SURVEYING UNDERWATER SHIPWRECKS WITH PROBABILISTIC ROADMAPS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Amy Jeannette Lewis

June 2019

COMMITTEE MEMBERSHIP

TITLE:	Surveying Underwater Shipwrecks with Probabilistic Roadmaps
AUTHOR:	Amy Jeannette Lewis
DATE SUBMITTED:	June 2019
COMMITTEE CHAIR:	Zoë Wood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Aaron Keen, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Jonathan Ventura, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Julie Workman Computer Science Lecturer

ABSTRACT

Surveying Underwater Shipwrecks with Probabilistic Roadmaps

Amy Jeannette Lewis

Almost two thirds of the Earth's surface is covered in ocean, and yet, only about 5% of it is mapped. There are an unknown amount of sunken ships, planes, and other artifacts hidden below the sea. Extensive search via boat and a sonar tow fish following a standard lawnmower pattern is used to identify sites of interest. Then, if a site has been determined to potentially be historically significant, the most common next step is a survey by either a human dive team or remotely operated vehicle. These are time consuming, error prone, and potentially dangerous options, but autonomous underwater vehicles (AUVs) are a possible solution.

This thesis introduces a system for automatically generating paths for AUVs to survey and map shipwrecks. Most AUVs include software to set a lawnmower path for a given region of ocean, and individualized paths can be set via specifying GPS encoded nodes for the AUV to pass through. This thesis presents an algorithm for generating an individualized path that permits the AUV, equipped with a camera to "see" all sides of a region of interest (i.e. a shipwreck). This allows the region of interest to be completely documented. Photogrammetry can then be used to reconstruct a three-dimensional model, but a path is needed to do so. Paths are generated by a probabilistic roadmap algorithm that uses a rapidly-exploring random tree to quickly cover the volume of exploration space and generate small maps with good coverage. The roadmap is constructed out of nodes, each having its own weight. The weight of a given node is calculated using an objective function which measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. In addition, the weight of a node is increased if this node allows the AUV to see a new side of the region of interest. In each iteration of the algorithm, a node to expand off of is selected based off its location in space or its high weight,

a new node with a given amount of freedom is generated, and then added to the roadmap. The algorithm has degrees of freedom in position, pitch, and yaw as well as the objective function to encourage the path to see all sides of the region of interest. Once all sides of the region of interest have been viewed, a path is determined to be complete.

The algorithm was tested in a virtual world where the virtual camera acted as the AUV. All of the images collected from our automatically generated path were used to create 3D models and point clouds using photogrammetry. To measure the effectiveness of our paths versus the pre-packaged lawnmower paths, the 3D models and point clouds created from our algorithm were compared to those generated from running a standard lawnmower pattern. The paths generated by our algorithm captured images that could be used in a 3D reconstruction which were more detailed and showed better coverage of the region of interest than those from the lawnmower pattern.

ACKNOWLEDGMENTS

Thanks to:

- Mom, Dad, Hanna, Trent, and my family for always being my biggest fans. I love you.
- Zoë for being a great advisor and friend. I will never be able to thank you enough for the ICEX project and your constant support.
- Julie, Alex, and my SLO family for consistent love, confidence, food, and much needed fun.
- The graphics team - Kirsten, Tim, and Kole for encouragement, debugging, endless help, and laughs.
- My committee - Zoë, Dr. Keen, Dr. Ventura, and Julie for editing and much appreciated help.
- My friends - Tori, Athena, Clara, Kirsten, Erik, Liam, Jake, Austin, Sam, Bria, Martin, and Katie - for getting me through this crazy and wonderful year.
- The ICEX teams, specifically Chris Clark, Timmy Gambin, Roslyn Patrick-Sunnes, Jane Wu, Jeffrey Rutledge, Katie Davis, and again, Sam Freed and Zoë
- Bonita Galvan and Mitchell Keller for base code.
- Leanne for carrying the entire CSSE department.
- Scout, Charlie, Bailey, Olive, Tigger, and all the other good dogs for being dogs.
- Andrew Guenther, for uploading this template.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Almost two thirds of the Earth's surface is covered in ocean, and yet, only about 5% of it is mapped. Yes, there are topographies of the seafloor publicly available, but these are created by ambiguous satellite estimates. Robert Ballard, the oceanographer best known for re-discovering the Titanic, claims these estimates are like throwing a wet blanket over the table set for a dinner party. One might be able to make out the outlines of the candelabras, but not much else. No one would know that there are utensils, plates, or food hidden beneath the somewhat bumpy surface [11]. In other words, the satellite estimates can only provide a rough image of what hides on the seafloor.

There is an unknown amount of sunken ships, planes, and other artifacts hidden under the sea. Currently, searching for archaeological sites underwater involves many time consuming steps with expensive equipment. The first step is to select a large area to survey. While doing this, it is important to take into account the risk that the value of the site and the potential artifacts there could be damaged (e.g. fisheries, pipeline construction). Next, a side sonar scan can provide a high altitude survey. Third, the sonar images are analyzed by experienced people who then rank potential sites. There is normally only enough time and resources to revisit the highest ranked sites, so the ranking is significant. Then, sites are revisited by an accomplished human dive team or remotely operated vehicles (ROVs) to confirm or deny its value [16]. The last, and most important step, is data collection from the site. The data can be used to map the site and create reconstructions for further research. This method still leaves many of its steps up to humans to complete, and human time is expensive in addition to being error prone and risky.

A potential safer and less expensive solution is the use of Autonomous Underwater Vehicles (AUVs). New advances in robotic hardware have led to AUVs that, in many

aspects, exceed the performance of human dive teams. AUVs can dive thousands of meters deep for many hours, while human dive teams are normally limited to 100 meters. Also, the advances made in sonar imaging technology allow for higher-resolution images to be taken of the seafloor by AUVs. More importantly, AUVs can dive without risk to human life. Along with advances in software, much of the survey process described above can be automated by AUVs [5]. By leaving this repetitive step to technology, archaeologists and others interested in what hides on the seafloor will have more time to focus on the content of their research.

Most importantly, AUVs are excellent for data collection. Detailed 3D reconstructions can be created from the images collected at a site. The reconstructions can be shared and examined to help motivate the care and study of these historically significant sites. Dr. Timmy Gambin from the University of Malta would like to apply this to studying the decay of these sites. If a site is mapped once a year, decay can be tracked to help decide on issues like site maintenance, fishing and shipping rules, and if recreational diving is affecting the site.

1.1 Photogrammetry

AUVs can be equipped with a camera to collect video data. This video data can then be split into a series of images to be used in photogrammetry. Photogrammetry is the process of generating a digital 3D model by taking measurements from a set of images. Photogrammetry software, like Agisoft PhotoScan®, is used to align the photos, create a dense point cloud, build a mesh, and then build a texture. The resulting reconstruction can then be exported. In order to create accurate reconstructions, it is important to have overlapping images from multiple different views of every aspect of the site. If the collected images allow photogrammetry to successfully produce a detailed reconstruction, that 3D model can be shared and used to represent the shipwreck in studies.

1.2 Path Planning

For an AUV to survey a site, it must be given a path to follow. Assuming the side sonar scan has provided low resolution scans of the region of interest, it is possible to compute a bounding box for the volume where the potential shipwreck is. Then, a path that allows the AUV to effectively "see" all sides of the bounding box containing the shipwreck is needed. Paths can either be hard-set or created in real-time with intelligent path planning. The lawnmower pattern or a basic circle pattern are examples of commonly used pre-computed paths. Intelligent path planning algorithms include Dijkstra's shortest path, A*, the fast-marching based algorithm, and the probabilistic roadmap algorithm.

According to Yamafune et al., the most effective way to collect data is by following a lawnmower pattern [24]. In a lawnmower pattern, data is collected by moving repeatedly up and down looking down from above the site. Due to the repetitive nature of the path, image alignment for photogrammetry is high. For Yamafune et al. there was always above 90% of images aligned. Yamafune et al.'s work is discussed in more detail in Chapter 3. While the lawnmower pattern gets great coverage of the top of the shipwreck, it does not visit the sides of the region of interest. The standard lawnmower pattern is a built in pathing option for AUVs, but it is not sufficient. If images are not collected of the sides of the shipwreck, the reconstruction will also lack the sides and be incomplete.

Intelligent path planners are dynamic and offer the option to influence paths towards more specific goals - like collecting images of all sides of the region of interest. The probabilistic roadmap algorithm uses a rapidly-exploring random tree to quickly cover the volume of exploration space and generate small maps with good coverage. View coverage can be determined by casting rays from the camera view (whether it is a camera on the AUV or a virtual camera) and intersecting them with the region of interest. The probabilistic roadmap algorithm can then automatically generate a path with good coverage of the shipwreck by moving to regions that offer these views.

