Master's thesis

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

Daniel Vennestrøm

Path Planning for Multi-Rotor Unmanned Aerial Vehicles (UAVs) Operating in Known Confined Space

Master's thesis in Industrial Cybernetics
Supervisor: Aksel Andreas Transeth, Sintef and Anastasios Lekkas, NTNU
Co-supervisor: Esten Ingar Grøtli and Dominik Natter, Sintef
July 2022

NTNU
Norwegian University of Science and Technology

Daniel Vennestrøm

Path Planning for Multi-Rotor Unmanned Aerial Vehicles (UAVs) Operating in Known Confined Space

Master's thesis in Industrial Cybernetics
Supervisor: Aksel Andreas Transeth, Sintef and Anastasios Lekkas, NTNU
Co-supervisor: Esten Ingar Grøtli and Dominik Natter, Sintef
July 2022

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

NTNU
Norwegian University of Science and Technology

NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

TTK4900 - Master's Thesis

Path Planning for Multi-Rotor Unmanned Aerial Vehicles (UAVs) Operating in Known Confined Space

Author:
Daniel Vennestrøm

April, 2022

Preface

This master's thesis represents the conclusion of a 2-year master's degree program; Industrial Cybernetics (MIIK) at the Norwegian University of Science and Technology (NTNU). The material presented are the results of a project spanning over the course of the 2022 spring semester.

I would like to thank my supervisor Aksel Andreas Transeth for his guidance and support throughout the process of the project. A thanks also goes out to my co-supervisors Esten Ingar Grøtli and Dominik Natter for providing their experience from working with path planners previously. I would also like to offer my condolences to Dominik Natter family, who unfortunately passed away during our cooperation. He was always encouraging me and helping me when any problems occurred. Thank you.

I would also like to thank all my friends who supported me during the writing of this master's thesis. The social gatherings and activities we did in between the writing of this thesis provided me with a lot of motivation. I would like to express gratitude towards my grandparents who have always been there for me. Finally, a special thanks and feeling of gratefulness goes to my parents for always encouraging me and supporting me throughout.

Table of Contents

List of Figures

List of Tables

Nomenclature

ACO	Ant Colony Optimization
API	Application Programming Interface
BIP	Binary Linear Programming
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
ESDF	Euclidean Signed Distance Field
GA	Genetic Algorithm
GSO	Glowworm Swarm Optimization
MA	Memetic Algorithm
MAV	Micro Air Vehicle
MILP	Mixed Integer Linear Programming
OMPL	Open Motion Planning Library
PAEDT	Propagating Approximate Euclidean Distance Transformation
POI	Point of Interest
PRM	Probalistic RoadMaps
PSO	Particle Swarm Optimization
ROS	Robot Operating System
RRG	Rapidly exploring Random Graph
RRT	Rapid exploring Random Trees
SBPL	Search-Based Planning Library
SFLA	Shuffled Frog Leaping Algorithm
SLAM	Simultaneous Localization and Mapping
SLCP	Safe Least Cost Path
STOMP	Stochastic Trajectory Optimization for Motion Planning
TSDF	Truncated Signed Distance Field

TSP Travelling Salesperson Problem

UAV Unmanned Aerial Vehicle

URDF Unified Robot Description Format

Vertex The term vertex/vertices is considered to be the same as node/nodes, however vertex/vertices will be preferred to prevent confusion with regards to ROS nodes.

Voxel A voxel is the equivalent of a 3D pixel. Many of the approaches that are going to be introduced will use this term seen as they utilize voxel based map representations.

Abstract

This master's thesis considers an use case where an Unmanned Aerial Vehicle (UAV) is going to autonomously investigating some point of interest (POIs) within a ballast tank onboard a ship. This requires a suitable path planner which performs well in the described scenario to be implemented. Through research and testing of various path planners this thesis aims to describe what constitutes a path planner that functions well for UAV in known confined space. The thesis only considers part of the use case, such that the focus is going to be on planning a path in known confined space and challenges related to this.

Throughout the work performed in this thesis it has been made clear that not all path planners for 3D applications are suitable for the use case at hand. The application area described in the use case above is not explored a lot in the available literature. Additionally, there is not a lot of available "complete" package which are up to data and can easily be implemented through ROS-melodic. This limits the approach of finding a pre-made path planner, which will work for the specific use case.

The path planners tested were GBplanner and Voxblox RRT planner. GBplanner did not achieve accuracy due to the global graph not including the goal pose. While the Voxblox RRT Planner was slow, but it usually managed to generate a path to the exact position. The final conclusion is that it is important to balance the different aspect of a path planning approach so that it fits the use case.

1 Introduction

1.1 Motivation and Background

Inspection of confined spaces is considered to be a dangerous field of work which traditionally has been performed by human workers. Boilers at power plants, storage tanks for fuel and ballast tanks on ships are examples of such spaces. Ballast tank inspection in particular is considered a hazardous job; partly because of the water being pumped in and out during operation leaving residues of aquatic life and/or pollution in the tank that can cause dangerous gasses to be formed. But also because of a lot of sharp edges and slippery surfaces. However, inspection of these types of environments is an essential part of routine maintenance that has to be performed to retain structural integrity and safety. It is therefore desirable to find alternative solutions to do these without exposing human workers.

Recently, due to the associated dangers, companies have developed and started to deploy specialized inspection robots to perform this type of inspection jobs instead. Through the recent development, it has been made clear that not all confined spaces are easy to traverse with ground robots. For example, a ballast tank's structural design may not allow for access with ground vehicles at all, or the POI where corrosion and fatigue fractures have a higher likelihood of occurring, may be in out of reach locations. There may also be residual water and debri in the bottom of the tank, making it even worse to traverse. In light of this, Unmanned Aerial Vehicles (UAVs) has become a common solution that provides more access through the added agility of flight.

Several companies have already developed UAVs with the purpose of inspecting confined or otherwise inaccessible spaces. One such company is ScoutDI, they have developed a system named; *Scout 137 Drone System*. Their system is under constant development and today their UAV has the capabilities of inspecting the confined spaces as described above. The system currently relies on a human operator along with advanced lidar localization which minimizes the risk of collision, but they are working towards making it more autonomous. One of the first steps towards this is to make it capable of planning paths from inspection point to inspection point without human intervention. This can be achieve through the implementation of a path planner. In 1.1.1, the specific use case ScoutDI is considering is detailed.

1.1.1 ScoutDI's Use Case

ScoutDI's use case concerns path planning for an UAV operating in the confined space of a ship's ballast tank. The UAV's task is to investigate a set of predefined inspection points (POI) within the ballast tank. To achieve this, the goal is to implement a path planner that enables the UAV to generate a path along a list of predefined poses. The provided poses correspond to the positions and orientations the UAV have to be in to get the inspection points within frame of the onboard camera. The layout of the ballast tank is considered to be known, so the 3D model and point cloud can be utilized to generate the path.

End Goal:

Provided a set of inspection points in a know environment, the UAV should be able to generate a safe path from an arbitrary starting position along all of the inspection points and then return.

However, there is not a lot of available research on path planners applied for this specific use case. Which is why this master's thesis is going to investigate through research and comparison, what constitutes a path planner that should functions well for target-orient path planning in known confined spaces. To limit the problem, only a part of the problems related to achieving the end goal (1.1.1) is going to be considered. This is further detailed in section 1.1.2.

1.1.2 Limitations

To limit the scope of the work, only part of the end goal (1.1.1) and use case (1.1.1) is going to be considered. This will correspond to only considering the generation of the path from the starting pose of the UAV to the first pose overseeing the first inspection point. The revised goal is then going to be to plan a collision free path from point A to B.

The thesis will present the most common and known approaches, this does not cover all the variations and is not an absolute overview. It will also be largely limited to static, known environments. Due to a lot of development within the field of path planners, many of these approaches are constantly being modified to work in certain scenarios, previously thought to be infeasible. This will eventually lead to some of the information presented in this thesis being inaccurate. However, considering the time frame of this project, it is not feasible to read up on all the novel

approaches currently being introduced. Seen as this being the case, consider most of the information presented as a base line representation of what is generally been thought to be the case with the path planner approach at hand.

Platform limitations
Due to potential integration of the path planner with ScoutDI's realistic AirSim-based simulator, the path planners tested will prioritize compatibility with Ubuntu 18.04 and ROS Melodic.

1.1.3 Research Questions

1. What constitutes a path planner that should functions well for target-oriented path planning in known confined 3D spaces?
2. What are the most important consideration to be aware of when implementing a path planner for the specified use case (1.1.1)?
3. Which metrics should be used to measure the performance of path planner operating in known confined space?

1.2 Literature review

3D path planning is a field that is currently under a lot of development and as such novel approaches is a common occurrence in the literature. The recent increase in computation power in small form factors has propelled the development substantially. Seeing as there is so much activity within the field lately, the different approaches available has become quite unstructured in terms of which type of implementations they are best suited for. Some articles ([1], [25]) has been published with the aim of structuring them, but so far no common taxonomy has been established.

There is in particular a lack of material comparing offline 3D path planners for known confined spaces. Compared to outdoor path planners that can rely on GPS signals and have significantly less obstacles to worry about, the types of path planners that are suitable for confined space have more challenges to overcome. However, some of the research performed in the related fields may be applied for path planning in confined space, so these articles have not been dismissed.

Comparison of Path Planning Approaches

Path planning in 2D has seen a lot of development over the years and several articles comparing these approaches have been published, some of which are [13] and [10]. These discuss the pros and cons of the various approach in 2D. If possible, it is desirable to apply some of the results provided in these works for identifying properties in 3D path planners.

Article [8] discusses an approach of applying 2D path planning for 3D by using a recursive algorithm to compute the missing dimension. This approach may be good for applications where the UAV is not expected to go through a lot of elevation changes. In the article this approach is applied for a A* and genetic algorithm (GA) in a simulated 3D rough terrain environment with variable levels of noise. It concludes that A* is best suited for real-time planning due to its low compute time, but that the GA is the more flexible approach, since it generates more than one path. This allows the path choice to be modified depending on the scenario.

In [13] the performance of the cell decomposition (CD), voronoi diagram (VD), probability roadmap (PRM) and visibility graph (VG) was measured for 2D cluttered environments. The conveyed results clearly indicated through varying the degree of clutteredness, that the performance is greatly impacted by how cluttered the environment is. Some of the planning approaches scaled better or worse depending on the degree of clutteredness. VG consistently returned one of the shortest path lengths. However, it was also the one with the longest computation time (except for in the test with the least clutter). The test indicated an exponential increase in computation time for linearly increased clutter. This was a trend that was somewhat evident in the other path planning algorithms too, but not as pronounced as for VG. If computation time is not a consideration, VG provides good results in cluttered 2D space. In [26] a 3D adaption of VG is successfully implemented for a collision avoidance oriented path planning algorithm, mainly with focus on manipulators. While in [9] an 3D adaption which utilizes VG in conjunction with the principle of minimum potential energy to identify the shortest paths is introduced. This article is more directed towards mobile robots, but also quite old which limits its relevance to UAVs. Overall, no articles concerning VG for use in 3D space on UAVs was discovered.

