variations in terrain creates fewer feasible paths for the ATV to traverse. For example, the vehicle navigates through valleys where the terrain is steeper than the max inclination, see section 4.6.3.

Figures 5.6 and 5.7 show four different paths planned in `heistadmoen`. As this is the main world for simulation, two sets of paths were planned. This is the largest of the five worlds, however, this does not affect the path planning system. The first figure shows a long and straight path from one end to the other, together with a path around a building. The second figure shows a path that goes down to the creek, see section 4.3.2, and up again past the large tree. This shows that the path planning system together with via-points creates reasonable paths for the ATV to traverse.

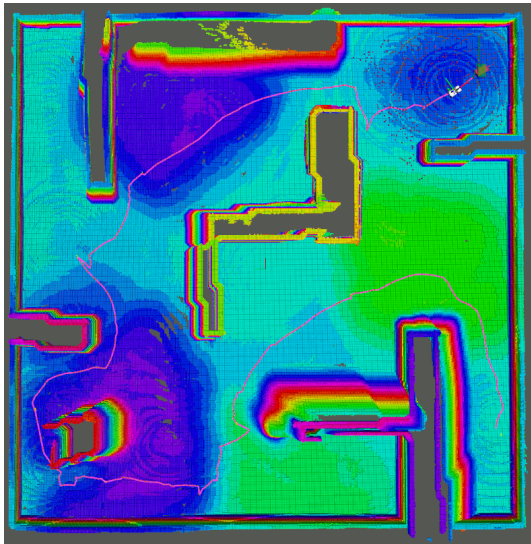


(a) maze_world with a long path

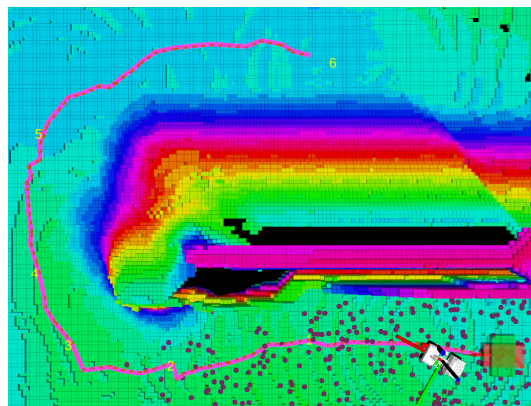


(b) maze_world with a short path

Figure 5.2: maze_world

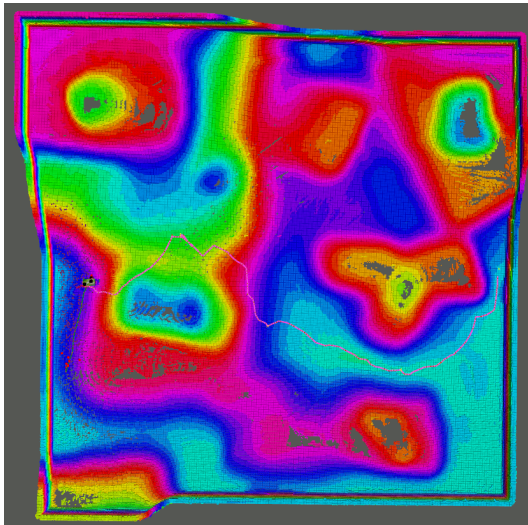


(a) simple_world with a long path

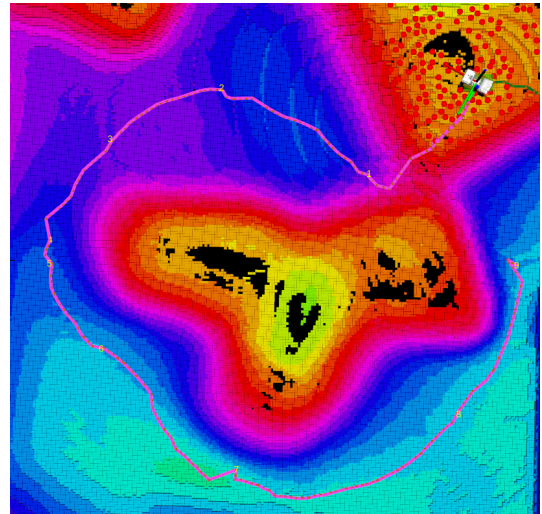


(b) simple_world with a short path

Figure 5.3: simple_world

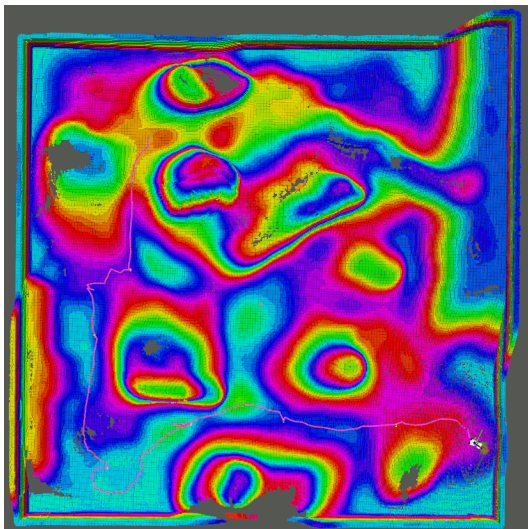


(a) normal_world with a long path

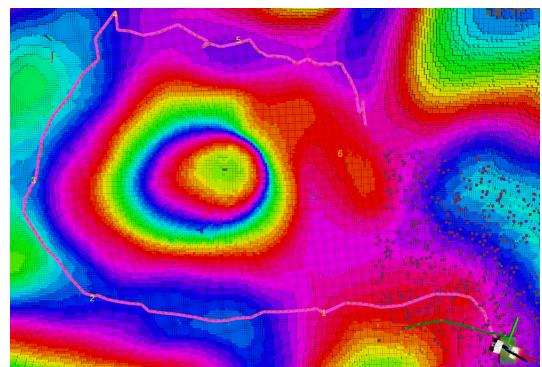


(b) normal_world with a short path

Figure 5.4: normal_world