1.3 International Computer Engineering Experience

This thesis has grown out of the International Computer Engineering Experience (ICEX), a decade-long project between Harvey Mudd College and California Polytechnic State University, San Luis Obispo. For the ICEX project, robotics and computer graphics students from the two universities travel to Malta with an AUV. With the AUV they are able to gather video, sonar scans, and data for photogrammetry. The collected data is then used to create 3D computer graphics models of the found artifacts (e.g., shipwrecks and plane wrecks) that are put into a virtual underwater world.

Figure 1.1: Iver3 - the AUV used by ICEX [5]

Although the methods this thesis explores are general, ICEX is focused specifically on finding artifacts in the part of the Mediterranean sea surrounding the island country of Malta. Malta's rich history includes the Tarxien temple site, one of the oldest freestanding structures on Earth, initially built between 3,600 and 3,000 BC [7]. Aspects of this civilization's relationship with the sea is depicted in the Tarxien temple with fish and ship images (c.3000-2500 BC) [12]. This means to archaeologists concerned with underwater artifacts, the window of potential wrecks covers 3000BC to the present.

The abundance of the Mediterranean sea around Malta is not made up of just ancient trade ships, but is also scarred by the Second World War. Modern estimates suggest that 707 aircraft were lost in the Mediterranean theatre from the British Royal Air Force alone. For the Axis power aircraft, estimates range from 1,129 to 1,252 destroyed [2]. This destruction leaves thousands of artifacts for archaeologists to recover. In 2017, the ICEX team made a historically significant discovery of a wreck - a Fairey Swordfish (an Allied WWII airplane) shown in Figure 1.2.

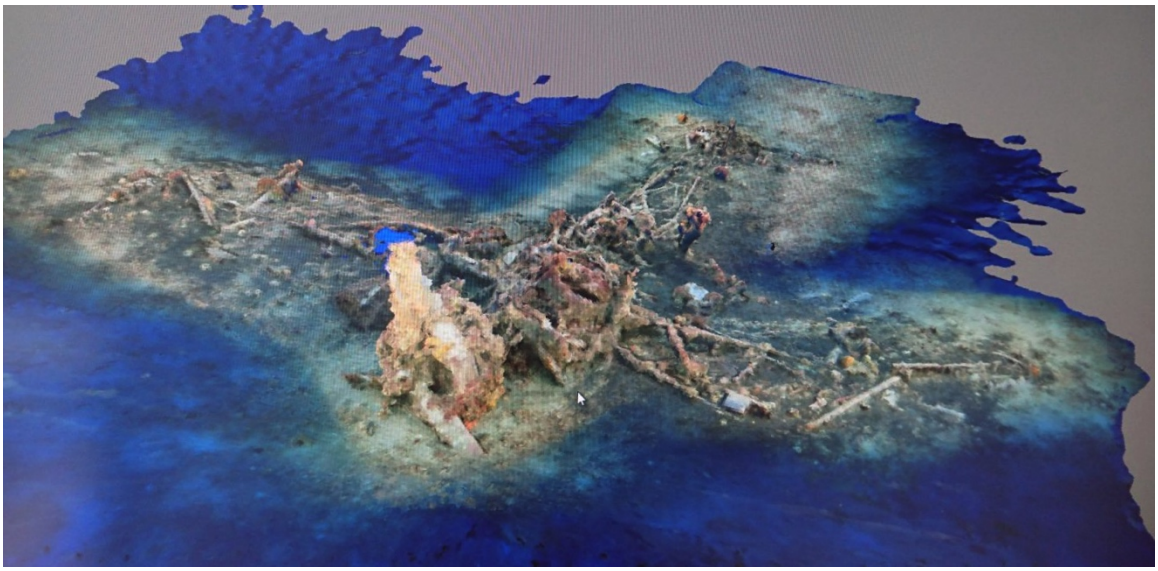

Figure 1.2: 3D Reconstruction of the Fairey Swordfish [5]

1.4 Contributions of this Thesis

This thesis explores a path planning algorithm used to map the shipwrecks given an identified region of interest. While AUVs are autonomous by definition, they do need to be given a path to follow in order to explore potential archaeological sites. Instead of just following a set path, AUVs would have a better chance of getting imagery while surveying if the path adapted to the site.

We have developed an intelligent path planning algorithm that takes into account the geometry of the site of interest to create a path for the AUV. The path is optimized to see all aspects of the region of interest with good visual coverage. Our algorithm can

automatically generate paths using a robotics motion planning algorithm, specifically the probabilistic roadmap (PRM). Small maps with good coverage are generated by rapidly-exploring random trees. Each node of the path is a potential viewpoint of the site, and is only added to the final path if it meets certain criteria (e.g., good view of the wreck, not in the same immediate location of previous nodes).

The algorithm was tested in a virtual world where the virtual camera acted as the AUV. All of the images collected from our automatically generated path were used to create 3D models and point clouds using photogrammetry. To measure the effectiveness of our paths versus the pre-packaged lawnmower paths, the 3D models and point clouds created from our algorithm were compared to those generated from running a standard lawnmower pattern. The paths generated by our algorithm captured images that could be used in a 3D reconstruction which were more detailed and showed better coverage of the shipwreck than those from the lawnmower pattern. A comparison of the side profiles of the original shipwreck 3D model used in the virtual world, the 3D model from the lawnmower pattern, and the 3D model from our algorithm is shown in Figure 1.3.

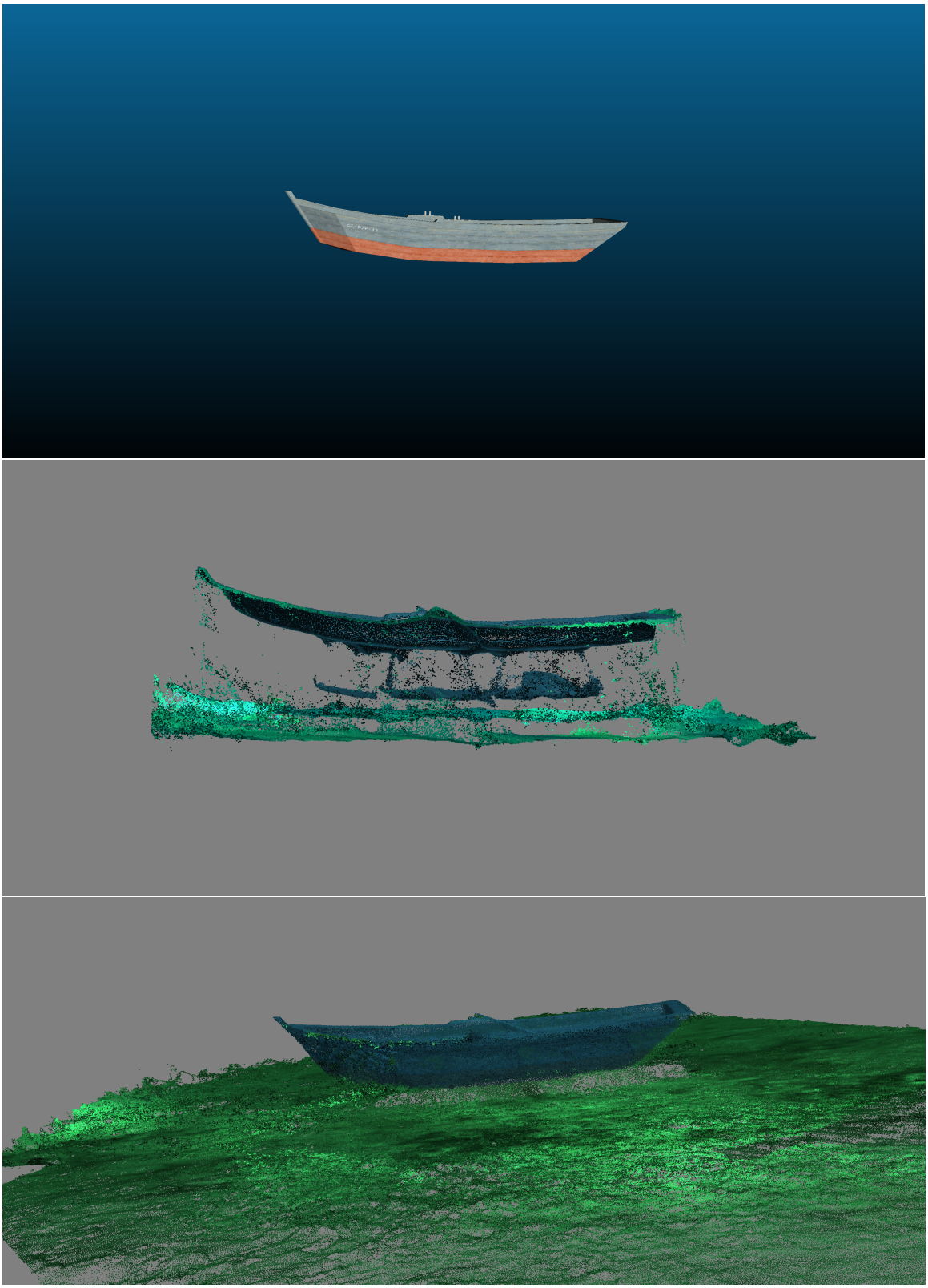

Figure 1.3: 3D Models from the Original Shipwreck, the lawnmower pattern, our algorithm

Chapter 2

BACKGROUND

The goal of this work is to automatically create paths for an AUV to travel through space and capture images of a site of interest which is represented by a bounding box in three-dimensional space. Our algorithm must support the way the AUV can move through space, thus we present the degrees of freedom for our system, background on path planning for robots, and photogrammetry.