[24] and [25] provides an overview of the most common approaches to 3D path planning, along with the basic properties of each. Another overview is provided in [1], but this is more directed towards 2D path planning, and is not as detailed as the two other articles.

In [27] A* and RRT (+variations) are applied for a real-time 3D path planning scenario. The approaches were compared and the conclusion was that A* produces the shortest path, the fastest computation time and the highest success rate. It is however important not note that the approaches discussed in this article is real-time, which for most applications means only a limited workspace is considered for each planning. This may or may not skewer the results with regards to the use case considered in this thesis.

In [14] the A* and Mixed Integer Linear Programming (MILP) approaches are compared. The results conveyed were that A* was the faster approach, but that MILP overall was more flexible and better for complex problems.

Indoor Path Planning

In [18] and [11] the topic of indoor path planning for UAVs is discusses. Indoor path planning has a lot in common with path planning in confined spaces.

[18] introduces an approach for *Indoor Path-Planning Algorithms for UAV-based Contact Inspection*. This approach leverages a point cloud of the entire environment, which first has to be pre-processed by segmenting the space into rooms and individual discretize them. The rooms are discretized into voxel maps, which can be used for accurate path planning. It also implements an additional feature which adds "Security-offset voxels", to keep a safe distance away from any obstacles. Lastly it is worth mentioning that the approach leverages an algorithm that aligns the coordinate system of the voxel map with the walls of the environment. This makes the map smoother and more accurate, given that there is a set voxel size and the walls of the rooms are orthogonal.

The article [11] discusses an approach that leverages an almost fixed flight altitude through a feature they have named; `Safe Least Cost Path (SLCP)`, along with a set minimum distance away from obstacles. It uses an voxel based map representation and a 3D propagating approximate euclidean distance transformation (3D PAEDT). Then the A* search algorithm is utilized on this map to generate a path. The 3D PAEDT map representation shares a lot with Voxblox ESDF maps. This approach proved to be safe and easily modifiable, but there existed a trade-off between accuracy and computation time. This was because the voxel size could be adjusted, which resulted in more or less voxels to check during path queuing.

1.3 Outline

The first chapter of this thesis is the introduction, here the background and motivation for performing the work is explained, along with what this thesis aims to achieve. This section also introduces some relevant research questions (**??**) to be solved later on. The second chapter is; Theory, this chapter will include theory on relevant material that will be of use for the following chapters. After the theory has been presented, the method for testing and obtaining results are explained in the chapter; Method and Setup. Following this is the Results and Discussion chapter, here the results from the performed tests and research are presented and compared. Finally the thesis will present a conclusion in the chapter; Conclusion and Future Work.

2 Theory

In this chapter some of the main concepts and approaches to 3D path planning will be briefly introduced and through that form a foundation for choosing which path planners to test, given the use case at hand.

Path Planner Approach: This term is used as to not confuse the full implementation of a package with what the literature refers to as a path planning algorithm. The term; path planning algorithm, is often loosely used in the litterateur to either describe a whole path planner approach or just a search algorithm. To avoid confusion due to this, the term path planner/planning approach is used to describe the whole implementation including the map representation/generation, search algorithm and addition features.

2.1 What is Path Planning?

According to the available literature, path planning is a term with many definitions. Some of which are listed in 2.1, from [4]:

1. A collision-free path from the start to the target according to an evaluation standard in the obstacle environment.
2. Planning feasible paths for robots according to the task they assigned with.
3. The process of computing an optimal or near-optimal, collision-free path through an environment containing obstacles. The path is optimal with respect to criterion specific to the application
4. The task to compute and follow a path from a given start to a given goal position in the game world.

As one may observe by how much these definitions vary, the definition largely depends on the application. This further emphasizes the importance of considering the use case, when designing a path planner. The goal and environment are determining factors to consider towards choosing the correct path planner. Seen as this thesis is considering a certain use case, this knowledge can be used to filter out a lot of the irrelevant planning methods available.

A lot of path planners today have been developed with a specific purpose in mind.

Over time this has created a bunch of different path planners, each with a specific purpose and certain functionalities built in to accommodate the use case. These tailor-made path planners usually function well in their specific environment, but are not necessarily easily adapted to other scenarios. In the following subsection some key elements with regards to choosing a path planning approach given a specific use case will be introduced.

2.1.1 Goal Variations

Since the goal of a path planner varies a lot, an consideration to make is whether the path planner is a target-oriented or an exploratory type of planner. The goal of a target-oriented path planners is in simple terms; to plan a collision-free optimal/non-optimal path from A to B. While an exploratory planner has the goal of incrementally plan safe paths into unknown areas and map them simultaneously. Exploratory planners are in a certain sense the 3D equivalent of what is referred to as a coverage planners, seen as these too has as its goal to cover an entire area. A couple of terms that are closely linked to exploratory planners are whether the planning that occurs is local or global. The local planning ensures that the space which is unexplored is identified and set safe trajectories towards. This planning usually occurs online which requires it to be fast. This usually means only a limited part of the map may be considered at a time, this part is commonly referred to as the workspace. Because of this the planner has to have another way of planning over entirety of the space when not exploring or returning, which is where the global planning occurs. Path planners may have other goals as well, but for the purpose of this thesis these are the ones that are considered.

Collision-free vs. Optimal Planning

The objective of planning a path from A to B can be done by simply iterating until a path that avoids the obstacles is achieved. Planning approaches that employ this strategy usually return a sub-optimal path that eventually reaches the goal. For other approaches utilizing algorithms such as dijkstra's algorithm and A*, the resulting path is inherently the shortest near-optimal path. Path planners that return optimal paths usually leverages optimization which requires some sort of cost function to be initialized. The cost functions may consider different properties/metrics during the path generation depending on the conditions enforced by the goal, environment and other factors related to the application. Examples of what the cost function may contain is: path length, altitude change, proximity, flight time, battery usage, etc.. The cost function is minimized, which will return the optimal path with

regards to the chosen criterion.

2.1.2 Environment Variations

When choosing a path planner approach there are some key elements to consider with regards to the environment of the use case. Below are some of the most important considerations to make with regards to this.

2D vs. 3D
An important distinguishment to make is whether the path planner is intended for use in 3D or 2D space. For 3D space, a lot of the same approaches as for 2D space can be applied, however these tend to have varying degree of success. This is mainly due to the increased complexity introduced by 3D space and the resulting computational load. The computational load is especially a problem for UAVs performing online path planning, since they usually have quite limited payload. Due to this they can not carry the required components for performing the planning fast enough. For offline path planning, this is less of a problem seen as the path can be calculated on another more powerful computer and then uploaded to the UAV. An assessment with regards to if the environment is 3D or 2D should be made regardless, because unnecessarily long path computations time is not a desirable property.

Static vs. Dynamic
Another consideration to make with regards to the environment is whether it is static or dynamic. For a static environment it is fine to utilize an offline path planning approach. This is because the environment is assumed to stay the same consistently, as so; a path generated prior to the actual flight of the UAV will still be valid. When considering a static environment, the path can be evaluated and tested in a simulator to verify that it clears all the obstacles. This is especially useful when the planner does not initially consider the dynamics of the UAV. This way of verifying a path may be used for dynamic maps as well, just to get an initial impression of the path.

For online path planning the environment is continuously re-evaluate in real-time and a new path can be generate live should the environment change during the path traversal. Online path planning requires a fast search algorithm that can generate a path given new sensor data faster than the UAV can traverse the previously generated path. Stopping in mid air to generate the new path is for many applications not ideal, as a lot of energy will be wasted. The fast planning may result in poorer

accuracy, but with the appropriate safety features this should not be a problem.

2.2 Components of a Path Planner

In this section a proposed component structure to the concept of a path planner is going to be introduced. This section is going to consider the most common approaches and identify common and different components across the field. These components are considered to influence the performance of the path planner, and are worth defining.

Since one search algorithm can be applied to different map representations, the term path planning algorithm is somewhat vague in the authors opinion, and is one of the reason for introducing the component structure.

Proposed structure:

- Search Algorithm
- Map Representation
- Additional Features

2.2.1 Search Algorithms

There are several different search algorithms that can be applied to either generate a collision-free or optimal path. In [13], it is mentioned that path planning typically has two stages; the pre-processing phase and the query phase. The pre-processing phase refer to the generation of the map representation given the available sensor data. Whilst the query phase refers to the search of valid paths, which is where a search algorithm is usually applied on the map representation. Note that the map representation is typically generated with respect to what search algorithm is going to be applied, and that not all search algorithms require pre-processing. Also, even though a search algorithm finds a path, it is not necessarily the optimal path. This depends on which search algorithm is being used.

2.2.2 Map Representation

Depending on the search algorithm, the environment needs to be setup accordingly. Some path planner require some pre-processing of the map to occur first. This

is usually done through some form of discretization of the 3D space. A common requirement for the map representation is that it distinguishes between free space and occupied space. This may be done in several different ways, see 2.3.6 for some common map representations.

2.2.3 Addition Features

Additional features may be added as additional layers to the path planning approach to make it function better with the relevant scenario. Some of which are detailed below.

Optimization
Some search algorithms does not perform optimization inherently, this usually results in sub-optimal paths. However, optimization layers may be added on top of these algorithms to make them return shorter paths. Another approach is to add conditions/constraints to non-optimizing search algorithms that nudges the final path towards a more optimal solution.

Smoothing
In certain scenarios it may be beneficial to implement a smoothing feature. In cases where a grid representation is used, depending on the resolution used on the map representation, the path may be quite jagged. To smooth these out, a common approach is to use polynomial algorithms that eliminates intermediate vertices and generates polynomials over the empty space. The polynomial can then be used to replace the intermediate vertices in a more ideal position, which results in a smoother path.

Paths with smoother curves work better with multi-rotor UAVs since the UAV's dynamics may be more compatible, making the traversal more efficient. Since the dynamic of a UAV does not allow for instantenious rotation, non-smooth paths may result in error that has to be corrected by the path following controller. This will increase time of traversal and energy consumption.