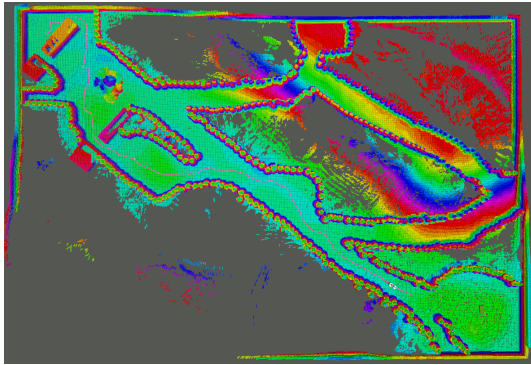


(a) complex_world with a long path

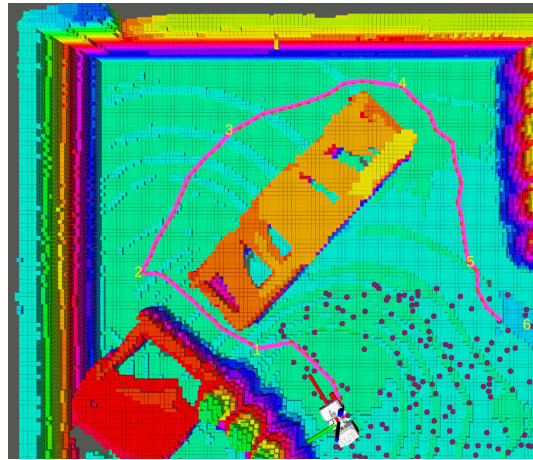


(b) complex_world with a short path

Figure 5.5: complex_world

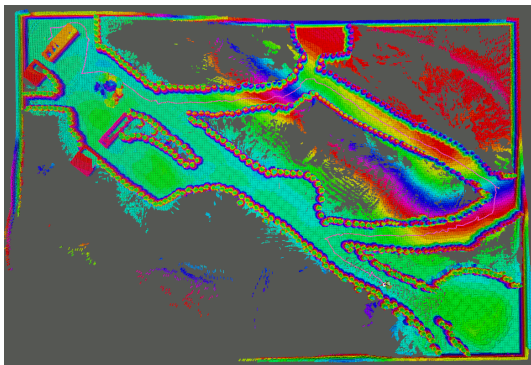


(a) heistadmoen with a long path

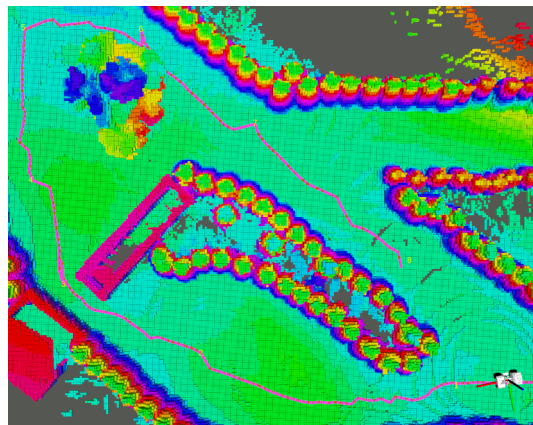


(b) heistadmoen with a short path

Figure 5.6: heistadmoen



(a) heistadmoen with a long path



(b) heistadmoen with a short path

Figure 5.7: heistadmoen

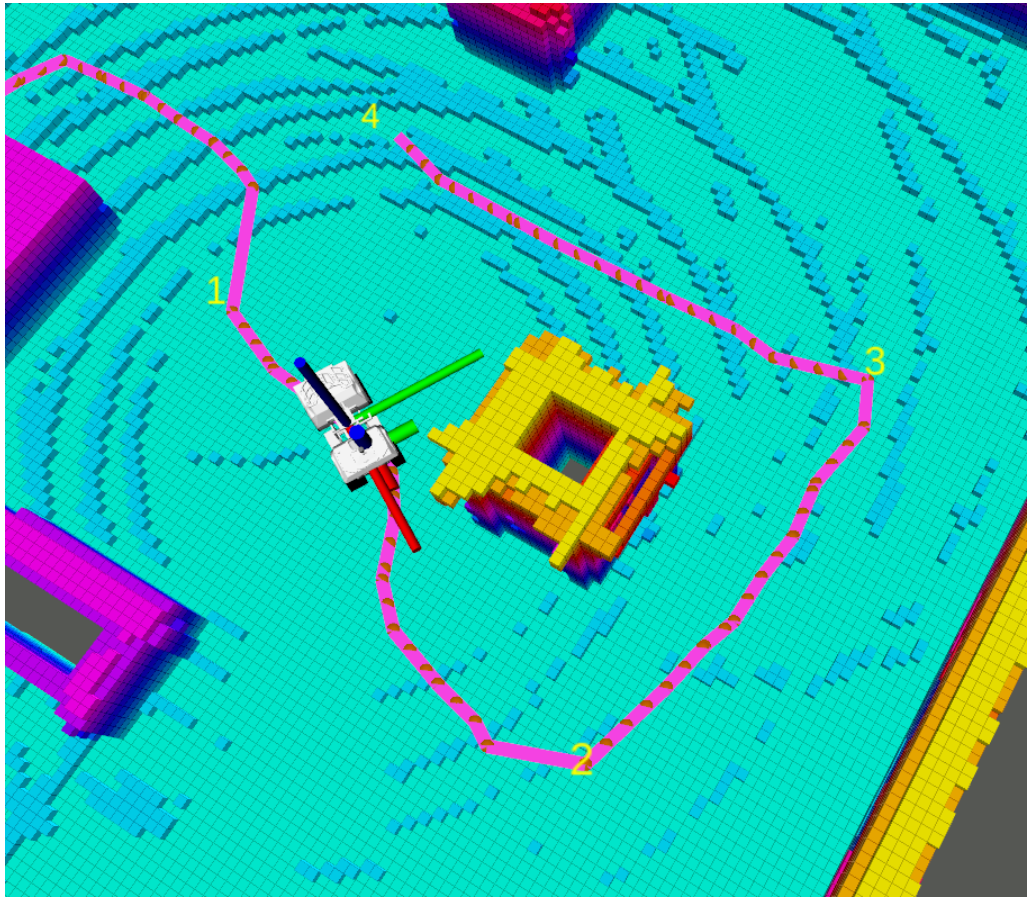


Figure 5.8: Closeup of four via-points with a path in `maze_world`.

Figure 5.8 shows the ATV while executing a path. It also shows the yellow numbers that visualize the via-points.

5.2.2 Path Planning Without Using Via-Points

The illustrations in this section demonstrate a planned path using *2D Nav Goal* in `heistadmoen`. The approach for doing so is described in section 5.1.1. Figures 5.9 and 5.10 show the planned paths in pink, while the navigation goal is illustrated by a blue arrow. The two figures also show what a world looks like when it is not fully explored, since the ATV has not traversed down to the creek.

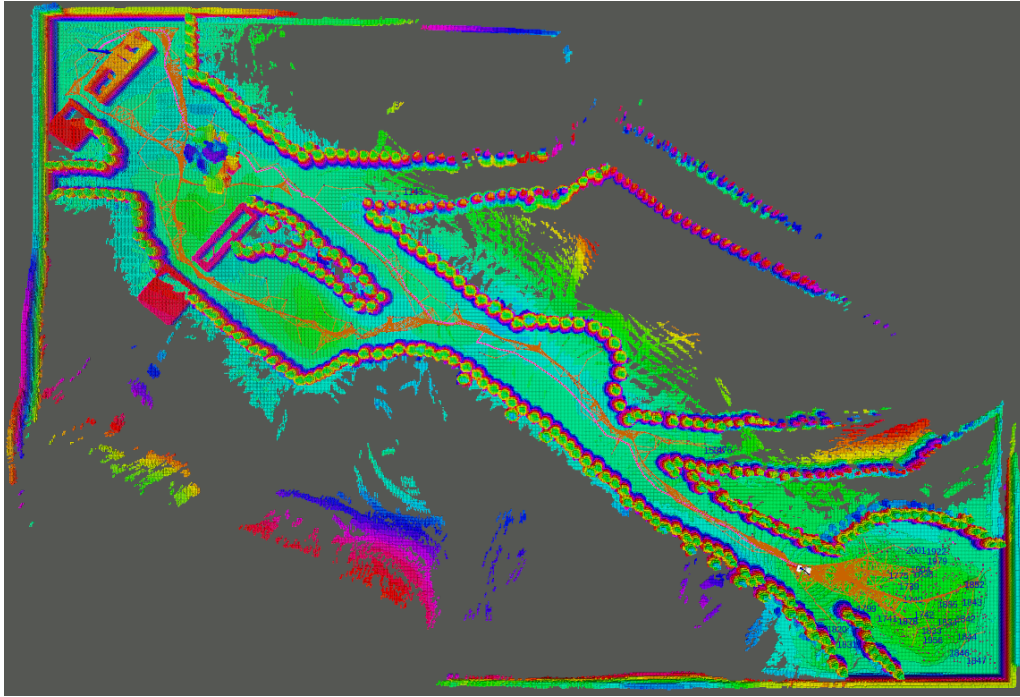


Figure 5.9: Planned path using *2D Nav Goal* in heistadmoen.

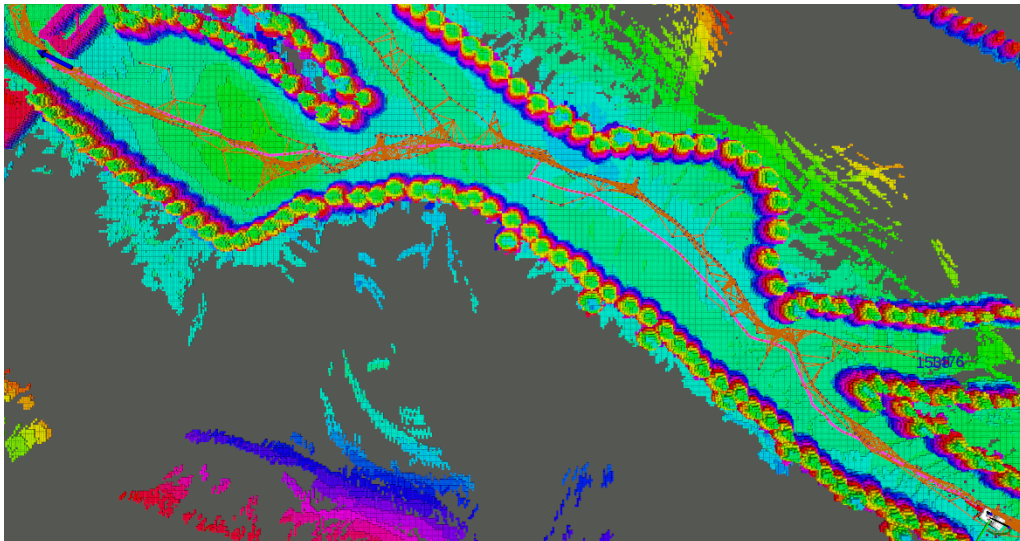


Figure 5.10: Planned path using *2D Nav Goal* in heistadmoen.