2.1 A Robot's Degrees of Freedom

Degrees of freedom are used to define the configuration of an object, in this case a robot, in three-dimensional space. The x, y, and z of position represent the first three degrees of freedom. They exist is relation to the current coordinate frame that the robot is in. The pitch, yaw, and roll of orientation are the next three degrees of freedom shown in Figure 2.1 below.

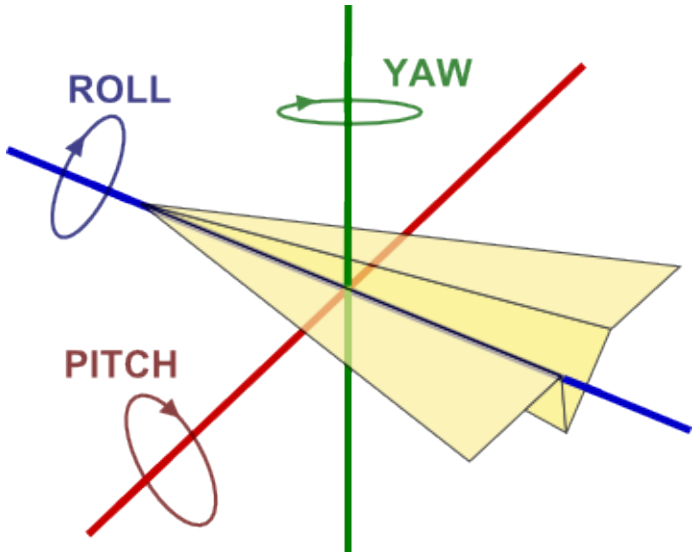

Figure 2.1: Three Rotation Actions of an Object in 3D Space [23]

Refer to Figure 2.1 for an explanation of the rotational degrees of freedom. We define the coordinate frame as follows: the x-axis is shown in red in Figure 2.1, the y-axis is shown in green, and the z-axis is in blue and is aligned along the length of the plane. Yaw is the plane turning left or right, or the rotation around the vertical y-axis. Pitch is the plane turning up into the sky or down towards the ground, or rotation around the horizontal x-axis. Roll is the plane rolling over its wings around the z-axis, so (hopefully) something only trained fighter pilots do.

2.2 Fundamentals of Path Planning for a Robot

The key problem in path planning is finding out if it is possible for a robot to move from one position to another while remaining in the configuration space, or "the space of all possible configurations of the robot" [4]. The invalid states, which the robot cannot occupy, are referred to as obstacles. A configuration itself defines the state of the robot, including its position, direction, and camera angle. To successfully navigate the configuration space, it is necessary to gather information on the following: robot perception, localization, and mapping. Robot perception is how the robot determines its configuration space and which parts of the environment have obstacles. Localization is how the robot determines its location in the configuration space. Mapping involves allowing the robot to determine its configuration space by building a representation of the its surroundings [9]. Solving for all of these elements together make it possible for a robot to navigate its configuration space.

In general, for robot path planning, the goal is to have a robot start at a location S and move to location T while remaining in the configuration space. Both the start and end locations also must be in the configuration space. If a direct path exists in the configuration space from S to T, then the robot should follow that straight line. If there is an obstacle in the way, the robot should follow the straight line drawn from S to T (ST) until it reaches the obstacle, circumnavigate it, return to ST at a point closer to the goal than where the robot encountered the obstacle, and then continue along ST once again moving towards the goal. If no point closer to the goal is found,

the robot can determine that there is no path to the goal T.

Algorithm 1: How the Robot Navigates its Configuration Space along ST [9]

Result: The Robot Navigates its Configuration Space along ST

```
visualize a direct path from the starting location S to the goal T;
while the goal T is not achieved do
    while the path ST to the goal is not obstructed do
        move towards the goal along the path ST;
        if the path is obstructed then
            mark the current location as P and circumnavigate the obstacle
             until the robot either:;
            (a) hits the line ST at a point closer to T than P and can move
             towards T, in which case the robot follows ST;
            (b) returns to P in which case T is unreachable;
        end
    end
end
```

While this Algorithm 1 does work, it is still necessary to have the robot's localization and mapping to generate complete paths.

2.3 Location of the Robot in Space

It is a common mistake to think that if the starting position of the robot and every move it has been instructed to make are known, then the location of the robot must also be known. This makes sense, but localization is not that straightforward. The real world is not so perfect. For every move the robot makes, an epsilon needs to be added to account for the small, random errors that are associated with motion [9]. Path planning for underwater vehicles introduces additional potential error factors

including rapidly varying currents and the fact that geolocation can only happen at the surface.

Therefore, if the the starting position S of the robot is:

$$(x_0, y_0)$$

then the location of the robot after N motions would be:

$$(x_N, y_N) = (x_0, y_0) + \sum_{i=1}^{N} (\Delta x_i, \Delta y_i) \tag{2.1}$$

2.4 Representing Space

Spatial decomposition allows the representation of space itself, instead of representing individual objects within it. A number of different subdivision methods can be used for sampling this space. By dividing the configuration space into regions, it can be more simply described as a grid. In two dimensions, these grids are sometimes known as pixel maps. In three dimensions, like real-world space and space in this thesis, the sampling elements are called voxels. No assumptions are made regarding object type, so the grid can represent anything - a classroom floor, a warehouse, or the ocean. However, the main disadvantage of this grid that is even when much of the environment is empty or occupied, fidelity is limited by cell size and the representation storage is intensive [9].

2.5 Path Planning in Unknown or Partially Known Environments

In the classic examples of path planners, knowing the environment in advance is a requirement. This allows the robot to calculate, plan its path, and then begin executing it. However, this is not accurate in the real world. At some point while the robot is executing its path it will most likely encounter an event that makes the path invalid. This causes the robot to replan, wasting the initial plan. Instead of starting over and replanning, repairing the plan can be more efficient. D*, Dynamic

A*, is an extension of the A* algorithm that allows path replanning when new events occur. Replanning the path involves searching the entire configuration space, and for complex spaces or complex robots with many degrees of freedom, this is not practical. Algorithms like Randomized Path Planner or Probabilistic Roadmap (PRM) can be used in these cases.

2.5.1 Probabilistic Roadmap (PRM)

For PRM algorithms, instead of sampling all of the configuration space, the space is sampled probabilistically instead. This is completed in two phases: the learning phase, where the roadmap is constructed in the configuration space, and the query phase, where probabilistic searches are conducted using the roadmap to accelerate the search.

In the learning phase, an undirected, acyclic graph is built in the configuration space where an edge connects two nodes if and only if a valid path can be found between the nodes, with nodes representing locations. To grow, random new locations are chosen in the configuration space and the algorithm attempts to make a path from this new location to one of the nodes already in the graph. The graph continues to grow until the path planner decides it is complete - for example, a certain path length or time limit is reached.

In the query phase, the search is sped up by using the roadmap. When a path is being built between two nodes, J and K, paths first are found from J to some node J' in the roadmap and from K to K'. The roadmap can then navigate between J' and K'. Following each query, the nodes and their edges that connect them to the graph are added to the roadmap [9].

Algorithm 2: Probabilistic Roadmap Algorithm [6]

Result: Path from starting node S to ending node T

```
R(N, E) = Roadmap(Nodes, Edges);
S = start configuration;
while the goal T is not achieved do
    select a random node x to expand from;
    randomly generate x' from x;
    if edge e from x to x' is collision free then
        R.add(x', e);
        if x' reaches the end T then
            return complete path;
        end
    end
end
```

The success or failure of the PRM algorithm is left to how random the different configurations generated are and the nature of the configuration space itself. For example, expanding into narrower regions of the configuration space is important as they are often high interest and offer access to other larger pieces of the configuration space [14].