Safety adjustments
Since the search algorithms does not necessarily take into consideration the limitations of sensors and UAV dynamics, another factor to consider is safety. It is possible to mitigate these potentially dangerous inaccuracies through the use of safety factors. In [2] the path is modified through a path improving feature which

can move the vertices in close proximity to occupied space further away. While in [18], a security offset is added by assigning a set of voxels near the surfaces as occupied. This functions as an addition buffer to avoid collisions. When implementing these types of features, it is important to make sure that it does not interfere with the goal of the planner. For inspection purposes, it is very inconvenient if a large amount of voxels near the surface that is going to be inspected is inaccessible due to safety concerns. In such cases, there should be implemented exceptions.

Simulator

Some path planners are packaged with simulators. Depending on how accurate the simulator is, it can be useful tool for verifying paths before sending them to the actual drone.

One such simulator is Gazebo, which when combined with `Rotors Simulator` allows for UAVs to be simulated. Gazebo used with ROS provides some tools that makes it easy to manipulate the models and get the current states. The computational load depends on the size and complexity of the environment. In terms of graphics it is quite simple.

Another UAV simulator is `AirSim`, which is a plugin for the game engine `Unreal` that allows UAVs to be simulated and interfaced through an included ROS wrapper. This simulator emphasizes realistic graphics, which is ideal for accurately simulating SLAM. By utilizing a path planner to generate a path based on data generated from SLAM simulated in AirSim, an very accurate to reality overall simulation may be achieved. ScoutDI utilizes this simulator for their testing.

2.3 Path Planner Approaches

This section aims to provide information about the most common and relevant path planning approaches with regards the use case. Since there are so many different approaches, this sections is going to start off by structuring them. A few papers were procured and investigated for this purpose, however there did not seem to be a common structure used across the field. This was especially true with regards to path planning approaches used in 3D environments. According to the papers [24] and [25], the 3D path planning approaches can be divided into the 5 categories. The papers propose a taxonomy method for 3D path planning. Table 1 summarizes the proposed taxonomy.

Table 1: Path Planner Taxonomy 1

<table>
<tr><th colspan="2">Sampling-based algorithms</th><th colspan="2">Node-based optimal algorithm</th></tr>
<tr><td>Passive</td><td>Active</td><td colspan="2" rowspan="2">Dijkstra's algorithm
A* search algorithm
D* search algorithm</td></tr>
<tr><td>3D Vornoi
RRG
PRM
k-PRM
s-PRM</td><td>RRT
RRT*
DDRRT
Artificial potential field</td></tr>
<tr><th colspan="2">Bio-inspired algorithms</th><th colspan="2">Mathematics model based algorithms</th></tr>
<tr><td>Evolutionary</td><td rowspan="2">Neural Networks</td><td>Linear algorithm</td><td rowspan="2">Optimal control</td></tr>
<tr><td>GA
MA
PSO
ACO
SFLA</td><td>MILP
BIP
Flatness based</td></tr>
</table>

Table 2: Path Planner Taxonomy 2

Sampling-based algorithms	**Artificial Intelligence techniques**
Cell decomposition	Heuristic search
- Exact cell decomp.	- GA
- Mod. adap. cell decomp.	- Greed algorithm
Roadmaps	Brute force search
- A*	- Depth first search
- RRT	
- Vornoi	Local search
- PRM	- Travelling salesman
Potential Fields	
- Artificial potential field	Artificial neural network

Another taxonomy was proposed in [1], table 2 summarizes the proposed structure. This taxonomy was more general and did not seem to focus as much on 3D path planning approaches specifically.

The taxonomy (table 2) suggested in [1] tends towards path planner approaches most commonly used for 2D environments. Whilst the taxonomy in table 1 was created specifically with 3D path planners in mind. In addition to this, taxonomy 1 was more thorough in differentiating the variations into subgroups, which made it more of a complete overview. Based on this, taxonomy 1 (1) is chosen for forming the basis of how the the path planner approaches are going to be structured in this thesis.

In the following sections, some of the most common and relevant path planning approaches will be introduced. The approaches will be briefly explained, followed

by some examples of what their most common use areas are.

The following path planning approaches are not necessarily complete since they will require an additional layer to really function. For some approaches the map representation is integrated, while for others the map representation is assumed to be generated separately. In the section 2.3.6 the most common environment representation will be somewhat explained, but note that these methods varies greatly and are not always that well documented. This is because in many cases the map representation is assumed to be unique to the planner and therefore a part of it. However, through studying the different approaches introduced in the literature it is clear that many of them utilizes variations of different groups of map representations.

2.3.1 Sampling Based Algorithms

According to the chosen taxonomy, sampling based algorithms are divided into passive and active types. Active types does not require any pre-processing of the environment, while passive sampling based algorithms require some pre-processing of the map to function. This category of path planner approaches are commonly used and a lot of the other categories build on these.

RRT - Rapidly exploring random trees
This is an active type of sampling based search algorithm. There are many variations of this algorithm, some of which generate optimal paths (RRT*), but the basic version of RRT generates a possible collision-free path. It functions by sampling a configuration space which indicates what space is occupied and not.

RRT methods are not suited for dynamic planning due to its tree like structure. If a new obstacle were to be dynamically introduced, RRT does not provide a function for re-planning already developed trees.

RRT* is a improved version of RRT which functions in a more goal-oriented manner, deleting branches that does not yield the desired results according to a designated cost function. This results in shorter paths, but also makes it more suitable for 3D applications. This is because the search space is effectively shrunk, since the path planner is not searching blindly.

PRM - Probabilistic Roadmaps
Generates a road map of the environment consisting of vertices and edges. Accord-

ing to the taxonomy it is a passive sampling based algorithm, which means it is not considered as an approach that finds the path on its own. It instead utilizes a separate search algorithm over the road map it has generated to find a path to the goal.

This method is a multi-query methods which means that it is not propagating from a single source vertex, but samples multiple vertices within the free space and attempt to connect them to the nearest vertices by checking the edges are collision free. This process occurs all over the free space of the map simultaneously, which is the main difference between this method and RRT.

A downside to this approach is the amount of time this process requires to complete. If the environment is large and requires a dense resolution, this method would use a lot of time to generate the entire map. However, as seen from a lot of the different approaches discussed in this section, there exists improved algorithms that limits this behavior by for example only sampling k-neighbors (K-PRM), this limits the time spent at each vertex. Another variation only connects to vertices within a set radius. In addition to this there are also combinations of these that are being used.

In a certain perspective PRM can be seen as a method for generating the map representation for a search algorithm to applied. However, according to the taxonomy it is a passive path planning approach which is why it is included here.

Artificial Potential Field

This approach employs a map representation which assigns all the surfaces of the environment a repelling force with respect to the goal. The force degrades depending on the proximity to the surface, so the sum of the forces on a point can be used to form a gradient which nudges it towards the goal. While the point is to close to an obstacle, the force will be adequately large so that it won't collide.

This method provides fast convergence, but runs into the risk of ending up in a local minima where all the forces are balanced and the point is not in the goal. There has been made efforts to avoid this, some of which are mentioned in [25]. Finally, it is also important to note that the map generation may still be time consuming.

2.3.2 Node Based Optimal Algorithms

Contrary to passive sampling based algorithms in a sense, this category of path planning approaches require some pre-processing of the map to be applied. These types of algorithms function by searching and calculating the cost of various path in a already generated grid/voxel map or graph. Node Based Optimal Algorithms are also called graph search algorithms, but this is quite inaccurate seen as these methods can be used on grids as well. It is important to note that these types of algorithms are not limited to graphs, but that it is often preferred as a way to limit the possible paths and keep computation time down.

Dijkstra's Algorithm
Dijkstra's algorithm is commonly used to find the shortest path between two vertices in a graph. In [1] it is referred to as a graph-search algorithm. This algorithm too has several variants depending on the exact application. Along with A* it is one of the most utilized shortest-path algorithms.

How the algorithm functions is well documented in [25]. The general idea is to initially assign all the vertices, except the start vertex, a value of infinity, and then iterate over all the vertices and assign them new values according to the currently discovered shortest path. Then stop when the the goal vertex is reached.

The greatest difference between A* and dijkstra's search algorithm lies in the fact that dijkstra's does not utilize any incentivizing measures to ensure that the algorithm is heading in the right direction. Dijkstra's search algorithm simply iterates until the shortest path has been discovered. This may be okay for an environment with few vertices, but for larger environments this is time consuming and ineffective.

A* algorithm
This search algorithm is essentially an extension of Dijkstra's algorithm which implements a feature to initially reduces the number of available vertices for planning. ,This approach only includes the vertices that are thought to be in the general direction of the goal. This makes the search algorithm more goal-oriented and efficient than dijkstra's algorithm, because it does not need to iterate over the whole space to make sure that it has achieved the "shortest path". This method does however not guarantee the shortest path due to the reliance on heuristics to determine which vertices should be included for the goal oriented search for a path towards the goal.

Over the years A* has seen a lot of use in 2D applications and is commonly known

for being used in Google maps and many games with path planning features. In these cases the map representation has mostly been on 2D maps, but due to its goal oriented approach this algorithm has also seen some use in 3D applications ([27], [17]).

2.3.3 Mathematical Model Based Algorithms

Similarly to the approaches in 2.3.2, this group leverages optimization theory to achieve optimal paths. Contrary to the node based optimal algorithms, this group of path planning approaches are generally not a complete representation of the environment. The environment is represented using simplified mathematical models of the obstacles through applying constraints to a dynamic system. By expressing the the system mathematically, a lot of the noise can be removed and the states that matters the most for the planning can be focused on. Because dynamic constraint can be modelled, these approaches generally handle dynamic environments well.

MILP - Mixed Integer Linear Programming
This is a method of setting up a optimization problem that achieves optimal path planning. Because of its flexibility, this method can be used to describe and solve large and complex problems in an optimized manner. It is widely discussed and relevant withing the field, because of its ability to solve large problems in a short amount of time. Path planning approaches utilizing this method benefit from including more factors and several moving parts to provide context to find the optimal path. In eq. 1, MILP can be seen in its standard form.

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & x \; are \; integers \\ & Ax \leq b \\ & A_{eq} x = b_{eq} \\ & lb \leq x \leq ub \end{aligned} \tag{1}$$

2.3.4 Bio-inspired Algorithms

This category of a path planners imitates various processes that naturally occur in different species' behavior and thinking. These methods are naturally suited for solving NP-hard problems. Similar to mathematical model based algorithms, these group of path planning approaches generally function better in large scale

implementation where minor inaccuracies matter less. As seen in fig. 1, this category has a subcategory of evolutionary algorithms and neural networks. The evolutionary algorithms focus the behavior of species, while neural networks are based on how the brain functions and thus how neurons process information. These algorithms tend to function better on large scale application where accuracy is not prioritized.