5.2.3 Fully Explored World

This section shows what a world looks like when it is fully explored by the ATV with the built-in explore-functionality in GBPlanner2. Figure 5.11 illustrates how `heistadmoen` looks when fully explored. This world is relatively flat, and as such, the maximum inclination does not affect this world. Figure 5.12 shows what `normal.world` looks like when fully explored. Since this map contains larger variations in terrain, the highest peaks are not covered by the global graph. This means that when planning a path through the world, these areas are excluded, even though they might result in a more direct path.

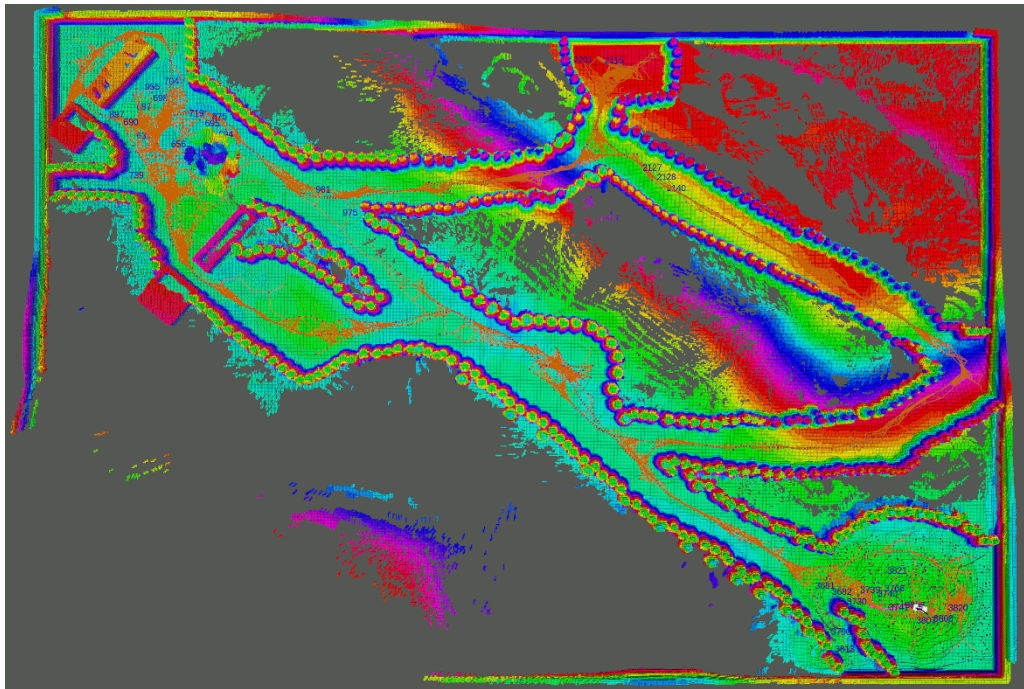


Figure 5.11: `heistadmoen` when fully explored.

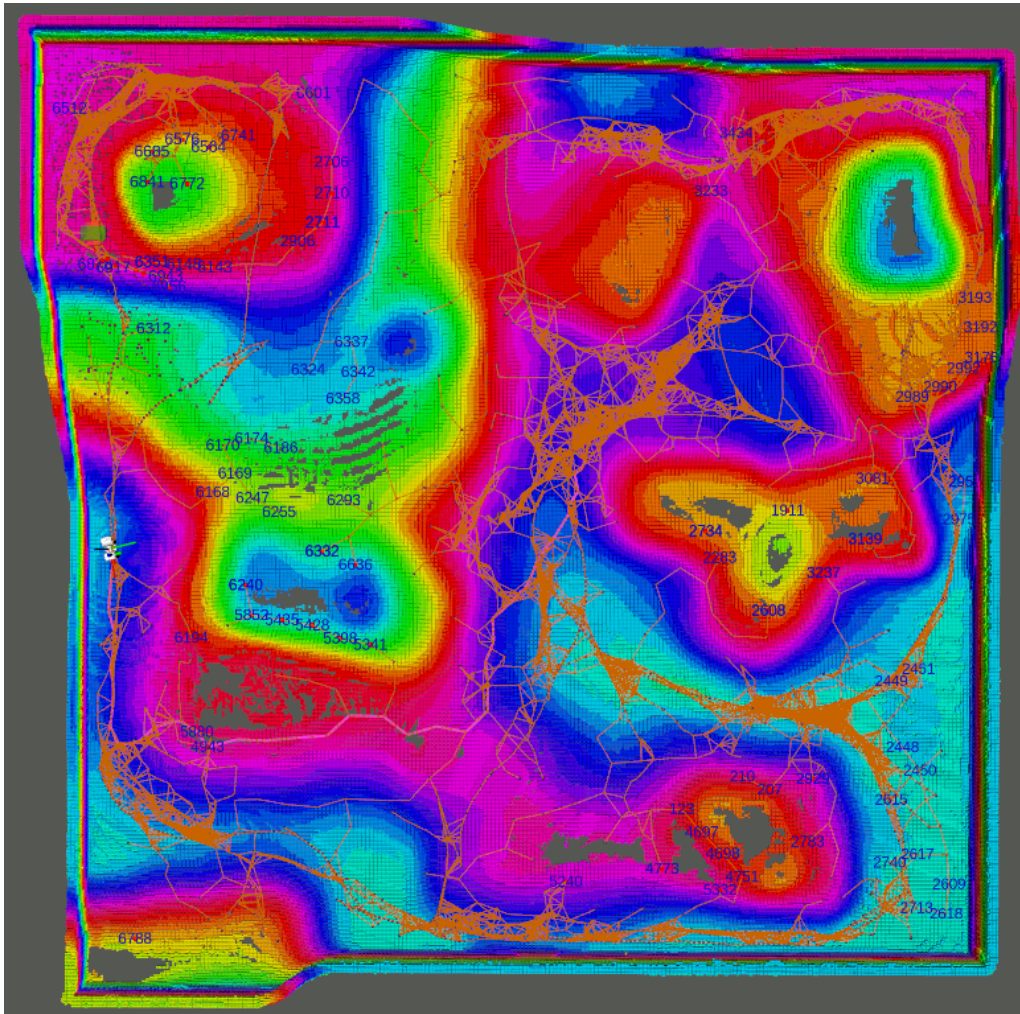


Figure 5.12: normal_world when fully explored.

5.3 Summary

To summarize the results shown in this chapter, it is clear that the implemented solutions work as intended. A number of simulations have been run across all five worlds, and the outcome of these simulations have given the expected results, as well as valuable information on how the ATV handles different terrain-based environments.

Chapter 6

Discussion

This chapter covers any further developments that would be beneficial for this project, as well as possible solutions to any complications. Since this project is part of the Lone Wolf project, there will also be suggestions on how to integrate the solutions in this project into the larger system.

6.1 Further Developments

When discussing further developments in this project, it is useful to separate them into repairs or new features. First, the suggested changes pertaining to known issues will be presented, before potential future developments that build upon the implemented system will be introduced.

6.1.1 Known Issues

When selecting two or more via-points outside the graph, a bug emerged which caused the system to crash. This issue should be resolved as soon as possible. However, due to the time constraints on this project, it was not prioritized.