2.5.2 Rapidly-Exploring Random Trees (RRT)

PRMs are suited for multi-query cases while rapidly-exploring random tree (RRT) handles single-query problems. RRTs make random selections in the path planning process to avoid examining all of the configuration space. A RRT grows in the configuration space starting from the initial position and expanding out until it reaches the goal, although there is no promise that there will be a solution. The roadmap is incrementally constructed as the search carries on following breadth-first search and A* [9]. Once a path is created from start to end by an RRT, the path often needs

to be smoothed by a path smoothing operation like cubic Hermite interpolation to make it operable for vehicles [8].

RRTs are a good approach for motion planning for several reasons. First, because the maximum size of the search is limited, local searches minimize exploring plateaus. They keep balance between exploration and exploitation during search. Also, only a relatively small tree must be kept in memory [1].

2.6 Photogrammetry

Photogrammetry is the process of generating a digital 3D model from images by making measurements from the images. Agisoft Photoscan®, takes images and uses photogrammetry to align them, generate a point cloud, and then a mesh and texture. The result of running Photoscan is shown in Figure 2.2. The images are taken from stills or video - either from a virtual camera moving through a virtual world or from a camera on an AUV. Framebuffer objects are objects in OpenGL that allow the creation of user-defined framebuffers. This makes it possible to render to locations other than the default framebuffer and, therefore, render without affecting what is displayed on the main screen. When capturing images in a virtual world, individual frames can be written out as images in a framebuffer object. The pipeline of taking this camera data and generating 3D models and point clouds is discussed in more detail by Seibert von Fock et al. [21].

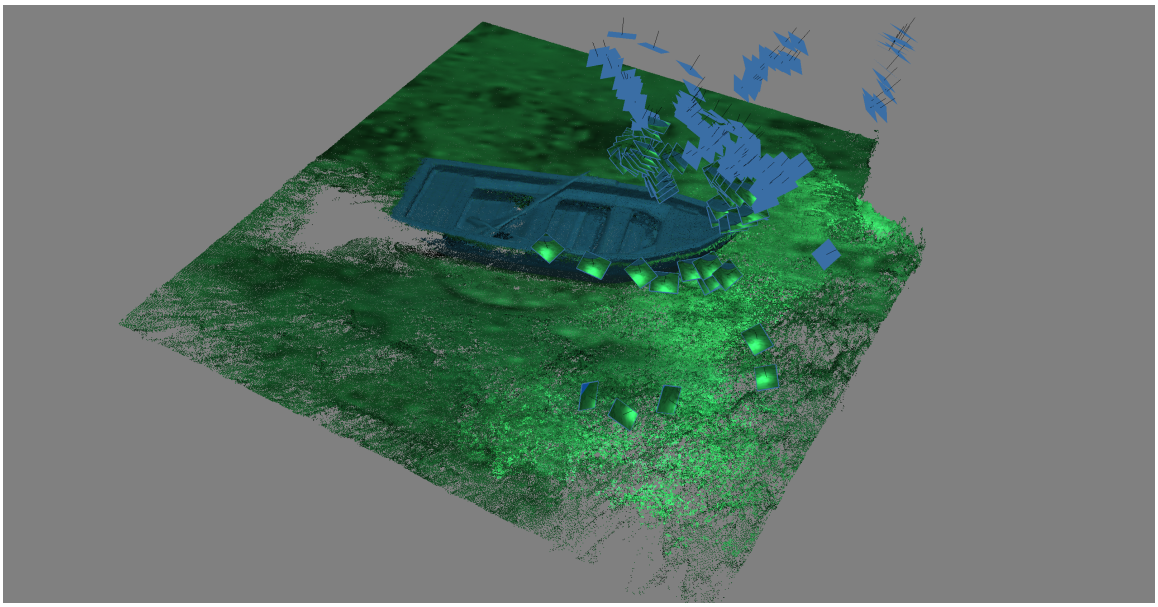

Figure 2.2: Result of Photoscan aligning the images to create a 3D model.

2.7 Summary

The process of generating 3D models of shipwrecks requires many steps. The robot needs to know both its configuration space and how it is located and orientated within it in order to navigate and gather data. A robot's degrees of freedom define it in 3D space. Path planning is finding out if it is possible for a robot to move from one position to another while remaining in configuration space. The probabilistic roadmap algorithm uses a rapidly-exploring random tree to quickly cover the volume of exploration space and generate small maps with good coverage. With a generated path that provides good view coverage, a robot can follow the path and collect video data. The video data can be split into images that can be aligned to generate 3D models with photogrammetry.

Chapter 3

RELATED WORKS

Path planning, or motion planning, for mobile robots is not a new idea. There are grid-based search algorithms, interval-based search algorithms, geometric algorithms, reward-based algorithms, sampling-based algorithms, and many more. The trade-offs of each algorithm and how they affect the final implementation are important to understand, as there is often no individual best approach to the problem of path planning for autonomous vehicles. Detailed below are some of the previously completed and popular methods used for path planning. Additional comments are also presented on the major differences between the related works and the algorithm implemented in this thesis.

3.1 Various Path Planning Algorithms for AUVs

Path planning for AUVs is not a new concept. Many planners use an algorithm called A* to find an optimal path over the configuration space. For example, Carroll et al. impose a regular grid and build a quadtree to represent collision-free space that they then use A* to navigate through [3]. Garau, Alvarez, and Oliver introduce a heuristic cost function that estimates the time the AUV would need to travel from one grid point to the next making their algorithm more dynamic. It is even able to take ocean currents into account [13]. Rao and Williams create a rapidly-exploring random tree (RRT) algorithm to plan collision-free paths for an underwater glider in 3D space [15]. Tan, Sutton, and Chudley also create a RRT algorithm to plan collision-free paths in three-dimensions while also accounting for vehicle dynamics [19].

Petres et al. account for vehicle dynamics as well in their path-planning algorithm with a fast marching-based approach. Their approach involves developing a fast-marching algorithm to efficiently find a continuous two-dimensional path in an

environment, accounting for ocean currents with an extension of the original fast marching algorithm, and introducing the AUV's constraints for optimal path curvature. This algorithm is great for finding optimal paths from a starting configuration to a goal configuration. However, this is not the point of this thesis. The goal of this thesis is to automatically create paths for an AUV to travel through space and capture images of all sides of a site. In other words, while the fast marching-based algorithm can find optimal paths, it is not necessarily good at creating paths that can get views of all sides of a shipwreck. We develop an algorithm that employs a probabilistic roadmap that uses a rapidly-exploring random tree to quickly cover the volume of exploration space in order to generate small maps with good coverage with a goal of seeing all sides of the site.

3.2 Probabilistic Roadmaps Algorithms for AUVs

Davis wrote a probabilistic roadmap planning algorithm to generate virtual camera paths for fly-throughs of a digital scene [6]. She uses cinematographic and geometric principles to determine if potential viewpoints belong in the final path. The 'rule of thirds' is the metric for cinematographic views and the normals of the 3D model relative to the camera viewpoint represent the geometric principles. Similarly to our algorithm, a RRT is implemented to quickly cover space generating small maps with good coverage. However, there are some differences as well. First, while she is planning for good viewpoints, the goal of this thesis is to create paths with good coverage. Yes, good viewpoints may provide good coverage, but the goal is not the same. Additionally, Davis' algorithm has degrees of freedom in only pitch and height as her virtual camera follows a roughly circular path. Our algorithm has freedom in both pitch and yaw, and has no given radius.

The research by Davis was continued the following year by Clark et al. [5]. The goal of this work was still to create virtual camera paths for fly-throughs of a digital scene, but with an additional degree of freedom - yaw. While the path was still given a radius to influence a circular path, it could rotate on the y-axis. This allowed the

paths to vary around the vertical axis, creating more random and interesting views from a cinematographic perspective. However, the goal of Clark et al.'s work was still to create good cinematographic and geometric views instead of getting good coverage while surveying a shipwreck.

Viswanathan et al. also implemented a motion planner with an RRT with a goal similar to ours - "to obtain images from multiple viewpoints of the wreck to enable off-board 3D mapping via photogrammetric reconstruction" [20]. In other words, they also wanted good coverage while surveying a shipwreck. However, there are differences in how they approach this solution. Most notably, in node selection. Node selection for this thesis is discussed thoroughly in Section 4.7, but in short, either a node is selected randomly from nodes that have a weight above a certain threshold or a node is selected if it is from a region that has been explored the least at that moment. Viswanathan et al. implemented node selection by either selecting a random node or selecting a node from a "high weight node" list. This list is made of previously grouped nodes that all have weights above a certain value, so in a way it performs similarly to the node selection above a certain weight threshold that this thesis adopts because nodes are guaranteed to have a high weight. How Viswanathan et al. determine node weight is significantly different than our objective function. Our objective function, as described in Section 4.3, measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. Viswanathan et al.'s objective function gives a high objective score to nodes correlating to areas with a high degree of elevation change. Also, like Davis's algorithm, in node generation, rotation is only free around one axis instead of two as in our algorithm. In the end, both the work of Viswanathan et al. and our own result in 3D models of surveyed shipwrecks.