2.3.5 Multifusion Based Algorithms

This group of path planning approaches was initially introduced in the taxonomy proposed in [25] by Yang. It concerns path planner approaches that utilizes a combination of the above mention categories to create a more flexible path planner. Some algorithms are better suited for large scale, whilst other require small environments to not become overly complex. The large scale algorithms does not provide good results in the small scale scenarios, but can be used as initial planning stage for getting the asset in the general area where a small scale algorithm can be applied and get the asset in exactly the desired position. Multifusion algorithms is a good alternative to keep complexity down while retaining accuracy in larger environments. A lot of algorithms already employ this strategy, one of which is GBplanner.

2.3.6 Common Map Representation

Since the above mention categories of path planning approaches may use a variety of map representations to plan over, this section is going to go through some of the most common representations.

Grid Map A grid based approach to implementing path planners which overlays a 3D map with a grid, segregating the space into cells. These cells contain information denoting whether it is occupied or not. This information can either be binary or weighted with regards to certain properties unique to the cell. Grid-based planning guarantees accuracy down to a certain resolution, but is hard to adapt to higher dimensional problems, due to the rapidly increasing number of cells and thereby paths.

The most basic representation of a 3D environments is a 3D grid or voxel grid. This type of map representation is usually generated through sampling and may be time consuming. This is because all the voxels within the environment has to labeled as occupied or free. There also exists variations of this type of representation that does not only consider the voxels as binary variables, but assign them values depending on the distance of the nearest obstacle. An search algorithm applied to

this type usually has to be quite goal-oriented, seen as sampling the entire space for a possible path is very time consuming. Another popular variation is to reduce the voxel grid into a 2.5D representations. This representation considers the space a pillars where only the highest/lowest point is labeled in a 2D grid where the cell is only considered as usable above/below certain points on the z-axis. By not considering a full 3D representation the computational load is greatly reduced.

Graphs
This map representation is used a lot in 3D environments due to the sparse nature. It consists of vertices and edges that is safe for the UAV to traverse. Search-algorithms (2.2.1) are then applied to the graph-structure to find an optimal route through the network of vertices and edges. Once a path is returned, it is usually refined through smoothing to avoid harsh movements that the physical dynamics of the UAV will not be able to follow.

It is important to note that there are different types of graphs. Some graphs are generated with the sole purpose of not causing any collisions, while others are created to form a skeletonization of the environment. The type of graph depends on the use case. If a graph is too sparse, it will lack accuracy for target-orient path planning, while if it is to detailed, it will cause long planning times.

For larger environments it is important to limit the computational load. A method of doing so is what is referred to as space skeletonization, which essentially is a skeleton of points and edges within the unoccupied space of the relevant map. These type of graphs are often generated from a network of vertices where the number of vertices are reduced to a specified limit which retains a skeleton structure of the environment. This is done through checking whether the edge between vertex i-1 and i+1 collide with any obstacles, and potentially deleting vertex i, should the tested edge be collision free. Another approach to reducing the vertices is discussed in [6], where a topological graph is generated using a 3D voronoi-based skeletonization method. This approach leverages the features of the surround surfaces to generate a graph which fills out the environment quite well.

Direct Point Cloud A point cloud consist of a list of point measurements, that by itself is quite difficult to apply to path planning due to the computational load it generates. Generally it is desirable to perform some sort of pre-processing and condense it down to a more easy to compute format. However, there are path planning approaches that utilizes sparse point clouds to plan directly. An example of such a planner is detailed in [28], here an target-oriented 3D path planner using a variation

of RRT in an outdoor environment is considered. The results showed that planning using point cloud data is feasible and can be effective given the correct use-case.

3 Method and Setup

3.1 Path Planner Selection

In this section a method for how to narrowed down the wide selection of path planners is going to be explained. The first step towards this is going to be to define some criteria the path planner has to fulfill. These are going to be based on the use case an what is practical given project's time frame.

- The path planners has to be available as packages, ready to be downloaded and applied to the robotics task at hand.
- The path planner has to be compatible with UAVs operating in 3D.
- The path planner should be able to generate a path from A to B.
- Support for ROS Melodic on Ubuntu 18.04.

These criteria are formed on the basis of what ScoutDI's use case and UAV requirements. The reason for only considered complete packages for the testing, is that there is no time for making path planners from the ground up. This is especially true when considering that there are countless of features that can be added on, which transcends the search algorithm itself. In a lot of cases these features are essential for achieving good performance in confined space.

The path planners chosen through this method are going to be the ones tested through the testing method described in 3.2.

3.2 Test Setup

This section is going to describe the test setup and define a procedural test for the chosen path planners to undergo, as well as define what performance measures that are going to be considered.

To set the path planner packages up so that the testing can performed, only necessary modification will be made. To avoid creating a "new" path planner approach from the ones being tested, only the available planning features will be utilized. However, potential features to improved the path planner for the use case at hand, may be discussed.

3.2.1 Platform

The testing is going to be performed on Ubuntu 18.04.6 LTS ROS melodic. This is partly because ScoutDI's AirSim based simulator is compatible with this operating system and ROS, and the initial plan was to test the path planner and simulator along side each other. However, it was later decided against due to confidentiality issues. The main reason however is that a lot of packages support this platform, which is beneficial when testing a variety of path planners.

3.2.2 Frameworks

A universal framework for several different path planning approaches would be desirable. During the research stage several frameworks were discover and considered for testing the path planners. However most of them do not do what is required by the task at hand. A lot of frameworks focus solely on comparing the search algorithms, without being aware of the additional stages that can be added to improve them. A lot of the available frameworks for testing the algorithms are not able to accept point cloud data, but instead utilizes internal simplified environments.

Moveit for UAVs
Moveit is a framework designed for path planning/motion control in manipulator arms. The manipulator models are simulated in gazebo and the sensor data and interface is visualized in Rviz. This framework provides a lot of different planners by including OMPL, Pilz Industrial Motion Planner, STOMP, SBPL and CHOMP. It also provides an UI in RViz to change between the search-algorithms quickly. Since this framework is designed for manipulator arms and not UAVs it is not initially applicable for the use case at hand. However, it is possible to use it with UAVs through some modifications detailed in Wil Selby´s blog [19]. The problems with this approach is however evident when one try to set it up according to the blog. The blog is quite old, so a lot of included packages only supports ROS Indigo, Ubuntu 14.04. These platforms are getting quite old and are therefore seeing a declined in maintenance, which is causing a lot of problems when attempting the install. The issues introduced can probably be easily solved, however provided the age, no more time was spent on this.

PathBench
PathBench is a python based library for testing different search algorithms on various 2D and 3D maps. [7] is an article which introduces PathBench and compares

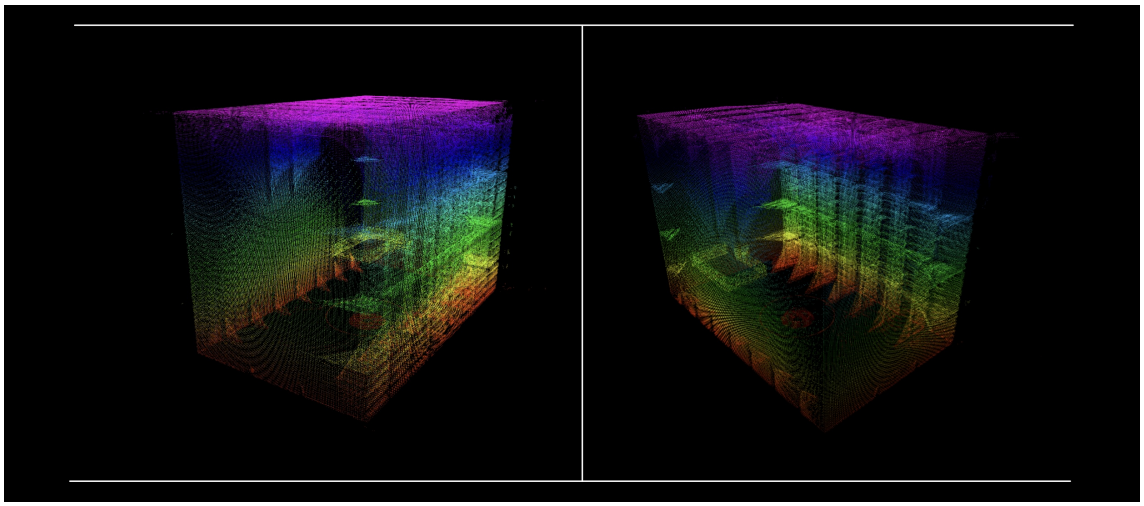


Figure 1: Surface point cloud of the ballast tank generated by GBplanner.

it to similar frameworks. It is unfortunately not able to import point clouds or the 3D models otherwise used for the testing. No easy workaround for introducing compatible with ros and voxblox was discovered as it is designed as a standalone testing program.

3.2.3 Maps

For the testing it is necessary to have relevant maps that at least relates to the use case. According to this, the selected maps should be of confined areas with various scattered obstacles. Another consideration is that different path planners require maps in different formats, it is therefore required that the maps are provided in the correct formats or at least can be converted cleanly. The planners are tested on the same maps to achieve comparable results for further analysis. For this thesis, due to time restrictions, and limited access to relevant maps only the following maps will be tested. Ideally the path planners should be tested on a bigger variation of maps.

Ballast Tank Map

The map in fig. 2 is of a generic ballast tank that was provided by ScoutDI. This map was provided in a few formats; point cloud (.pcd), 3D model (.fbx) and an OctoMap OGM (.bt). The map is quite large with a lot of open space, but at the same time it offers complex structures that has to be avoided. It correlates directly to the specific use case at hand.