The physics in Gazebo are not perfectly tuned for the ATV and the worlds in which it drives. For this reason, driving the ATV sometimes feels unnatural and the vehicle does not behave as one would expect. As driving the vehicle was not a central part of this assignment, this task was considered outside the scope, and thus not prioritized.

When planning a path using via-points, where the resulting path ends up crossing itself, see figure 6.1, the ATV will take the shortest total route from start to finish when navigating using the motion planner in GBPlanner2. This likely stems from the implementation of the motion planner and is assumed to be a relatively simple fix; however, as the motion planning is outside the scope of this assignment, so is this repair.

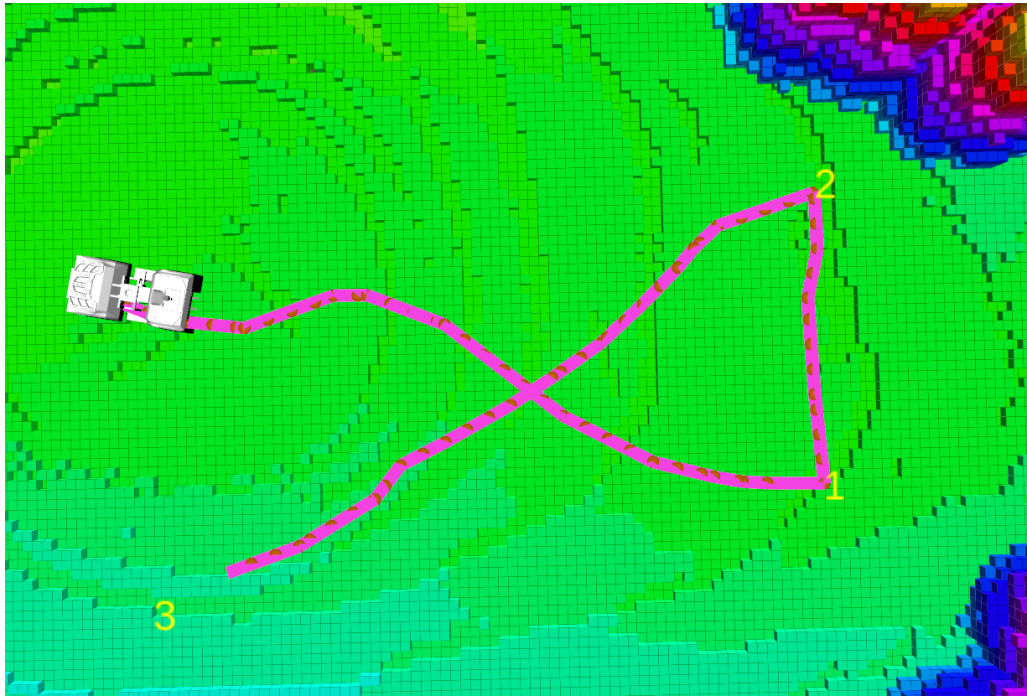


Figure 6.1: Via-point path crossing

6.1.2 Suggested Developments

Using GBPlanner2, the entire pipeline from robot perception to action is done internally. It would be useful to implement a system where an external point cloud could be used for path planning within GBPlanner2's framework. This would also be useful when bridging the project with ROS 2, as will be discussed in section 6.2.

GBPlanner2 was created with exploration as its main focus. Because of this, the graph used for path planning is only generated while exploring. It would be beneficial to implement the ability to generate the graph during both manual driving and while executing a path.

Allowing the vehicle to generate a graph during path execution could be used further to allow placement of waypoints in the void, that is, areas that have not yet been mapped by the robot. This will be necessary at one point, as planning in the real world could be realized by feeding GPS coordinates to the planner. These points will often be unknown to the vehicle, and as such it needs the ability to plan towards an unknown waypoint, updating its path and graph along the way.

GBPlanner2 combined with the implemented worlds and ATV serves as a foundation from which alternative path planning algorithms can be tested. This would be very useful as testing various algorithms could result in finding more optimal algorithms. One might for example test path planning algorithms which do not rely on a graph. Such algorithms might prove more effective at finding the shortest possible path than the currently implemented algorithm.

6.2 The Lone Wolf Project

As mentioned in the introduction of this thesis, see chapter 1, this project is part of a larger system involving several other technical groups who work on the Lone Wolf ATV. This thesis is an independent project, but it is desirable to integrate this project with the rest of the Lone Wolf system. In order to do this, it is necessary to transition the parts of this project that rely on ROS 1, over to ROS 2. In correspondence with NTNU ARL, this is something they are already working on. They do, however, not have an exact timeline for when this will be done. A reason for this is that GBPlanner2 has a tightly integrated software stack with ROS, and dependencies with other ROS-dependent packages. Since the planner only needs the point cloud and information about how the robot is situated in the world, it could be possible to use ROS Bridge in order to integrate this system with the Lone Wolf project. Both ROS 1 and ROS 2 are separate, and there is no direct communication between the nodes. However, the *ros1_bridge* package "provides a network bridge which enables the exchange of messages between ROS 1 and ROS 2." [57].

Chapter 7

Conclusion

This thesis has presented the work that has been done in order to solve the thesis statement presented in section 1.3; *Implement a system for path planning in a simulated terrain-based environment with support for adding via-points to the desired path. Furthermore, design and implement multiple worlds into the Gazebo simulated environment.*

Five novel worlds were designed using Blender and were then imported into Gazebo for use as a simulated environment. Figure 7.1 shows the five worlds in Blender; `maze_world`, `simple_world`, `normal_world`, `complex_world` and `heistadmoen`.

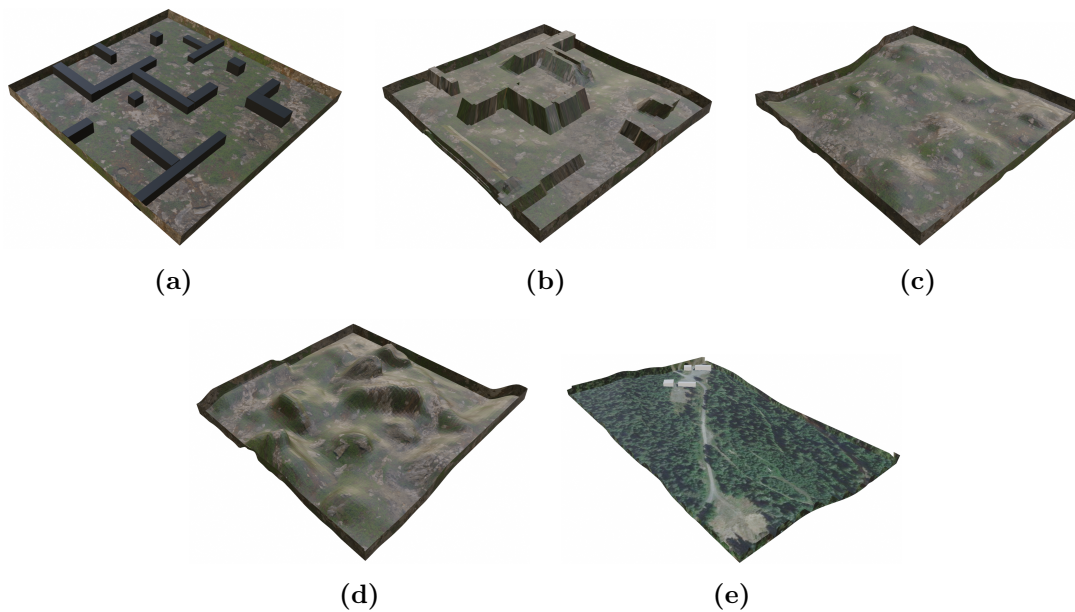


Figure 7.1: (a) `maze_world` (b) `simple_world` (c) `normal_world` (d) `complex_world` (e) `heistadmoen`

Once the worlds were designed, they were implemented into GBPlanner2's framework. When using GBPlanner2, the worlds were visualized using the built-in SLAM-package, Voxblox, in RViz. Furthermore, the previously designed 3D model of the Lone Wolf ATV was implemented into GBPlanner2 in order to test the path planning algorithm on the relevant vehicle. Figure 7.2 shows the worlds, including the Lone Wolf ATV, in RViz.