3.3 Photogrammetry for Shipwreck Reconstruction

Yamafune et al. share the same goal in their work as our own - to survey shipwrecks for the purpose of reconstruction. Their pipeline involves "extracting, integrating, and

sharing archaeological information from tridimensional models" [24]. The differences between their work and ours is in how the data is collected. Instead of an AUV, Yamafune et al. use a team of professional human divers to capture their video data. Additionally, they follow a lawnmower pattern as shown in Figure 3.1 as opposed to our PRM generated path.

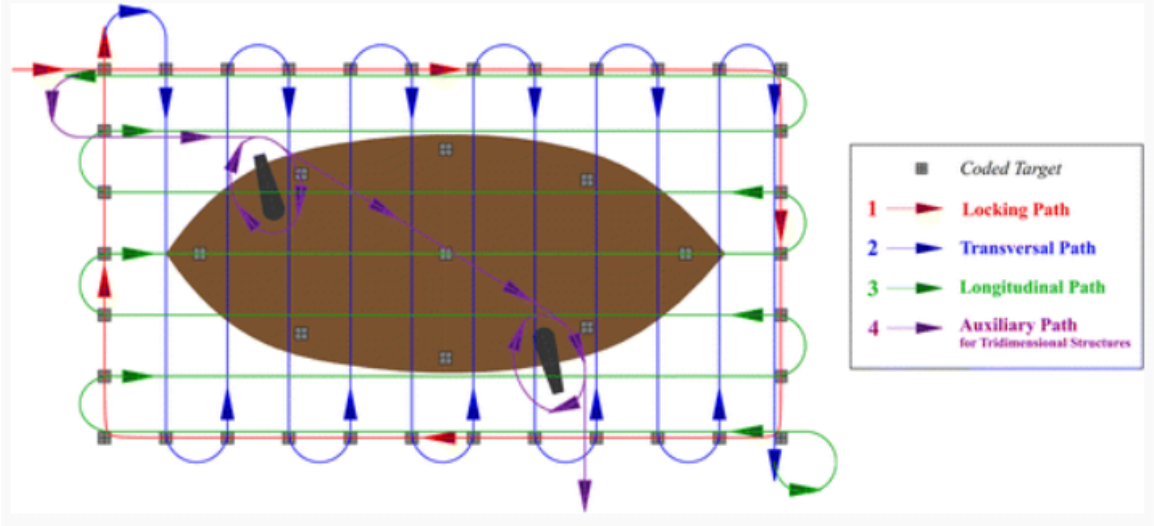

Figure 3.1: Yamafune et al. Lawnmower Pattern [24]

Images collected from following the lawnmower pattern were then inputted into Agisoft PhotoScan® to align the photos, create a dense point cloud, build a mesh, and then build a texture. The completed model can then be used as a reference to the actual shipwreck.

3.4 Next Best View Path Planning

Dunn et al. have also created a path planning for autonomous vehicles. With a goal of creating detailed 3D reconstructions, their paths a motivated to get the "next best view". Computer vision is used to obtain the geometric structure of the scene being reconstructed. The next best view is determined by their novel cost function that "quantifies the expected contribution of future viewing configurations" [10]. By using

a cost-driven recursive search of immediate viewing configurations, the path can move towards the next best view. Similar to our algorithm, once the path has determined that it has seen every view, it is complete.

Chapter 4

IMPLEMENTATION

This chapter details the implementation of the AUV path planning algorithm. Probabilistic roadmaps, a robotics motion planning algorithm, is used to create the paths as a rapidly-exploring random tree explores the configuration space. After a root node is generated and added to a newly initialized roadmap and spatial data structure, the process of node selection, node generation, and node weighting continues until a path is created. To create a path that fits the objectives of this thesis (i.e., viewing all sides of the region of interest and having large coverage of the region of interest), we use an objective function to evaluate any configuration with respect to these features. This objective function allows us to compute the weight for any node in the configuration space. More specifically, the objective function measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. In addition, the weight of a node is increased if this node allows the AUV to see a new side of the region of interest. While there are many potential paths, we want to explore the configuration space completely to find the best paths. To do this, our path planning algorithm uses a mix of randomly selected nodes with a high weight (with the weight measured using the objective function) and nodes located in less explored regions of the space. An overview of this is shown in Figure 4.1 below and then covered in detail throughout this chapter. After that, other information such as a description of the virtual world and implementation details are discussed.

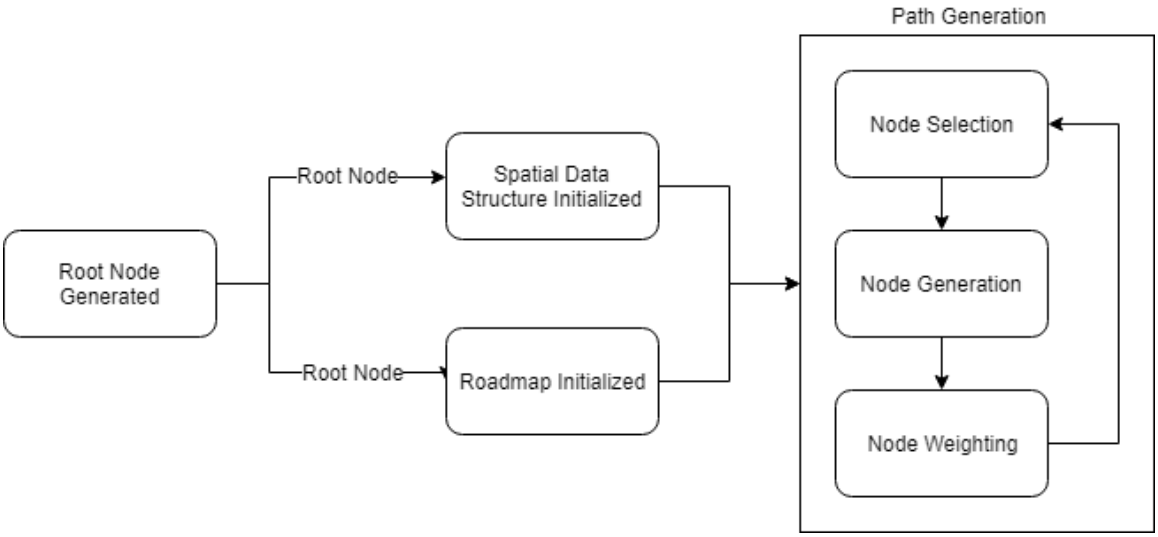

Figure 4.1: System Diagram of the PRM Path Generation Algorithm

4.1 Terminology

Much of the terminology was covered in the background (Chapter 2), but in this section it will be defined more specifically to this thesis.

4.1.1 Configuration Space

As stated in Section 2.2, the configuration space is "the space of all possible configurations of the robot" [4]. Here, the configuration space is a virtual representation of a general volume of ocean encasing the region of interest. In the real world, this region of interest is identified from low resolution side scan sonar data obtained from a high altitude scan by the AUV [16]. A bounding box is placed around the region of interest (i.e., the potential shipwreck), creating a constraint that keeps the robot from colliding with it as well as defining the region we want to capture multiple good views of for mapping and reconstruction. More specifically, the configuration space is the position and pitch-yaw space within the bounding box of the virtual underwater world.

4.1.2 Nodes

Nodes mark the potential robot configurations in the configuration space. Displayed in Listing 4.1 is the information that each node contains.

```
class Node
{
    vec3 position;
    Camera camera;
    float weight;
    int newSideSeen;
    vec3 velocity;
    int index;
    int parentIndex;
    int pathlength;
    unsigned int hit;
}
```

Listing 4.1: Node Class

Position of the node is a valid potential location for the AUV in the configuration space. The camera is oriented 90 degrees orthogonal to the AUV, thus, the view is orthogonal to the velocity as shown in Figure 4.2. In the real world, the camera is how data is collected. In this virtual world, the virtual camera represents the camera on the AUV. In other words, all of the views generated in the path and captured by the virtual camera are similar to what a camera on the AUV would collect. The potential views captured by the camera are a major part of determining the weight of a given node. The weight is high if it maximizes information gain and low if not. If a new side of the wreck is visible from a node that had not been before, the weight is increased as well. How weights are determined by the objective function is explained in more detail in Section 4.3. Velocity is what determines the magnitude and the direction

in which the robot moves towards the next node in the path. Index represents the ordered location of the node in the path. If a node's index is *n*, its parent index would be *n - 1*. Pathlength is the length of the current path generated by the PRM algorithm. Finally, hit is an unsigned integer that uses 5 bits to keep track of which sides of the region of interest have been seen so far. This is discussed in detail in Subsection 4.3.3.