Bookstore Map

The second map is a simplified version of the
textttaws-robomaker-bookstore-world [15] modelled by AWS-Robotics. It is an

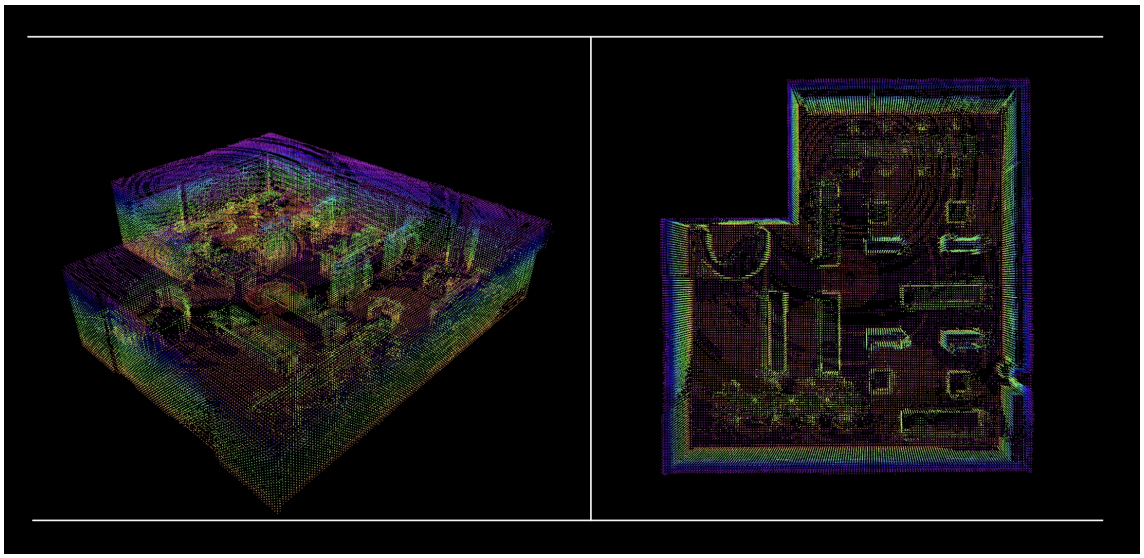


Figure 2: Surface point cloud of the simplified AWS bookstore map generated by GBplanner.

open-source 3D model package consisting of several models that can be added in individually. There are several smaller models which largely functions as minor details, these are left out to keep the complexity down. See fig. 2 for a general idea of the environment layout. As one may observe, the roof height is quite low and might cause some difficulties for the path planners.

The varying degree of clutteredness in maps are going to provide hints towards how each path planner performs in increasingly confined space.

3.2.4 Test Procedure

To measure the performance of the chosen path planners the planners are going to be tested for a set of pre-determined start and goal poses. For both planners the same map will be utilized. For graph based planners, the same graph will be used each time. The graph has not been adjusted to favor any of the goal poses.

For the ballast tank map a list of know POI provided by ScoutDI are going to be set as the goal, whilst the starting pose is going to be kept at the origin. The list of POI consists of 6 poses which will simply be referred to as POI1, POI2, POI3 and so on. The list cannot be included in the appendix due to confidentiality reasons.

In the bookstore map, non-occupied goal poses are going to be chosen and tested. For this map too, the starting pose will be kept the same for each test. The goal poses are going to be selected so that they will have varying of distance away from

the starting pose. The chosen goal poses are going to be kept the same for all the path planners tested.

A work flow is devised in 3.2.4. This is going to be the approach for testing all the POI on the testing maps, for all the path planners. Once all the data has been collected, the results are going to be compared and discussed in 4.

Work Flow:

1. Input start and goal pose
2. Run planner
3. Log the results
4. Reset
5. Repeat

3.2.5 Performance Measures

Between each iteration a set of metrics from the generated paths are going to be collected. These metrics are going to be used as performance measures for comparing the path planners later on.

The performance measures decided to be used for the testing have been inspired from the work performed in [14] and [13]. However, not all of these are deemed necessary when considering the use case. This is because some of these are inherent from the goal and approach of the path planners that are set up or are going to return the same results each time. This is why only the performance measures listed below are going to be included for the testing performed in this thesis.

- Computation time
- Path length
- Accuracy

Accuracy is going to refer to the distance between the final pose in the path and the actual goal pose.The accuracy metric will not consider error in orientation with respect to the POI, seen as this naturally will not matter if the position is off.

Another reason is that orientation error is not necessarily a path planning issue, and can often easily be corrected by using the correct controller, camera gimbal or wide lens camera.

3.3 Hardware and Software

3.3.1 Hardware

Most of the work performed for this master's thesis will be performed on computer 1. Due to travelling, the first installation and familiarization with the path planners was done on computer 2. The general consensus is that computer 2 is more powerful than computer 2. The full specifications of the computers can be found in the appendix (5).

3.3.2 Software

There are some main software packages and programs used during the testing process. Below is a list of the major software packages used for the testing:

- ROS-melodic
- Gazebo
- RViz
- Voxblox Ground Truth
- GBplanner
- MAV Voxblox Planning

RViz
This program is a tool for visualizing sensor data and other pre-processed data. RViz subscribes to the relevant topics published by the path planner and visualizes them according to the user's configuration profile. Through this software it is possible to choose what to include and what not. It is very useful to visually confirm that the path planner is generating a feasible path that is not way off with regards to optimality.

Voxblox

A ROS integrated package developed by ETHZ ASL [12], which is based on Truncated Signed Distance Fields (TSDFs). The TSDF maps are generated using the data from a point cloud. Voxblox has a feature for building Euclidean Signed Distance Fields (ESDFs) from TSDF maps. ESDF maps are better for path planning because they contain an explicit representation of the distance to the nearest surface. This is in contrary to TSDFs, which expresses this implicitly, making the path planning using these maps more complex.

4 Results and Discussions

4.1 The Chosen Path Planners

When exploring which path planning packages were available, very few packages proved to be valid according to the defined criteria. This limited the variety of path planners that could be tested for this thesis without going the way of coding them from the bottom or using a framework. The final path planner packages that were chosen was; GBplanner and Voxblox RRT planner. GBplanner's global planner utilizes a graph map representation and dijkstra's algorithm to plan, while Voxblox RRT planner uses a grid based map representation with RRT* algorithm.

4.1.1 GBplanner

GBplanner is an exploratory path planner designed for both UAVs and ground vehicles. It was developed by the Autonomous Robots Lab with DARPA Subterranean Challenge in mind. It consists of a local part, which is used to explore the drone's immediate surroundings and set trajectories towards areas with the most "volumetric-gain" ([2]) and a global planner which consists of a sparse graph generated as the drone explores. This global graph is intended to be used during exploration to plan a path from the current "frontier" ([2]) back to its starting position, should the battery be empty, or an alternative frontier that was tagged earlier during the exploration.

It relies on an TSDF map generated from a point cloud by Voxblox. The TSDF map contains distance information about the environment stored in voxels, which is used by the planner to make paths. The path planner utilizes this representation to sample the vertices and make sure they are not located within a obstacle. Then the edges are tested for collision and added to a graph, along with a weighted value indicating the distance away from an obstacle. When the graph is complete, dijkstra's algorithm may be applied to generate an optimal path to the goal. The local part only considers a workspace with a set of active vertices and edges. While the global planner saves a sparse global graph while it is exploring to form a skeletonization of the map. GBplanners supports functions for planning globally to either a set waypoint or a homing position. It does so by checking which vertex on the global graph is closest to the waypoint and plan to that.

The path planner package includes a simulator based in gazebo while the sensor

and planning visualization is provided by Rviz. This simulator uses a 3D model of an environment to simulate surfaces of the real world. The UAV utilizes a virtual sensor to sample the simulated environment to generate a point cloud of the environment, which is then used to generate the TSDF map.

Why?
The intended use case of this path planner is UAV exploration of various cave systems, so with regards to not colliding with obstacles in confined spaces, this planner is considered as a good alternative. The planner is compatible with ROS-melodic and has functions for planning paths to waypoints using a generated global graph. As so, it fulfills the criteria defined in 3.1.

This package contains a local and global planner, for target-oriented planning the global path planner is of interest. The global planner utilizing a graph that along with Dijkstra's search algorithm should provide safe and fast planning. While the package as a whole provides room for expanding the use case, for example if the environment was unknown and the initial task was to map it.

Installation Notes
The installation of GBplanner is normally quite straight forward. However, for the testing performed in this report, the first time the installation process was attempted, the provided `Aerial Robot Demo` would not launch. This was because of an upstream error with the `xacro` version in ROS-melodic, causing the urdf model to not load and crash. This was at the time solved by changing to the devel branch of ROS-melodic. In this branch a working version of `xacro` was utilized, which allowed for the urdf models used in the demo to be loaded properly.

Importing an Environment
When using the simulator included in GBplanner it requires a Gazebo world to be loaded. This is because the UAV is constantly sampling the environment with the simulated 3D lidar sensor to generate a point cloud which is used by Voxblox to generate the TSDF map. The package is supplied with some example worlds, consisting of different cave systems which can be modified. In the `Aerial Robot Demo` (rmf_sim.launch), `darpa_subt_final_circuit.world` is loaded as the default map, but this can simply be changed by adding the path to another gazebo world.

The ballast tank map required some modifications to be utilized with GBplanner. GBplanner in its simulation mode requires a gazebo world to be inputted, and since this was not provided, it had to be made from the 3D model. The first step was

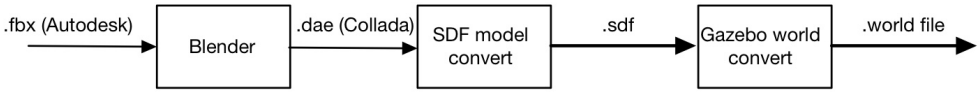


Figure 3: Process diagram of environment import.

to convert the 3D model (.fbx) to `Collada mesh` (.dae) using `Blender`. Then the Collada mesh could be made into a gazebo model (.sdf) by following the instruction in [3]. Then the gazebo model could be included in a gazebo .world file, along with a line including the gazebo ROS plugin (see 5).

The same method for importing the bookstore map was employed, with the only difference being the bookstore map consisting of several gazebo models. See figure 3 for a progression diagram describing the process of converting the 3D model to a format that can be used with GBplanner step by step.

Planning from A to B

Through the research done for this report it is clear that GBplanner is not a path planner designed for target-oriented planning, but rather an exploratory path planner. However, it is possible to make it function like a target-oriented path planner. This section is going to go through how it can be done.

Once GBplanner is downloaded and compiled correctly, it is possible to launch the aerial demo through the command: `roslaunch gbplanner rmf_sim.launch`. This will load the UAV along with an "unexplored" world with some mesh generated from the sensor, depending on the proximity to surrounding obstacles. In this new world, no vertices has been generated, so trying to plan a path through setting a waypoint will fail due to the global graph not being generated yet. A workaround used for this is to have the UAV explore the space first in exploration mode, and once the space skeletonization is over, save the graph for future use. There are built in services in GBplanner that enables the global graph to be saved and then re-loaded on a later stage. This function greatly reduces time spent between resets, and makes the testing more consistent. See fig. 4 for the graph used in the testing.