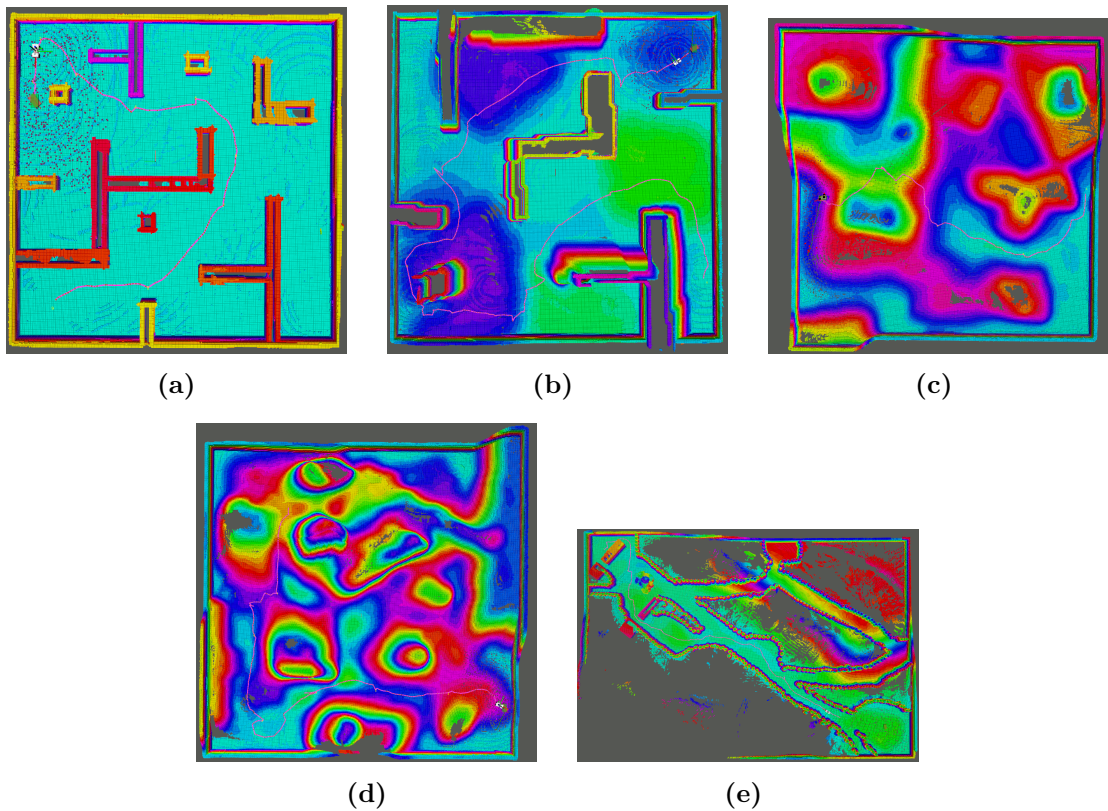
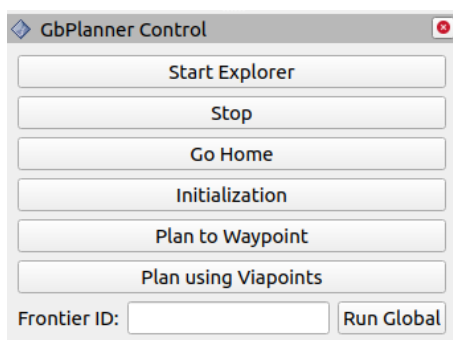


Figure 7.2: (a) maze_world (b) simple_world (c) normal_world (d) complex_world (e) heistadmoen

The ATV was now able to be tested within the custom worlds using GBPlanner2's path planning algorithm. Moreover, via-point functionality was implemented into the GBPlanner2 framework, and a GUI was developed within RViz to ensure usability of both path planning with and without the use of via-points. Figure 7.3 shows the GUI in RViz, and figure 7.4 shows path planning with and without the use of via-points in heistadmoen.