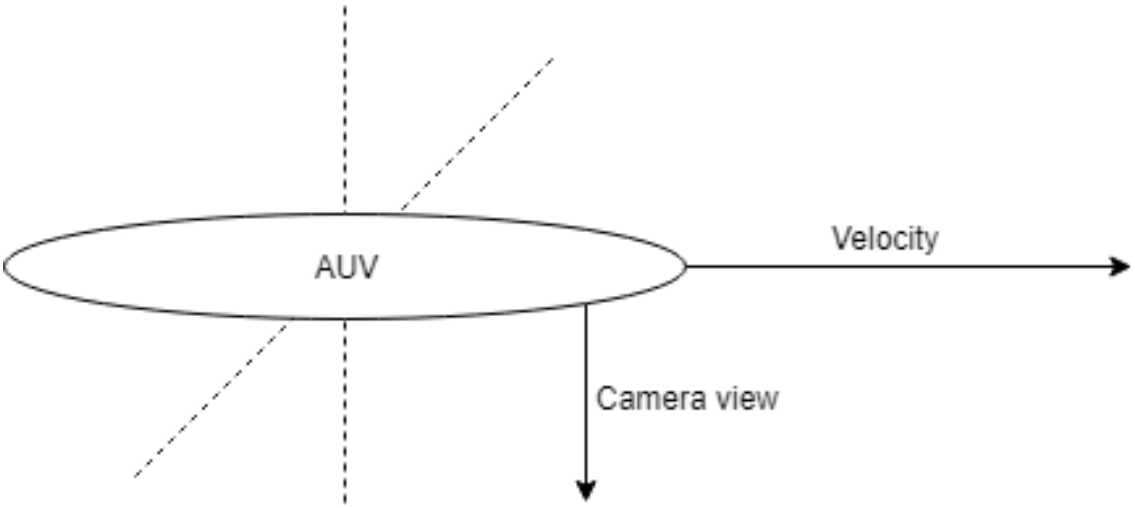

Figure 4.2: AUV configuration in space.

The starting configuration of the robot, the node where the PRM algorithm begins, is called the root node. The root node is the first node placed on a tree called a roadmap. The PRM algorithm populates the roadmap each time it runs. After a run of the PRM algorithm, the roadmap contains all of the generated nodes including those that will not be in the final path. The path is the selected set of nodes from the roadmap with configurations that complete the goal of seeing every side of the region of interest.

4.1.3 Spatial Data Structure

A spatial data structure is a division of three-dimensional space. They are often used because they are "compact and depending on the nature of the data, they save space

as well as time and also facilitate operations such as search" [17]. One type of spatial data structure, and the one implemented in this thesis, is a uniform spatial grid. As shown in Figure 4.3, it can be drawn as a 3D grid. The bounds of space used for creating our path planning algorithm is two times the bounding box of the region of interest, but it should be noted that this bound is configurable. Every element, or volume, of the grid is known as a voxel. Each voxel corresponds to a certain region of space. All nodes that can be found within that region in the virtual world are placed into the same voxel. In this way, it is possible to easily see which region in space has the least nodes, and expand the roadmap in that direction. This process of using the spatial data structure will be explained more in Section 4.6.

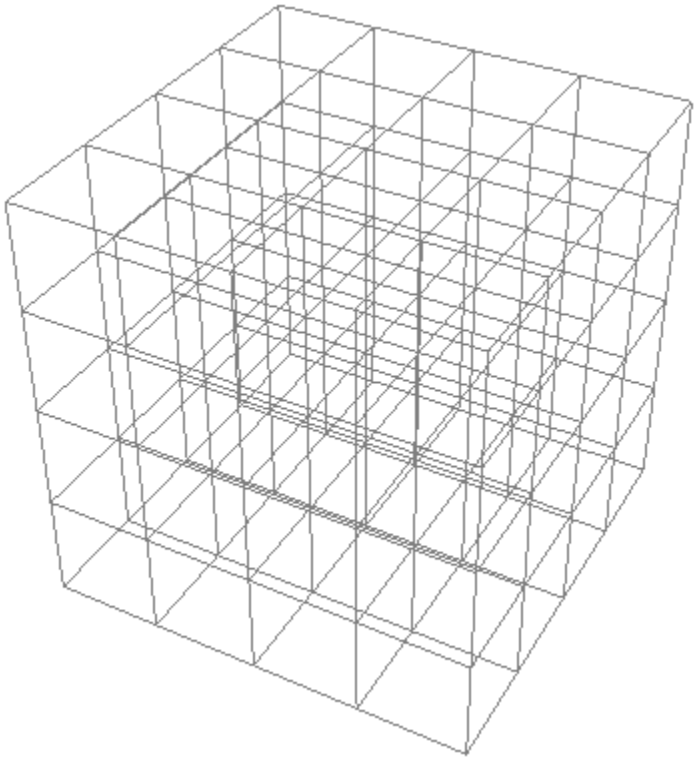

Figure 4.3: Three-Dimensional Uniform Spatial Grid [17]

4.2 Probabilistic Roadmap Algorithm

The PRM algorithm implemented in this thesis is given at a high level in Algorithm 3. It uses a rapidly-exploring random tree to explore the three-dimensional virtual configuration space. The configuration space, as described in Section 4.1.1, is the 3D position and pitch-yaw space in the virtual underwater world where the PRM can generate paths with variation in x, y, z, pitch, and yaw.

The goal is to see as much of the region of interest as possible by exploring all of potential space, checking that the views have good coverage of the region of interest, and adding nodes to a roadmap until all sides of it have been marked as seen. In Algorithm 3, the algorithm starts by creating a roadmap on Line 1 and a spatial data structure on Line 2. On Line 3, the root node is added to the roadmap to act as the root of the tree. The root node is also inserted into the spatial data structure. How the root node is selected is described in Section 4.5. How nodes are stored in voxels in the spatial data structure is detailed in Section 4.6. Lines 7 through 15 select which node to expand from as described in Section 4.7. On Line 16, a node is generated which is specified in Section 4.8. The new node is given a weight by the objective function on Line 18 which is discussed in Section 4.3. If the new weight exceeds the high weight threshold, the threshold is increased in order to maintain a high standard for weight. Finally, once all sides of the region of interest have been seen, the loop finishes and a path is outputted.

Algorithm 3: High Level PRM Algorithm for Path Generation

Result: A path through 3D space that visits all sides of the region of interest

```
RM(N, E) = RoadMap(Nodes, Edges);
SDS(P, V) = SpatialDataStructure(Position, Voxel);
Add rootNode to beginning of RM;
Insert rootNode into appropriate Voxel in SDS by its position;
while !allSidesSeen do
    // select node n to expand off of
    if even iteration of loop then
        n = getNodeFromVoxelWithLeast();
    else
        index = RM.size();
        while n.weight < highLevelWeightThreshold do
            index = index - 1;
            n = RM.at(index);
        end
    end
    randomly generate n' from n;
    calculate edge e from n to n';
    n'.weight = calculateWeight(n');
    RM.add(n', e);
    SDS.insert(n'.position);
    pathlength = n'.pathlength;
    if n'.weight < highLevelWeightThreshold then
        updateHighLevelWeightThreshold();
    end
end
```

4.3 The Objective Function

To create a path that fits the objectives of this thesis (i.e. viewing all sides of the region of interest and having large coverage of the region of interest), we use an objective function to evaluate any configuration with respect to these features. This objective function allows us to compute the weight for any node in the configuration space. More specifically, the objective function measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. There are several factors that contribute to a node's weight, and all of them are motivated towards the goal of getting good views of all sides of the region of interest.

Algorithm 4 shows the objective function discussed in this section. On lines 1 and 2, weight and the number of intersections are initialized to zero. The number of rays that are cast from the camera is set to 200 and the camera rays are initialized. Then, on lines 6 through 8, every ray is checked to see if it intersects the bounding box around the region of interest. If it does, the number of intersections is increased. This is detailed below in Subsection 4.3.1. On line 9, the weight is set to how many of the cast rays intersect the bounding box. Lines 10 through 17 can increase the weight if the node whose weight is getting set is not the root node. On lines 11 through 13, if the new weight is greater than the weight of the parent, the weight is increased. On lines 14 through 16, the weight is increased if any of the rays intersect a side of the bounding box that had not been intersected (or "seen") previously. This is discussed in more detail in Subsection 4.3.2. Finally, on line 18, the weight is returned.