There are several ways to set a waypoint in GBplanner. The way to set waypoints according to the GBplanner wiki is to use the Rviz tool "2D Nav Goal", which is a

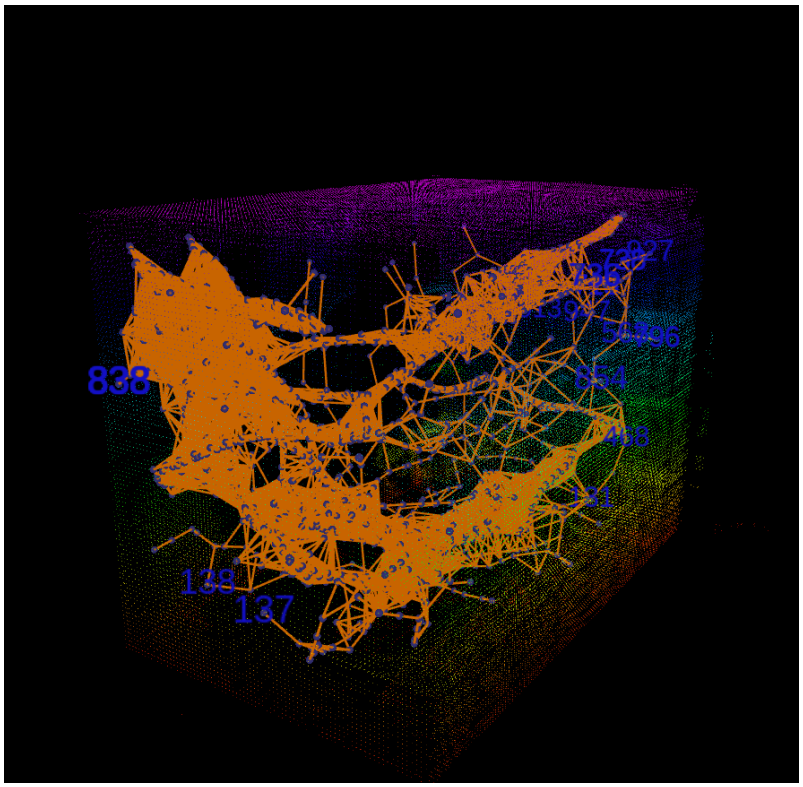


Figure 4: GBplanner graph generated for ballast tank map.

graphical tool that sets a waypoint on the surface of whatever the user uses the tool on within Rviz. A more exact way of setting a waypoint is to publish the desired pose directly to the `/global_planner/waypoint_request` topic. See the appendix for the specific command used (5).

Now that the waypoint is in the desired pose, the path planning can be initiated either through the Rviz UI button: "Plan to Waypoint" or by calling the service `/planner_control_interface/std_srvs/go_to_waypoint`, which is a trigger service that essentially executes the same commands as the Rviz button.

Finally the issue with GBplanner is that the waypoint provided by the user is not added as a vertex to the global graph. This means that the path planner can not plan to it exactly, because it got no way of knowing if there are any obstacles between the waypoint and the nearest vertex.

Due to inaccuracies in the final position w.r.t the provided waypoint, it is likely that the orientation will also be wrong, essentially meaning that the forwards facing axis of the actual pose does not intersect the POI (Point of Interest). This is not ideal since the camera frame may not be able to capture the POI when the UAV is in the position of the nearest vertex to the intended pose. However, the camera of

Table 3: GBplanner ballast tank test results

Metric	**POI1**	**POI2**	**POI3**	**POI4**	**POI5**	**POI6**
Computation time [sec.]	0.4328	0.3438	0.6080	0.3121	0.3855	0.4069
Path Length [m]	10.89	8.52	9.13	5.39	11.36	1.86
Accuracy [m]	1.44	0.59	0.32	0.43	0.62	1.09

the ScoutDI UAV has its camera on a gimbal allowing for orientation adjustments according to the actual position of the POI. This will not be investigated for this report.

Test Procedure

GBplanner did not providing the path length, so a simple python program (5) which utilized the outputted path to calculate the path length was made. In the same program a function for calculating the accuracy of the path planner by comparing the final pose in the path to the actual inputted goal pose was also implemented. This program communicates with ROS and is subscribed to the topic where the path is outputted. It also allows for the goal pose to be assigned and the planning to be triggered through publishing to topics and calling service proxies. This approach significantly sped up the testing, and can easily be expanded on.

Testing Results

For the testing a modified version of the provided aerial demo was utilized. This demo contains a UAV urdf model by name `rmf_obelix` along with a cave world generated from individual pre-made sections. The map was changed according to the steps detailed in fig. 3.

Unfortunately the graph generated from the *bookstore map* did not yield a graph that would work. So it was only possible to get results from the ballast tank map. From the results in table 3, the accuracy measurement indicated that there is a clear difference between the requested goal pose and final pose in outputted path. This was expected to be the case, but it is not considered to be a big issue. Because it is assumed that the issue can be easily resolved by running a local planning from the final pose to the actual goal pose. This would require minimal modification to be implemented.

Overall

The biggest issue with GBplanner is that it does not plan to a specific pose, only to the nearest vertex of a given goal. Another functional issue with GBplanner is that it does not generate a uniform graph, which results in varying accuracy. On

the other hand, the setup process is quite simple and fast. It also provides a lot of configuration which can be changed to fit the relevant scenario.

Considering everything, it would not take a lot of work to modify this path planner package to work well for the use case at hand. Gbplanner utilizes a sparse graph to plan to the nearest neighbor of the goal pose. With the addition of a feature that allows the desired pose to be added as a vertex in the graph, in theory the same collision checking methods used for generating the global graph vertices in the first place may be used. This should achieve an accurate path plan to the desired pose, without sacrificing computation time or safety.

4.1.2 Voxblox RRT Planner

The open-source package; `mav_voxblox_planning` is developed by `ethz-asl` and is available on Github [20]. As of June 2022, the last update was 3 years ago. The package utilizes the OMPL package, which forms the foundation for the various different path planners included. It does however not include a simulator and only supports voxblox maps (TSDF, ESDF). Despite of it not having a simulator integrated, it publishes paths the generated path to RViz which can be used to verify that the safety measure are good enough to avoid collision. It includes a few different path planners both for local and global planning. The included planner, which is being considered for this use case, is Voxblox RRT Planner.

Why?
This path planner uses RRT* to plan over an ESDF map of the environment. It is intended for planning paths for UAVs integrated with ROS-melodic. It is possible to designate a specific goal pose, which will be the actual final pose of the path. There is however a somewhat limited scale to the planning that can occur, since RRT methods generally struggle if the space is to big. This is especially true when accuracy is a consideration. The use case environment is an enclosed confined space, so in this case it should not be a problem.

Installation Notes
Detailed in this section is the authors experience with installing the package for the first time. These were written during June 2022, and are subject to change.

During the installation, there were some additional steps that had to be performed for the package to be successfully built. First, `protbuf_catkin` [5] had to be added

to the SRC folder manually, as this was not part of the repository. Secondly, `line 69` in `skeleton_io.cpp` had to be changed from `uint32_t tmp_byte_offset = 0` to `uint64_t tmp_byte_offset = 0`. The package also assumes that a functional OMPL is already installed. Important to note here is that the standard OMPL package for Ubuntu 18.04 will not result in an successful build, it is therefore necessary to install it from the development branch through: `sudo apt-get install libompl-dev` and `cd mav_voxblox_planning && git checkout devel/melodic_ompl && sudo apt install libompl-dev && catkin build`.

All in all, these issues were well documented and easy to resolve.

Importing an Environment

As its default, this package utilizes Voxblox TSDF or ESDF maps as its input. If it receives a TSDF map, it is able to generate an ESDF map from the TSDF. The availability of ESDF and TSDF is somewhat scarce, which somewhat limits which maps that can be easily inputted without modifying the package. However, by utilizing GBplanner to generate and save TSDF maps, the same maps used in GBplanner can be tested in Voxblox RRT Planner.

In an attempt to generate TSDF maps from an "objective source"; `Voxblox ground truth`, developed by `ethz-asl` as well, was considered. This package' purpose is to generate a TSDF maps from a gazebo worlds or PLY meshes. However, this approach turned out to not function too well. Unfortunately, an issue presented itself with the gazebo to TSDF map generation, which caused the Voxblox RRT Planner to not be able to start planning in any configurations due to "start pose being occupied". Many configurations was attempted, but non yielded any improvement. In the end it was concluded that non of the voxels within the map was labeled as free space, which resulted in there being no valid start or goal pose.

Eventually it was discovered an issue on `voxblox_ground_truth`'s github (see [16]), which mention the same issue. Here a solution which involved setting the parameter; `floodfill_occupied = true`. This solution was attempted for a gazebo world conversion, but it was discovered that this feature was unique to the PLY mesh to TSDF conversion part of the package. Since non of the maps being utilized for the testing was provided in PLY format, the maps were first converted to PLY mesh through Blender. This was followed by a realization that Blender does not utilizing the same PLY mesh format as the one supported by the PLY to TSDF converter in `voxblox_ground_truth`. A final attempt to resolve the issue was done through investigation the example files provided with package. Here it was discovered that

the program utilized to generate the example PLY mesh was *ZipPack* by Stanford [21]. This package was attempted installed, but it did not play well in Ubuntu 18.04, so in the end this approach of generating a TSDF map was abandoned.

Instead, GBplanner was used to generate and save the TSDF maps for use in the testing. GBplanner provided a service for saving the maps, which made the the process simple. Finally, the saved maps was simply added to the input path and the planner was able to utilize the maps for planning.

Planning from A to B

To enable path planning from A to B in this path planner, the process was much simpler than for GBplanner. Since this path planner is not initially an exploratory path planner, the task of assigning a start pose and goal pose was as simple as adding some interactive markers to RViz and placing them were you wanted. However an even simpler method was to utilize the planner service: `/voxblox_rrt_planner/plan`. This service took the start and goal pose as arguments and generated a path upon execution.

Test Procedure

The Voxblox RRT Planner node provided the path length and computation time as printed data in the terminal where the `RRT Planner Voxblox node` had been launched. To simplify the testing process a simple python program 5 was made and used. This utilized the `/voxblox_rrt_planner/plan` service and made it easier to change the goal pose between runs and allowed for quick testing. The results from the printed output were then simply noted down between runs. In addition to this, the paths generated were visually confirmed in RViz for any obvious collisions.
An issue with this path planner package is that there are were strict conditions for assigning the start and goal pose. This made it especially difficult to assign poses in very confined spaces. No documentation or available configuration files were provided for changing these conditions. As such the bookstore map was not possible to test seen as it was so cluttered. The POI of the ballast tank map was fine, so the work flow, as devised in 3.2.4, was simply followed for all the points.

Testing Results

The results (4) from this path planner indicated a slower planning process, which was expected. The accuracy was not relevant, seen as the goal pose is the condition which RRT* uses for determining if the path generation is done.