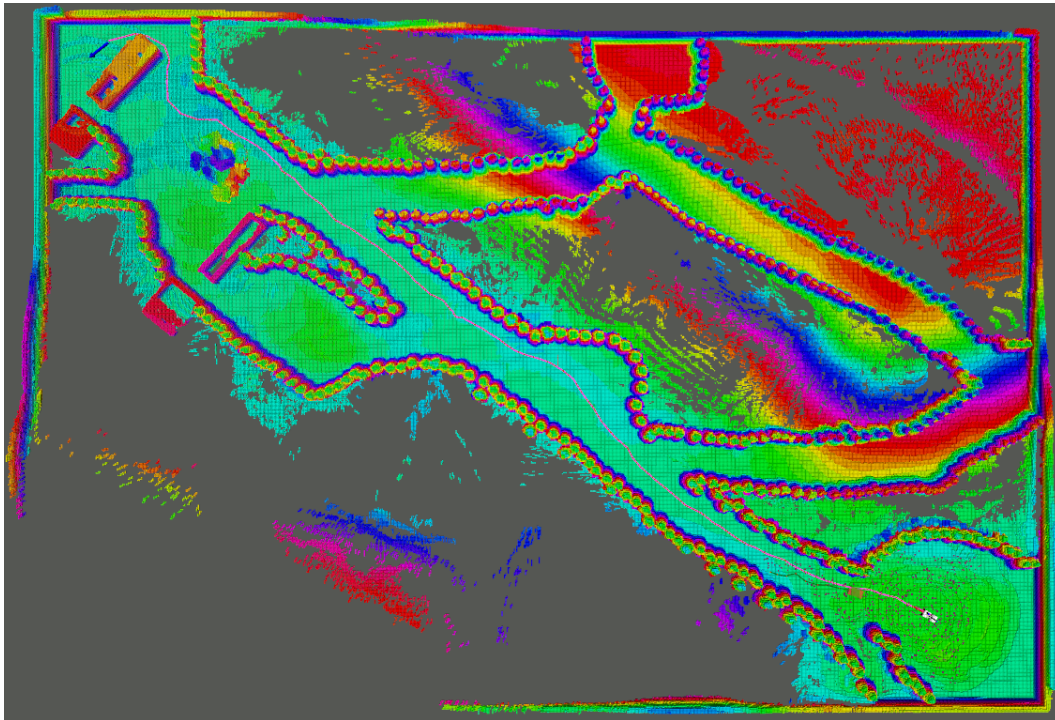


(a) GBPlanner2 Control Panel

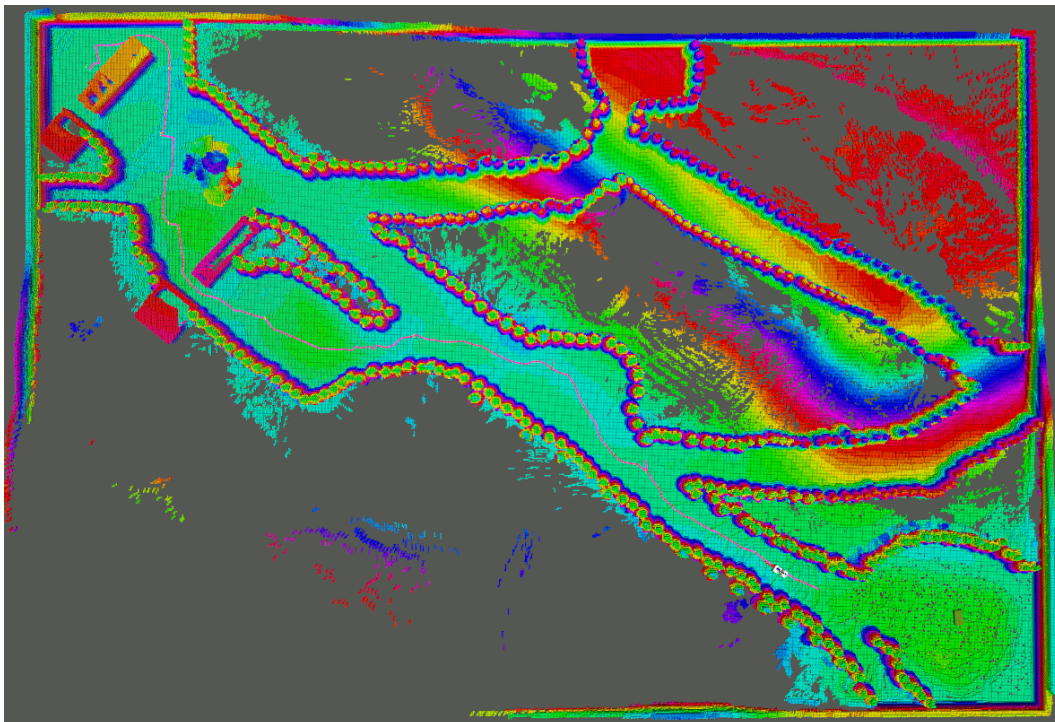


(b) 2D Nav Goal and Publish Point-button

Figure 7.3: Graphical User Interface in RViz



(a) Planned path using *2D Nav Goal*



(b) Planned path using via-points

Figure 7.4: Path planning in heistadmoen with and without the use of via-points

Bibliography

- [1] Kongsberg Defence & Aerospace. *Lone Wolf*. <https://www.kongsberg.com/no/careers/summer-interns/lone-wolf/>. Accessed: 26.01.23.
- [2] Kongsberg Gruppen. <https://www.kongsberg.com/>. Accessed: 12.05.23.
- [3] Kongsberg Gruppen. *Who we are*. <https://www.kongsberg.com/who-we-are/>. Accessed: 10.05.23.
- [4] Kongsberg Defence & Aerospace. *Who we are*. <https://www.kongsberg.com/kda/Who-we-are/>. Accessed: 10.05.23.
- [5] Sebastian Thrun. “Learning metric-topological maps for indoor mobile robot navigation ”. In: *Artificial Intelligence 99 (1998) 21-71* 1997 (1996), p. 23. URL: <https://core.ac.uk/download/pdf/82741752.pdf>.
- [6] MathWorks. *SLAM (Simultaneous Localization and Mapping)*. <https://se.mathworks.com/discovery/slam.html>. Accessed: 14.02.23.
- [7] NOAA. *What is lidar*. <https://oceanservice.noaa.gov/facts/lidar.html>. Accessed: 17.02.23. Jan. 2023.
- [8] Voxblox. *Voxblox*. <https://voxblox.readthedocs.io/en/latest/index.html>. Accessed: 05.05.23.
- [9] Ankit . *What is Signed Distance Function ?* <https://not-another-engineer.medium.com/what-is-signed-distance-function-818101f6fe3e>. Accessed: 05.05.23.
- [10] Voxblox. *Using Voxblox for Planning*. <https://voxblox.readthedocs.io/en/latest/pages/Using-Voxblox-for-Planning.html>. Accessed: 05.05.23.
- [11] Sudeep Sharan, Juan José Domínguez-Jiménez, and Peter Nauth. “Development of an modified Ant Colony Optimization algorithm for solving path planning problems of a robot system”. In: *2023 10th International Conference on Signal Processing and Integrated Networks (SPIN)* (2023).
- [12] Priyanka Upadhyay, Ravinder Singh, and Asha Rani. “Reliable autonomous navigation of mobile robot in unconstrained environment”. In: *2023 10th International Conference on Signal Processing and Integrated Networks (SPIN)* (2023).
- [13] Leonardo Spampinato et al. “DRL Path Planning for UAV-Aided V2X Networks: Comparing Discrete to Continuous Action Spaces”. In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023).
- [14] Liang Yang et al. “Survey of Robot 3D Path Planning Algorithms”. In: *Journal of Control Science and Engineering* 2016 (2016), p. 22. URL: <https://www.hindawi.com/journals/jcse/2016/7426913/>.

-
- [15] Kevin M. Lynch and Frank C. Park. “Modern Robotics; Mechanics, Planning, and Control”. In: 7th ed. Cambridge University Press, 2017. Chap. 9, 10.
- [16] Hui Liu. “Chapter 1 - Introduction”. In: *Robot Systems for Rail Transit Applications*. Ed. by Hui Liu. Elsevier, 2020, pp. 1–36. ISBN: 978-0-12-822968-2. DOI: <https://doi.org/10.1016/B978-0-12-822968-2.00001-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128229682000012>.
- [17] Allahyar Montazeri, Aydin Can, and Imil Hamda Imran. “Chapter 3 - Unmanned aerial systems: autonomy, cognition, and control”. In: *Unmanned Aerial Systems*. Ed. by Anis Koubaa and Ahmad Taher Azar. Advances in Nonlinear Dynamics and Chaos (ANDC). Academic Press, 2021, pp. 47–80. URL: <https://www.sciencedirect.com/science/article/pii/B9780128202760000108>.
- [18] Justin Svegliato. *How does a robot plan a path using RRT?* <https://towardsdatascience.com/how-does-a-robot-plan-a-path-in-its-environment-b8e9519c738b>. Accessed: 08.05.23. 2020.
- [19] Steven M. LaValle. “Rapidly-exploring random trees : a new tool for path planning”. In: *The annual research report* (1998).
- [20] Rahul Kala. “Rapidly exploring random graphs: Motion planning of multiple mobile robots”. In: *Advanced Robotics* 27 (Oct. 2013), pp. 1113–1122. DOI: 10.1080/01691864.2013.805472.
- [21] Amit Patel. *Introduction to A**. <https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>. Accessed: 20.05.23.
- [22] Syed Abdullah Fadzli et al. “Robotic Indoor Path Planning Using Dijkstra’s Algorithm with Multi-Layer Dictionaries”. In: *2015 2nd International Conference on Information Science and Security (ICISS)*. 2015, pp. 1–4. DOI: 10.1109/ICISSEC.2015.7371031.
- [23] František Duchoň et al. “Path Planning with Modified a Star Algorithm for a Mobile Robot”. In: *Procedia Engineering* 96 (2014). Modelling of Mechanical and Mechatronic Systems, pp. 59–69. ISSN: 1877-7058. DOI: <https://doi.org/10.1016/j.proeng.2014.12.098>. URL: <https://www.sciencedirect.com/science/article/pii/S187770581403149X>.
- [24] Blender. *The Freedom to Create*. <https://www.blender.org/about/>. Accessed: 05.05.23.
- [25] Open Robotics. *About Gazebo*. <https://gazebo.org/about>. Accessed: 16.02.23.
- [26] Adrian Macneil. *The importance of data visualization in robotics*. <https://www.therobotreport.com/the-importance-of-data-visualization-in-robotics/>. Accessed: 08.05.23.
- [27] automaticaddison. *What is the Difference Between rviz and Gazebo?* <https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/>. Accessed: 01.05.23.
- [28] Open Robotics. *ROS - Robot Operating System*. <https://www.ros.org/>. Accessed: 10.05.23.
- [29] Open Robotics. *Distributions*. <http://docs.ros.org/en/foxy/Releases.html>. Accessed: 16.05.23.
- [30] Open Robotics. *Understanding nodes*. <http://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html>. Accessed: 16.05.23.
- [31] Open Robotics. *Understanding topics*. <http://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html>. Accessed: 18.05.23.
- [32] Open Robotics. *Understanding services*. <http://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html>. Accessed: 18.05.23.
- [33] Arun Venkatadri. *ROS 1 vs ROS 2 What are the Biggest Differences?* <https://www.model-prime.com/blog/ros-1-vs-ros-2-what-are-the-biggest-differences>. Accessed: 10.05.23.
-