Algorithm 4: The Objective Function

Result: The weight of the node.

```
1  weight = 0;
2  numIntersections = 0;
3  numRays = 200;
4  //initialize 200 rays from camera to bounding box
5  cameraRays = rays[numRays];
6  for ray : cameraRays do
7  |   numIntersections += rayBBIntersection(ray);
8  end
9  weight = numIntersections / numRays;
10 if parentIndex != -1 then
11 |   if weight > roadMap[parentIndex]->weight then
12 |   |   weight += (numIntersections / numRays) * 0.25;
13 |   end
14 |   if this->hit > roadMap[parentIndex]->hit then
15 |   |   weight += (numIntersections / numRays) * 0.25;
16 |   end
17 end
18 return weight;
```

4.3.1 View Coverage

Most of what contributes to a node's weight has to do with how much of the region of interest is being seen at that configuration. From the configuration stored in the current node, 200 rays are cast from the camera towards the bounding box that surrounds the region of interest. Figure 4.4 shows this in a simplified manner where the solid lines collide with the region of interest and the dashed line does not.

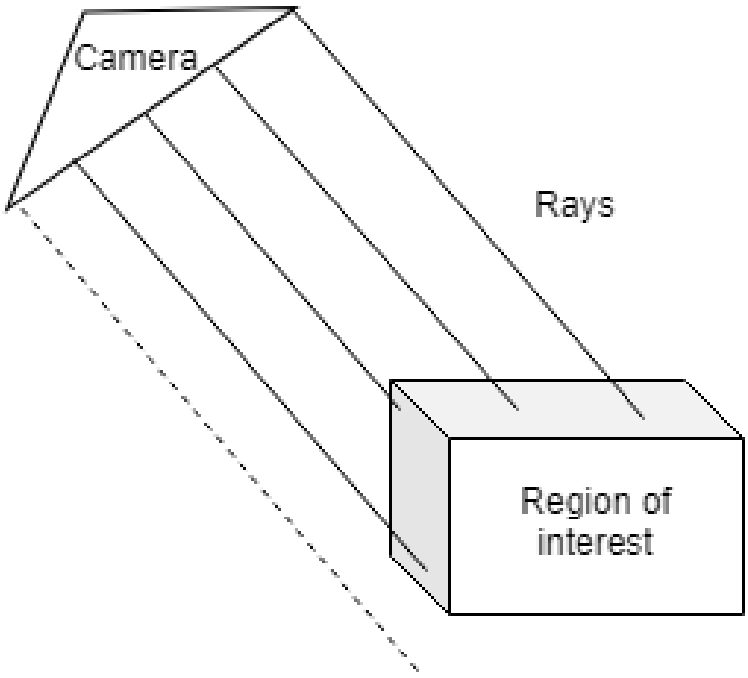

Figure 4.4: Rays cast towards the region of interest from camera to determine weight of node.

Every ray has an origin and direction as shown in Equation 4.1. The parameter **t** can be any real value and, by changing it, we can use it to define any point on the ray.

$$pt(t) = ray.origin + ray.direction * t \tag{4.1}$$

Each ray originates at the current position of the camera and ends when it collides with the bounding box around the region of interest or the far plane of the view frustum. The view frustum is the region of space in the modeled world that is in the field of view of the camera and may appear on the screen. The far plane represents the maximum depth visible in the scene where everything behind it is clipped. Each ray is oriented towards a random position on the far plane. This calculation is only made once so the direction of the rays from the camera to the far plane remains consistent as the camera moves through space. Then, for each ray, we use the camera's location

relative to the current node's position as the ray origin, and see how many of the rays collide with the bounding box on their way to the far plane.

This computation is completed for every ray and is detailed in Algorithm 6, but first, the logic behind it is explained in detail. For axis-aligned bounding boxes like the one encasing the region of interest, we us Equation 4.2 to find where the Ray r intersects one side of the bounding box.

$$r.origin.x + r.direction.x * t = bb.min.x \tag{4.2}$$

Reordering this we get **t0x** in Equation 4.3. We can solve for the bounding box's maximum extent for the x-component in a similar way as shown in Equation 4.4.

$$t0x = (bb.min.x - r.origin.x)/r.direction.x \tag{4.3}$$

$$t1x = (bb.max.x - r.origin.x)/r.direction.x \tag{4.4}$$

The minimum and maximum extents, also known as tValues, are calculated for the y-component and z-component as well. The problem then becomes finding which of these values corresponds to an intersection (if the ray intersects the bounding box at all). The tValues represent where the ray intersects the planes defined by each face of the bounding box (or the individual sides) as shown in Figure 4.5.

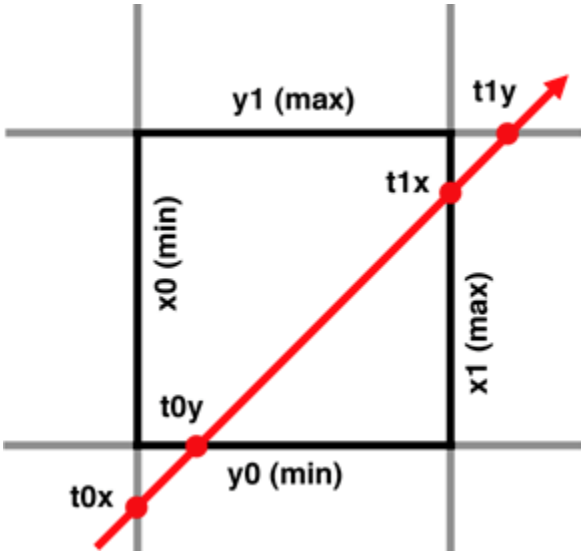

Figure 4.5: Where the ray intersects the plane at the tValues [18].

The ray first intersects the planes defined by the minimum extent of the bounding box - in Figure 4.5 this is t0x and t0y. While the intersections do occur on the plane, this does not necessarily mean they lie on the bounding box. We can find which of the tValues lie on the bounding box by comparing them as in Equation 4.5 for the minimum extent and Equation 4.6 for the maximum extent.

$$tMin = (t0x > t0y)?t0x : t0y \tag{4.5}$$

$$tMax = (t1x < t1y)?t1x : t1y \tag{4.6}$$

Figure 4.6 demonstrates that rays can intersect planes while missing the bounding box.

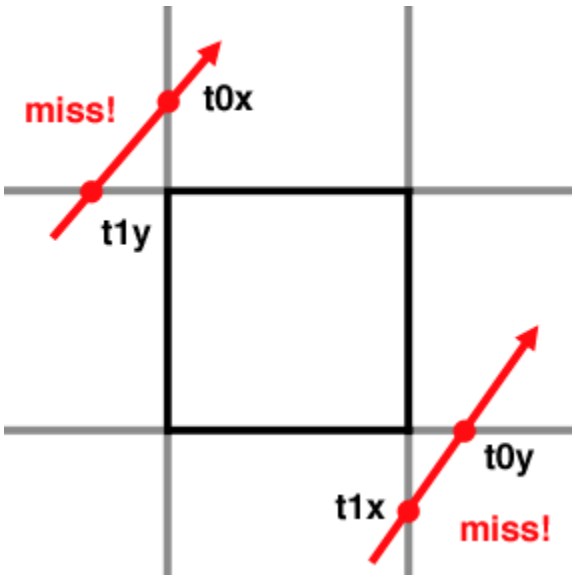

Figure 4.6: Rays may intersect planes but miss the bounding box [18].

This can be tested for by comparing the tValues as in Equation 5. If the test is true, we can automatically return 0 because we know there is no intersection and the ray is completely outside of the bounding box.

Algorithm 5: Ray Intersection

Result: If the ray misses the intersection with the plane.

```
1 if (t0x > t1y) || (t0y > t1x) then
2 |   MISS;
3 end
```

Otherwise, we extend this method to the third dimension [22]. The first seven lines of Algorithm 6 cover testing intersections with the x and y-components of the bounding box. Then, on lines 8 through 13, we reset the minimum tValue and maximum tValue to reflect the minimum and maximum for both the x and y-component. This can then be compared to the z-component on lines 16 through 18 in the same way we did in Equation 5. On line 19 we see which side was actually intersected by the ray. This will be discussed in more detail in Subsection 4.3.3. Finally, we return 1 so that Algorithm 4 can account for another ray intersection.