Overall

Table 4: Voxblox RRT planner, ballast tank test results

Metric	**POI1**	**POI2**	**POI3**	**POI4**	**POI5**	**POI6**
Computation time [sec.]	5.0337	5.0290	5.0340	5.0328	5.0361	5.0357
Path Length [m]	10.33	8.49	8.71	5.41	10.125241	2.47
Accuracy [m]	x	x	x	x	x	x

The results clearly indicates that this path planner is a path planning approach worth considering for the use case. It is a slower method than the one utilized in GBplanner, but it guarantees a nearly optimal path exactly to the goal pose. Seen as there is no time constraint involved in generating a path offline, this is not too big of a concern. It is worth mentioning that some paths lengths for certain POIs were longer for Voxblox RRT Planner compared to GBplanner. This is probably due to GBplanner not planning all the way to the goal pose, since the graph does not extend far enough. Some modifications may have to be made with regards to the start and goal assigning for this planner to function well in confined spaces.

4.2 Other Observations

An observation was made with regards to the difference between computer 1 and 2. Computer 2 has Intel UHD Graphics 630 CPU-integrated graphics, which according to UserBenchmark ([23] and [22]) is a considerably weaker GPU than the Nvidia GTX660ti. Despite of this computer 2 performed vastly better with regards to the visualizations of voxblox occupancy map in RViz. While computer 1 nearly froze up completely when enabling this function, computer 2 did not have any problems at all. This hints towards these visualizations actually being more CPU intensive than GPU. This will not matter to much when considering an actual implementation, but is wort mentioning for anyone attempting to recreate these experiments in the future.

Another observation that was made during this project was how much more difficult it is to implement the path planners in Windows 10. Due to travelling and limited permissions on the host computer, this was attempted as a means to get some work done while being gone from Trondheim. However, it resulted in a lot of time being wasted on finding workarounds to simple problems, that normally would be simple as a single command line on Linux. The difficulties was largely owed to packages not being available through *apt-get* (seen as this is a Linux package manager), which lead to having to use an alternative package managers such as Chocolatey. In Chocolatey, not all the necessary packages were available, and the ones that were did not necessarily correspond to the version needed. Overall, this

has made it clear why GBplanner and many other packages prefer Linux.

5 Conclusion and Future work

Through the work done in this report it has been made clear that there are much more to a path planner than its search algorithm, which is especially true for path planners in 3D. There exists many features for making the path safer, but these are not usually part of the path planning algorithm itself. These have to be added as an additional layer after the base planner has been implemented. During this thesis the basic aspects of what to consider when choosing a path planner approach for a specific use case has been discussed. Through this, it has provided a way of navigation the field of path planners in a way that emphasise the use case and its requirements. The thesis has also investigated the approach of adapting already existing path planning packages and applying them for a specific use case. Based on the results and analysis of the available material it has been made clear that it is quite difficult to adapt an already existing path planner package to work for a certain scenario.

The two planners tested in this report were GBplanner and Voxblox RRT Planner. These were two packages with different initial use cases. Through slight modifications it was possible to utilize them for the use case considered in this thesis. GBplanner was quite inaccurate, but provided very fast planning and was easy to install. While Voxblox RRT Planner was more difficult to install, but it was accurate and planned exactly to the assigned goal. Unfortunately, the Voxblox RRT planner was somewhat slower to plan the paths and made it difficult to assign goals in tight spaces. Overall the planners would need some more modifications to meet the exact requirements the use case presents, but for being adapted without any big modifications they performed quite well.

In conclusion, it was made clear that many path planners are not suitable for 3D applications due to the increased computational complexities. While other planners are simplifying the map representation too much costing it the required accuracy. It is therefore important to balance the different aspect of a path planning approach so that it fits the use case. It also worth considering if the use case is going to be expanded on later on, which will affect the approach with regards to for example if it should be real-time capable or not.

If any future work related to this thesis were to be performed, it would be of interest to provide more number-based results through testing of more path planning approaches on a wider array of maps. It would also be of interest to develop a better

taxonomy which separates the components in a more explicit manner. This would be with the goal of making it easier to differentiate between the use areas of the different path planning approaches.

References

[1] Shubhani Aggarwal and Neeraj Kumar. 'Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges'. In: *Computer Communications* 149 (2020), pp. 270–299. URL: https://www.sciencedirect.com/science/article/pii/S0140366419308539.

[2] 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.

[3] Gazebo. *Make a model*. URL: https://classic.gazebosim.org/tutorials?tut=build_model&cat=build_robot (visited on 04/07/2022).

[4] IGI Global. *What is Path Planning*. URL: https://www.igi-global.com/dictionary/an-adaptive-path-planning-based-on-improved-fuzzy-neural-network-for-multi-robot-systems/22026 (visited on 05/07/2022).

[5] Google. *protobuf*. URL: https://github.com/protocolbuffers/protobuf (visited on 27/06/2022).

[6] Masauki Hisada, Alexander Belyaev and Tosiyasu Kunii. 'A 3D Voronoi-Based Skeleton and Associated Surface Features.' In: *Proceedings of the Ninth Pacific Conference on Computer Graphics and Applications, 2001* (Jan. 2001), pp. 89–96.

[7] Hao-Ya Hsueh et al. 'Systematic Comparison of Path Planning Algorithms using PathBench'. In: (2022). URL: https://arxiv.org/abs/2203.03092.

[8] V. Jeauneau, L. Jouanneau and A. Kotenkoff. 'Path planner methods for UAVs in real environment'. In: *IFAC-PapersOnLine* 51.22 (2018). 12th IFAC Symposium on Robot Control SYROCO 2018, pp. 292–297. URL: https://www.sciencedirect.com/science/article/pii/S2405896318332634.

[9] K. Jiang, L.S. Seneviratne and S.W.E. Earles. 'Finding the 3D shortest path with visibility graph and minimum potential energy'. In: 1 (1993), 679–684 vol.1.

[10] Mehmet Korkmaz and Akif Durdu. 'Comparison of optimal path planning algorithms'. In: (Feb. 2018), pp. 255–258.

[11] Fangyu Li et al. 'Universal path planning for an indoor drone'. In: *Automation in Construction* 95 (2018), pp. 275–283. URL: https://www.sciencedirect.com/science/article/pii/S0926580517311184.

[12] Helen Oleynikova et al. 'Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning'. In: (2017).

[13] Rosli Omar, Che Ku Nor Hailma and Sabudin elia nadira. 'Performance comparison of path planning methods'. In: 10 (Jan. 2015), pp. 8866–8872.

[14] Apuroop Paleti. 'Performance Evaluation of Path Planning Techniques for Unmanned Aerial Vehicles : A comparative analysis of A-star algorithm and Mixed Integer Linear Programming'. In: (2016), p. 75.

[15] aws robotics. *aws-robomaker-bookstore-world*. URL: https://github.com/aws-robotics/aws-robomaker-bookstore-world (visited on 27/06/2022).

[16] saihv. *Maps generated using this repo not playing well with mav$_v$oxblox$_p$lanning*. 2020. URL: https://github.com/ethz-asl/voxblox_ground_truth/issues/3 (visited on 27/06/2022).

[17] Jose Sanchez-Lopez et al. 'A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments'. In: *Journal of Intelligent Robotic Systems* 93 (Feb. 2019), pp. 1–21.

[18] Luis Miguel González de Santos et al. 'Indoor Path-Planning Algorithm for UAV-Based Contact Inspection'. In: *Sensors* 21.2 (2021). URL: https://www.mdpi.com/1424-8220/21/2/642.

[19] Wil Selby. *3D Mapping Navigation*. Daniel: website was down when I tried to access it 30.06.2022, but I have the instructions for integrating a quadrotor simulator in moveit download as a pdf. If this of interest, send me a mail: daniel.v@hotmail.no. URL: https://www.wilselby.com/research/ros-integration/3d-mapping-navigation (visited on 06/04/2022).

[20] Wil Selby. *mav$_v$oxblox$_p$lanning*. URL: https://github.com/ethz-asl/mav_voxblox_planning (visited on 04/07/2022).

[21] Board of Trustees of The Leland Stanford Junior University. *The ZipPack Polygon Mesh Zippering Package*. URL: https://graphics.stanford.edu/software/zippack/ (visited on 14/07/2022).

[22] UserBenchmark. 'Intel UHD Graphics 630 (Desktop Coffee Lake i5 i7)'. In: *GPU benchmarks* (25th June 2022). URL: https://gpu.userbenchmark.com/SpeedTest/356797/IntelR-UHD-Graphics-630 (visited on 25/06/2022).

[23] UserBenchmark. 'Nvidia GTX 660-Ti'. In: *GPU benchmarks* (25th June 2022). URL: https://gpu.userbenchmark.com/Nvidia-GTX-660-Ti/Rating/2183 (visited on 25/06/2022).

[24] Liang Yang et al. 'A literature review of UAV 3D path planning'. In: *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)* 2015 (Mar. 2015), pp. 2376–2381.

[25] Liang Yang et al. ‘Survery of Robot 3D Path Planning Algorithms’. In: *Journal of Control Science and Engineering* 2016 (), p. 22.

[26] Yangwei You, Caixia Cai and Yan Wu. ‘3D Visibility Graph Based Motion Planning and Control’. In: ICRAI ’19 (2019), 48–53. URL: https://doi.org/10.1145/3373724.3373735.

[27] Christian Zammit and Erik-Jan Van Kampen. ‘Comparison of A* and RRT in real–time 3D path planning of UAVs’. In: (Jan. 2020). Is not available for the public, was requested by author.

[28] Zhaoliang Zheng, Thomas R. Bewley and Falko Kuester. ‘Point Cloud-Based Target-Oriented 3D Path Planning for UAVs’. In: (2020), pp. 790–798.