-
- [34] Navigation 2. *Nav2*. <https://navigation.ros.org/>. Accessed: 13.05.23.
- [35] Navigation 2. *Navigation Concepts*. <https://navigation.ros.org/concepts/index.html>. Accessed: 13.05.23.
- [36] Steve Macenski. *Architecture of the Navigaton2 stack*. <https://answers.ros.org/question/337994/architecture-of-the-navigaton2-stack/>. Accessed: 13.05.23.
- [37] Navigation 2. *Getting Started*. https://navigation.ros.org/getting_started/index.html. Accessed: 21.05.23.
- [38] Eitan Marder-Eppstein and Kurt Konolige. *NavFn Planner*. https://github.com/ros-planning/navigation2/tree/main/nav2_navfn_planner. Accessed: 21.05.23.
- [39] Anshumaan Singh. *nav2_theta_star_planner*. https://github.com/ros-planning/navigation2/tree/main/nav2_theta_star_planner. Accessed: 21.05.23.
- [40] Steve Macenski. *nav2_smac_planner*. https://github.com/ros-planning/navigation2/tree/main/nav2_smac_planner. Accessed: 21.05.23.
- [41] Navigation 2. *Tuning Guide*. <https://navigation.ros.org/tuning/index.html>. Accessed: 13.05.23.
- [42] Navigation 2. *Robots Using*. <https://navigation.ros.org/about/robots.html>. Accessed: 21.05.23.
- [43] Navigation 2. *Navigation Concepts*. <https://navigation.ros.org/concepts/index.html>. Accessed: 13.05.23.
- [44] MathWorks. *vehicleCostmap*. <https://se.mathworks.com/help/driving/ref/vehiclecostmap.html>. Accessed: 19.05.23.
- [45] Mihir Dharmadhikari, Tung Dang, and Nikhil Khedekar. *Graph-based Exploration Planner 2.0*. https://github.com/ntnu-arl/gbplanner_ros.
- [46] Tung Dang et al. “Graph-based subterranean exploration path planning using aerial and legged robots”. In: *Journal of Field Robotics* 37.8 (2020). Wiley Online Library, pp. 1363–1388.
- [47] Mihir Kulkarni et al. “Autonomous Teamed Exploration of Subterranean Environments using Legged and Aerial Robots”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 3306–3313. DOI: 10.1109/ICRA46639.2022.9812401.
- [48] Andreas Jonsebråten, Vegar Karlsen, and Glenn R. Varhaug. *Path planning simulation for an ATV in a terrain based environment using GBPlanner2*. <https://github.com/vegarkarlsen/lonewolf>.
- [49] Eline Marie Håve, Sigrid Mellemseter, and Cecilie Nikolaisen. “ROS Simulated World for ATV with SLAM”. In: (2022). URL: <https://hdl.handle.net/11250/3002448>.
- [50] Adobe. *COLLADA files*. <https://www.adobe.com/creativecloud/file-types/image/vector/collada-file.html>. Accessed: 21.04.23.
- [51] New Media Supply. *BlenderGIS Addon*. <https://blender-addons.org/blendergis-addon/>. Accessed: 21.04.23.
- [52] Open Robotics. *World File*. https://classic.gazebosim.org/tutorials?tut=components&cat=get_started. Accessed: 27.04.23.
- [53] Dave Hershberger, David Gossow, and Josh Faust. http://docs.ros.org/en/hydro/api/rviz/html/c++/classrviz_1_1Panel.html. 2015.
- [54] Open Robotics. *Make a Model*. https://classic.gazebosim.org/tutorials?tut=build_model. Accessed: 16.02.23.
-


-
- [55] Open Robotics. *Building a world*. https://classic.gazebo-sim.org/tutorials?tut=build_world. Accessed: 10.05.23.
- [56] Open Robotics. *teleop_twist_joy*. http://wiki.ros.org/teleop_twist_joy. Accessed: 16.05.23.
- [57] MathWorks. *Using ROS Bridge to Establish Communication Between ROS and ROS 2*. <https://se.mathworks.com/help/ros/ug/communicate-between-ros-and-ros-2-networks-using-ros-bridge.html>. Accessed: 14.05.23.

Appendices

Appendix A

Bachelor Assignment

Oppgaveforslag bacheloroppgave elektroingeniør (BIELEKTRO) i Trondheim, vårsemester 2023

Navn bedrift: Kongsberg Defence & Aerospace	Kontaktperson: Roger Werner Laug Epost: roger.werner.laug@kongsberg.com Telefon/mobil: 98220530						
Tittel på oppgave: <p style="text-align: center;"><i>Optimal path planning in a terrain-based environment</i></p>							
Hvilke studieretninger passer oppgaven for? (kryss av for alle aktuelle retninger; flervalg er mulig):	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px;">Automatisering og robotikk</td> <td style="text-align: center; width: 40px;"><input checked="" type="checkbox"/></td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">Elektronikk og sensorsystemer</td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">Elkraft og bærekraftig energi</td> <td style="text-align: center;"><input type="checkbox"/></td> </tr> </table>	Automatisering og robotikk	<input checked="" type="checkbox"/>	Elektronikk og sensorsystemer	<input type="checkbox"/>	Elkraft og bærekraftig energi	<input type="checkbox"/>
Automatisering og robotikk	<input checked="" type="checkbox"/>						
Elektronikk og sensorsystemer	<input type="checkbox"/>						
Elkraft og bærekraftig energi	<input type="checkbox"/>						
Er oppgaven reservert for noen bestemte studenter? (skriv navnene på studentene inn til høyre)	Glenn Varhaug, Andreas Jonsebråten, Vegar Karlsen						
Har dere arbeidsplass for studentene	<input type="checkbox"/> ja <input checked="" type="checkbox"/> nei <input type="checkbox"/> usikker?						
Er dette en lukket oppgave? Dvs. at sluttrapporten ikke kan publiseres fordi den inneholder sensitiv informasjon.	<input type="checkbox"/> ja <input checked="" type="checkbox"/> nei <input type="checkbox"/> ikke enda bestemt						
Kort beskrivelse av oppgaven med problemstilling. Lone Wolf er navnet på sommerprosjektet for studenter i Kongsberg Defence & Aerospace (KDA), Division Land Systems (DLS) der vi utforsker teknologier for fremtiden. Denne Bacheloroppgaven passer godt inn i utviklingen til neste års sommerprosjekt, der studentene skal blant annet finne en farbar vei og navigere autonomt med vår ATV.							
							

Lone Wolf ATV'en er konstruert til å kunne svinge, akselerere og bremse slik at den skal kunne kontrolleres fra en datamaskin. Det er tidligere implementert objekt gjenkjenning som et hjelpemiddel hvis det dukker opp uforutsette hinder i den planlagte ruten. Det er laget og montert en egenutviklet sprutsikker kasse til en PC med egen strømforsyning. PC kjører autonomi Software slik at kjøretøyet kan kjøre av seg selv, men autonomi er kun testet i Gazebo simulator til nå. ATV'en har en fjernstyrt nødstopp som skal brukes under kjøring. Kommunikasjon mellom ATV og en operatør posisjon er laget med en to-kanals kommunikasjonsløsning. Fra operatørposisjon kan ATV styres med en playstation-controller, og tilbake fra kjøretøyet mottas GPS-punkter, fart og retning som vises på skjerm.

Eksempler på komponenter og teknologier brukt: Raspberry PI, Arduino, SLAM, ROS2, Gazebo og MPC (Model Predictive Control).

Det er flere oppgaver på Lone Wolf som gjenstår for å kunne kjøre godt og trygt autonomt. Det studentene har vært mest interessert i er Path Planning. Dette tema er bredt og kan betinge god input for gjøre gode valg. For å kunne kontrollere ATV'en best mulig så trenger man input som Path Planning. Dermed kan denne oppgaven gå i begge retninger, eller man må anta visse input eller output for å kunne gjøre Path Planning. Denne oppgaven er dermed fleksibel og ikke trenger nødvendigvis inneholde dette:

- Bruk Gazebo Simulator til å lage en 3D verden der Path Planning algoritmer kan testes ut. Spesifikasjoner og krav til verden kommer.
- Bruk eller lag en algoritme for å kjøre ATV i simulator til et punkt og beregn optimal rute dit. Vurder alternative algoritmer og framgangsmåter for best mulig resultat.
- Det skal også være mulig for brukeren å legge inn «via-punkter» som den skal innom.

Alternative utvidede oppgaver:

- La ATV finne veien til et punkt i verden der punkt-sky ikke er etablert. Her må ATV prøve og feile på veien til målet basert på informasjon tilgjengelig.
- Bruk eksisterende algoritmer for gi input fra video om detaljer i omverden: et tre, gress, fjell, vei, etc. Her må man kombinere Lidar og Video slik at vi tar bedre beslutninger.
- Kombiner punkt-sky data og GPS for å plassere ATV i et topografisk kart. Operatøren skal kunne sette et mål punkt (evt via-punkter) og at ATV skal planlegge en rute dit. Alternative ruter til et mål kan også vises i et kart.
- Vurder hvilken input en kontroll Software har behov for slik at ATV beveger seg i ønsket retning og fart.

Opgaven vil kunne løses med allerede eksisterende Gazebo Simulator der en ATV modell er laget. Lidar er allerede modellert inn i modellen. Det vil kunne være behov for utvidelser for å oppnå ønsket resultat. Simulator og annen Software kjører i ROS2.

Dersom det er behov for utstyr skal KDA være behjelpelig med det (begrenset av tilgjengelighet og kostnad). KDA har hatt et møte med studentene og de virker interessert i oppgaven.

<https://www.kongsberg.com/careers/summer-interns/lone-wolf/>

Appendix B

Poster

Abstract

This poster summarizes the implementation of the path planning framework, GBPlanner, for a simulated version of the Lone Wolf ATV. It covers the implementation of via-point functionality to this framework, as well as a visual representation of the five novel worlds created for the ATV to be simulated in. Furthermore, simulations using the path planner are presented both with, and without, the use of via-points.

Worlds

For testing the ATV in various terrain and environments, five novel worlds were designed using the 3D graphics software, Blender, and were then imported into the robotics simulator, Gazebo. Figure 1 shows the five worlds in Blender.

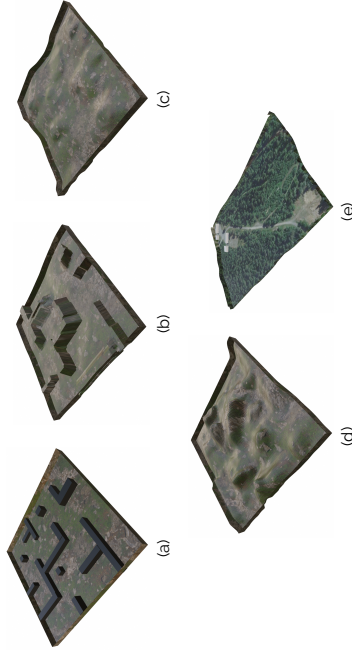


Figure 1. (a) maze_world (b) simple_world (c) normal_world (d) complex_world (e) heist_admoen

Planner

Graph-based Exploration Planner 2 (GBPlanner2) was used as a key component in this project. GBPlanner2 is a ROS package developed and maintained by NTNU Autonomous Robots Lab, and is primarily developed for autonomous exploration of subterranean environments.

To implement path planning with the use of via-points, GBPlanner2's global planner was modified. Calculation of small path segments was implemented into the planner, and each segment was combined in order to form a full path. This feature was integrated into the original structure of GBPlanner2's system, as seen in figure 2.

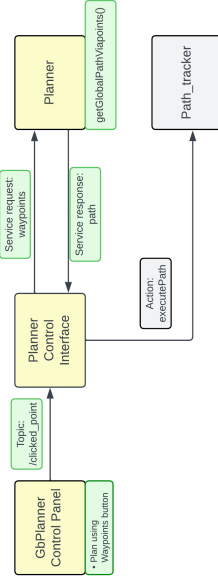


Figure 2. Overview of the via-point implementation. New components are marked in green, modifications are highlighted in yellow, and components kept intact are marked in grey.

System Overview

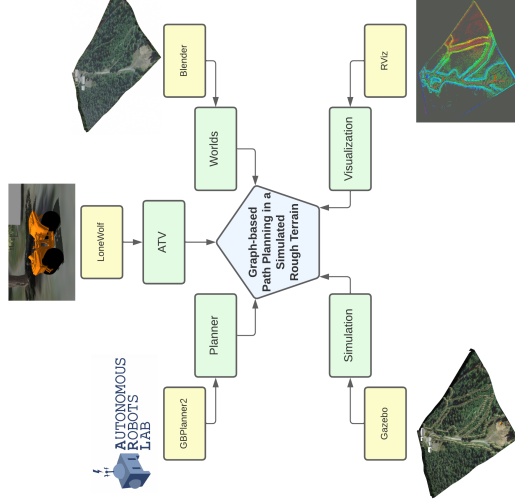


Figure 3. General system overview

Simulation and Visualization

Once the worlds were designed, they were implemented into GBPlanner2's framework. This enabled testing of GBPlanner2's functionality within the custom worlds. When using GBPlanner2, the worlds are visualized using the built-in SLAM-package, VoxBlox, in RViz. Furthermore, the previously designed 3D model of the Lone Wolf ATV was implemented into GBPlanner2 in order to test the path planning algorithm on the relevant vehicle.

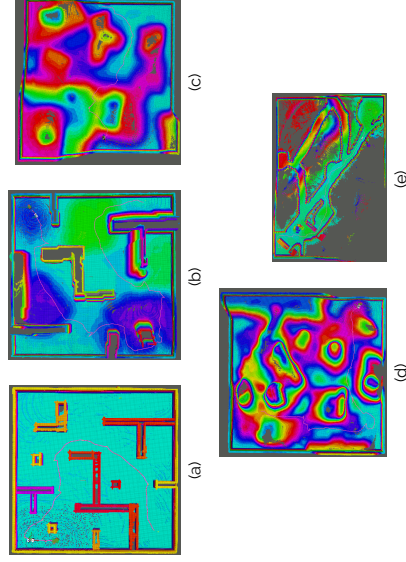


Figure 4. (a) maze_world (b) simple_world (c) normal_world (d) complex_world (e) heist_admoen

Conclusion

Figures 4 and 5 demonstrate that the implemented solutions work as intended. A number of simulations have been run across all five worlds, and the outcome of these simulations have given the expected results, as well as valuable information on how the ATV handles different terrain-based environments.

The figures show heist_admoen fully explored, in addition to a planned path with, and without, the use of via-points. It also shows a close-up view of path planning with via-points in maze_world. The planned path is illustrated in pink and the via-points are marked with yellow numbers.

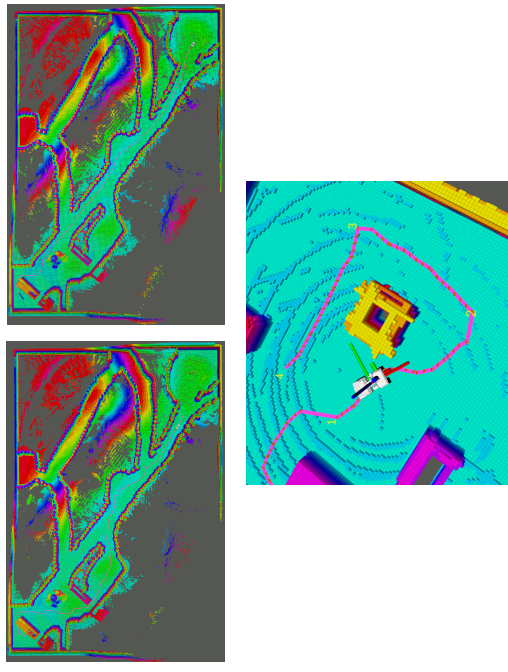


Figure 5. Top left: Path planning in heist_admoen using via-points. Top right: Path planning in heist_admoen without using via-points. Bottom: Path planning in maze_world using via-points, close-up.

Suggested Developments

Suggestions for any further developments that could be beneficial for the project:

- Implement the ability to utilize the RRG-algorithm in both manual driving and while executing a path.
- Allow the ATV to plan towards an unknown waypoint, while updating its path and graph along the way.
- Implement other path planning algorithms into GBPlanner2's framework.
- Integrate this project with the rest of the Lone Wolf project, possibly with the use of ROS Bridge.

References

- A. Jonsebråten, V. Karlsen, and G. R. Vårhaug, "Graph-based path planning in a simulated rough terrain," 2023.
- A. Jonsebråten, V. Karlsen, and G. R. Vårhaug, "Path planning simulation for an atv in a terrain based environment using gbpplanner2," <https://github.com/Vegardkarlsen/lonewolf>.
- T. Dang et al., "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1303-1388, 2020.
- M. Kulkarni et al., "Autonomous learned exploration of subterranean environments using legged and aerial robots," in 2022 *International Conference on Robotics and Automation (ICRA)*, pp. 3306-3313, 2022. Wiley Online Library.