Appendix

waypoint_set command:

```
rostopic pub
/global_planner/waypoint_request geometry_msgs/PoseStamped
'{ header: { frame_id:  "/world"}, pose: {

position: {
x: 9.44593070333590000,
y: 4.11341416258736000,
z: 1.25682145595904000},

orientation: {
x: -0.02015746988173210,
y: 0.61617203326212100,
z: 0.02395873676155050,
w: 0.78698899659811500}}}'
```

Gazebo ROS Plugin:

```
<plugin name='ros_interface_plugin' filename='
   librotors_gazebo_ros_interface_plugin.so'/>
```

Hardware specifications

Computer 1:

- Model: Dell OptiPlex 7070
- Processor: Intel Core i7-2600
- RAM: 16GB
- Graphics: Nvidia Geforce GTX660ti
- Operating system: Ubuntu 18.04.6 LTS/Windows 10
- ROS: Melodic Morenia

Computer 2:

- Model: Dell XPS 8300
- Processor: Intel Core i7-9700
- RAM: 32GB
- Graphics: Intel UHD Graphics 630 (internal)
- Operating system: Ubuntu 18.04.6 LTS
- ROS: Melodic Morenia

Code used for testing GBplanner:

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


import rospy
from geometry_msgs.msg import PoseArray, Pose, PoseStamped
import time
import numpy as np
from planner_msgs.srv import *
import sys
from std_srvs.srv import *
from std_msgs.msg import String
import tf

# Extracts the data from the data recievedfrom the topic
def callback(data):
    global path
    global start_time
    print("Path Recieved!")
    comp_time = (time.time() - start_time)
    print("The planning took sec:", comp_time)
    comp_time_str = "\nComputation_time:" + str(comp_time)
    with open('/home/daniel/testing/results.txt', 'a') as file:
        file.write(comp_time_str)


    path = data.poses

    #with open('path.txt', 'a') as file:
            #file.write(str(path))

    path_length = 0
    for i in range(len(path) - 1):
        position_a_x = path[i].position.x
        position_b_x = path[i+1].position.x
        position_a_y = path[i].position.y
        position_b_y = path[i+1].position.y
        position_a_z = path[i].position.z
        position_b_z = path[i+1].position.z

        path_length += np.sqrt(np.power((position_b_x -
```

```
    position_a_x), 2) + np.power((position_b_y- position_a_y), 2)
    + np.power((position_b_z- position_a_z), 2))
41
42      print("Path Length:", path_length)
43      time.sleep(2)
44
45      with open('/home/daniel/testing/results.txt', 'a') as file:
46              file.write("\npath length:" + str(path_length))
47
48      accuracy()
49
50      #Subscribes to path output topic
51  def pathFetcher():
52      global start_time
53      rospy.Subscriber('/pci_command_path', PoseArray, callback) #
    call callback function when something is recieved
54      time.sleep(5)
55
56      rospy.wait_for_service('/planner_control_interface/std_srvs/
    go_to_waypoint')
57
58      start_time = time.time()
59      go_to_waypoint_req = rospy.ServiceProxy('/
    planner_control_interface/std_srvs/go_to_waypoint', Trigger)
60      go_to_waypoint_req() #Call the service
61
62      print("waiting for path")
63      rospy.spin()
64
65  def goalPub():
66      global goal
67      goalPub = rospy.Publisher('/global_planner/waypoint_request'
    , PoseStamped, queue_size=10)
68
69      goal = PoseStamped()
70
71      goal.header.frame_id = 'world'
72      goal.header.stamp = rospy.Time.now()
73
74      #JSON file format
75      #x = -1.67580196512856000
```

```
76      #y = 1.67412108507590000
77      #z = -1.38534101485052000
78      #goal.pose.position.x = x
79      #goal.pose.position.y = -z
80      #goal.pose.position.z = y
81
82      #Converted into ROS geometry_msg/pose
83      goal.pose.position.x = -6.408
84      goal.pose.position.y = -0.994
85      goal.pose.position.z = 0.3
86      goal.pose.orientation.x = 0.0
87      goal.pose.orientation.y = 0.0
88      goal.pose.orientation.z = 0.0
89      goal.pose.orientation.w = 1.0
90
91      print(goal)
92
93      rate = rospy.Rate(1)
94      #while not rospy.is_shutdown():
95      for i in range(2):
96          goalPub.publish(goal)
97          time.sleep(1)
98      print("Goal Published!")
99
100     pathFetcher()
101     #rate.sleep()
102
103     #rospy.init_node('talker', anonymous=True)
104     #rate = rospy.Rate(10) # 10hz
105     #while not rospy.is_shutdown():
106
107 def accuracy():
108     global path
109     global goal
110
111     last_pose = path[len(path)-1]
112
113     position_a_x = last_pose.position.x
114     position_b_x = goal.pose.position.x
115     position_a_y = last_pose.position.y
116     position_b_y = goal.pose.position.y
```

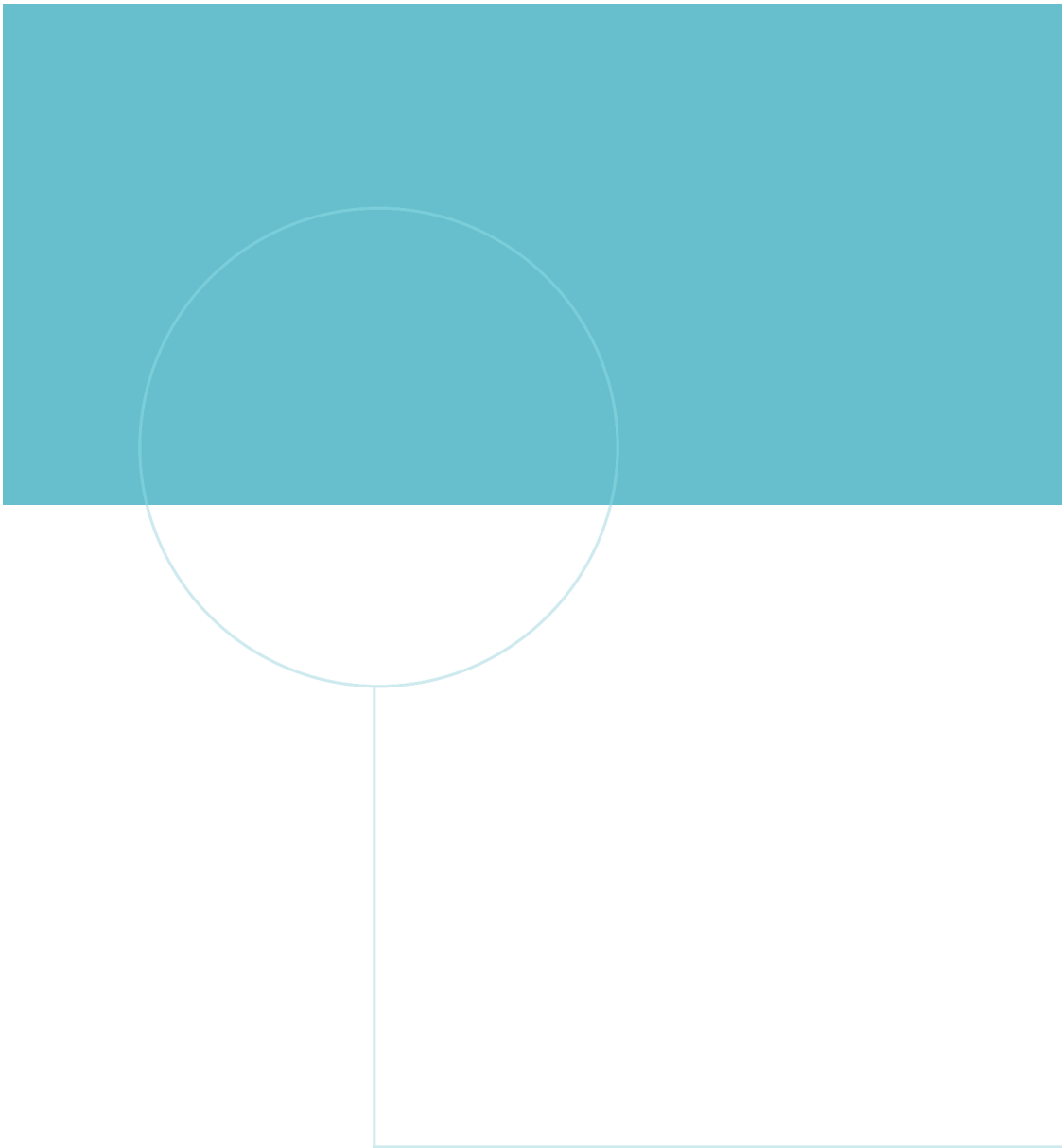
```
117     position_a_z = last_pose.position.z
118     position_b_z = goal.pose.position.z
119 
120     rest = np.sqrt(np.power((position_b_x - position_a_x), 2) +
    np.power((position_b_y- position_a_y), 2) + np.power((
    position_b_z- position_a_z), 2))
121     print("The final pose missed the goal by:", rest)
122     with open('/home/daniel/testing/results.txt', 'a') as file:
123             file.write("\nAccuracy:" +str(rest) + "\n")
124     rospy.signal_shutdown("Done")
125 
126 if __name__ == "__main__":
127     try:
128         rospy.init_node('pathRunner', anonymous=True)
129         goalPub()
130         #timer()
131     except rospy.ROSInterruptException:
132         print("Det gjekk t helvete! :8")
133         pass
```

Code used for testing Voxel RRT Planner:

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

from termios import VEOL
import rospy
from geometry_msgs.msg import PoseArray, Pose, PoseStamped,
    Point
from mav_planning_msgs.srv import *
import time
import numpy as np
import sys
from std_srvs.srv import *
from std_msgs.msg import String
import tf

def goWaypoint():
    rospy.wait_for_service('/voxblox_rrt_planner/plan')

    go_to_waypoint_req = rospy.ServiceProxy('/
    voxblox_rrt_planner/plan', PlannerService)
    go_to_waypoint_req(start,vel, goal, vel, box) #Call the
    service



def argDef():
    global start
    global goal
    global vel
    global box

    vel = Point()
    vel.x = 1
    vel.y = 1
    vel.z = 1

    box = Point()
    box.x = 200
    box.y = 200
    box.z = 200

```

```
38      start = PoseStamped()
39
40      start.header.frame_id = 'map'
41      start.header.stamp = rospy.Time.now()
42
43      start.pose.position.x = 0
44      start.pose.position.y = 0
45      start.pose.position.z = 0.5
46      start.pose.orientation.x = 0.0
47      start.pose.orientation.y = 0.0
48      start.pose.orientation.z = 0.0
49      start.pose.orientation.w = 1.0
50
51      goal = PoseStamped()
52
53      goal.header.frame_id = 'map'
54      goal.header.stamp = rospy.Time.now()
55
56      #JSON file format
57      x = -7.91403797569966000
58
59
60      y =  2.69889089686363000
61      z = -2.91218146021762000
62      #goal.pose.position.x = x
63      #goal.pose.position.y = -z
64      #goal.pose.position.z = y
65
66      #Converted into ROS geometry_msg/pose
67      goal.pose.position.x = x
68      goal.pose.position.y = -z
69      goal.pose.position.z = y
70      goal.pose.orientation.x = 0.0
71      goal.pose.orientation.y = 0.0
72      goal.pose.orientation.z = 0.0
73      goal.pose.orientation.w = 1.0
74
75      goWaypoint()
76
77  if __name__ == "__main__":
78      try:
```

```
79          rospy.init_node('pathRunner', anonymous=True)
80          argDef()
81          #timer()
82      except rospy.ROSInterruptException:
83          print("Det gjekk t helvete! :8")
84          pass
```



NTNU

Norwegian University of
Science and Technology