Algorithm 6: Ray Bounding Box Intersection Function

Result: Int value 1 if Ray r collides with BoundingBox bb or 0 if not.

```
1  float tMin = (bb.min.x - r.origin.x) / r.direction.x;
2  float tMax = (bb.max.x - r.origin.x) / r.direction.x;
3  float tyMin = (bb.min.y - r.origin.y) / r.direction.y;
4  float tyMax = (bb.max.y - r.origin.y) / r.direction.y;
5  if (tMin > tyMax) || (tyMin > tMax) then
6  |  return 0;
7  end
8  if tyMin > tMin then
9  |  tMin = tyMin;
10 end
11 if tyMax < tMax then
12 |  tMax = tyMax;
13 end
14 float tzMin = (bb.min.z - r.origin.z) / r.direction.z;
15 float tzMax = (bb.max.z - r.origin.z) / r.direction.z;
16 if (tMin > tzMax) || (tzMin > tMax) then
17 |  return 0;
18 end
19 bitEncodeSidesHit(ray);
20 return 1;
```

For each ray, if it collides with the bounding box, the integer value 1 is returned, 0 if not. The returned integer values are added up for every ray as shown on lines 6 through 8 of the Objective Function - Algorithm 4. The total number of intersections is then divided by the total number of rays to find a value for weight. Therefore, the more rays that hit the bounding box, the higher the weight.

4.3.2 Encouraging Exploration of Space for Greater Coverage

As the goal is to see all sides of the region of interest, if a node offers a view of a side that had not previously been seen, the weight is significantly increased as shown on lines 11 through 13 of Algorithm 4. It is explained in Section 4.7 on Node Selection that nodes with a weight above a certain threshold are selected for expansion. Therefore, increasing the weight of nodes that offer new views encourages expansion to continue in that new direction. How a ray determines if it has, in fact, collided with a new side is explained in Subsection 4.3.3. Also, as seen on lines 14 through 16 of Algorithm 4, if a node's weight is greater than it's parent's weight, it is increased to keep the idea of continuing to expand towards high weight nodes. In Figure 4.7, the sides of the region of interest in red have already been seen and the white sides have not. There is a ray colliding with a white side of the box, so the weight of the node would be increased.

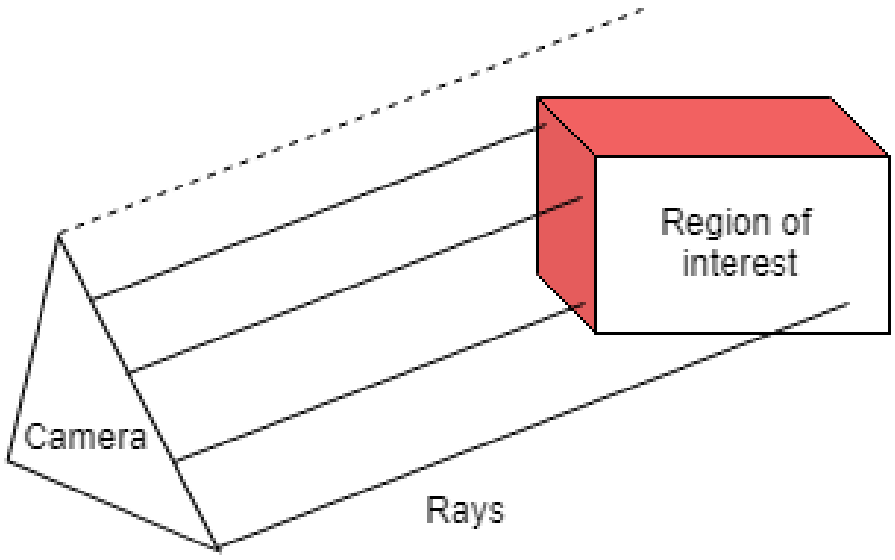

Figure 4.7: Rays cast towards area of interest from camera to determine weight of node.

4.3.3 Bit Encoding to Keep Track of Sides Seen by Camera

Due to the fact that there are 440 nodes, it would take quite a bit of space for every node to hold five boolean values indicating if each of the visible sides of the region of interest has been seen. Instead, each node holds an unsigned int where five bits are encoded which sides of the region of interest have been seen by the camera.

Algorithm 7 shows how this is implemented. Every visible side of the bounding box was given a general label (i.e. side1, side2, side3, side4, and side5). On lines 3 and 4, we calculate a float to see if the ray intersected the first side of the bounding box. If the float value if greater than 0, the side was intersected and, therefore, pushed back into the vector of intersected sides. This is reflected on lines 5 through 7. Lines 3 through 7 are completed in the same way for all of the sides of the bounding box. Then, if any sides were pushed back into the vector, we encode its respective bit to reflect that. Every side of the bounding box corresponds to a specific bit in the unsigned int *hit* value that each node has. Thus, on lines 8 through 25, we test to see if a side has been seen. If it has, we flip its respective bit. We also can set a bit if it's parent node has seen a side. This is how we can keep track of how many sides of the bounding box have been seen for any potential path.

Algorithm 7: Bit Encoding Sides Seen

Result: An unsigned int with its bits set corresponding to what sides of the region of interest have been seen.

```
1  vector<float> sidesHit;
2  //the following 5 lines are repeated for sides 2-5 as well
3  float side1 = bb.side1.normal.w - dot(r.origin, vec3(bb.side1.normal.x,
     bb.side1.normal.y, bb.side1.normal.z);
4  side1 /= dot(r.direction, vec3(bb.side1.normal.x, bb.side1.normal.y,
     bb.side1.normal.z);
5  if side1 > 0 then
6  |  sidesHit.pushback(side1);
7  end
8  if sidesHit.size() > 0 then
9  |  float t = tValues.at(0);
10 |  if t == t1 then
11 |  |  hit = roadmap[parentIndex].hit | 16;
12 |  end
13 |  if t == t2 then
14 |  |  hit = roadmap[parentIndex].hit | 8;
15 |  end
16 |  if t == t3 then
17 |  |  hit = roadmap[parentIndex].hit | 4;
18 |  end
19 |  if t == t4 then
20 |  |  hit = roadmap[parentIndex].hit | 2;
21 |  end
22 |  if t == t5 then
23 |  |  hit = roadmap[parentIndex].hit | 1;
24 |  end
25 end
```

4.4 High Weight Threshold Initialization

In Section 4.7 in the discussion of node selection, it was mentioned that every other iteration a node is randomly selected from the roadmap, but the node is not actually chosen unless it is above a certain weight threshold. The weight threshold exists to encourage the path towards high weight nodes, so the value is initialized quite high. To set it, the weight is calculated for the initial 440 nodes placed in the virtual world with the objective function discussed in Section 4.3. Then, both the average and standard deviation are found for these 440 weights. The high weight threshold is then initialized to the average weight plus the standard deviation.

4.5 Root Node Selection

The root node is important because it represents the starting configuration of the AUV where all generated paths and the roadmap begin. Using the known data of a region of space of high interest (i.e., the upper corner of the bounding box), the search begins with the camera looking directly at its center. This creates an optimal starting position, orientation, and high weight for the root node.

This is the chosen starting position and orientation for the root node, but the algorithm can start in any configuration. The different tested root node configurations include: a root node with a random position and orientation pointing towards the center and root node starting in the upper corner of the bounding box with random orientation. While these configurations did successfully create paths, they often took several minutes longer to do so. For the starting configuration with random orientation, the roadmap would expand in a direction away from the region of interest before correcting itself. The starting configuration with random position started so close to the bounding box that the roadmap had to first expand away from the bounding box before it could see other sides of it. While these were successful options, the starting configuration in the upper corner oriented towards the center of the region of interest performs best and, therefore, was selected for implementation.

4.6 Using the Spatial Data Structure

Each newly created node is inserted into the spatial data structure by its location. This is done by first converting the world coordinates of the node's position into index coordinates. The equation for this is shown in Equation 4.7 where *worldCoord* is the coordinate in world space, *minSDS* is the minimum value of the spatial data structure, and *voxelSize* is the size of the voxels that make up the spatial data structure. The *indexCoord* is the returned index coordinate which has x, y, and z values. This index coordinate is then used to insert the node into the correct location in the spatial data structure.

$$indexCoord = floor((worldCoord - minSDS)/voxelSize) \tag{4.7}$$

Inserting the nodes into the spatial data structure by their location is important because it is then possible to keep track of the direction in which the roadmap is expanding. With a goal of viewing the entire region of interest, it is important that the algorithm expands in all directions. Therefore, the voxel that contains the least amount of nodes is the direction in which the algorithm is encouraged to continue expanding. The roadmap would produce many nodes in a single area that offers high weight views of the region of interest, so this expansion serves to de-cluster it. This is part of node selection, which is discussed further in Section 4.7 below.

4.7 Node Selection

As mentioned in the discussion of Algorithm 3 in Section 4.2, lines 7 through 15 select which node to expand from. While, the main goal is to see all sides of the region of interest, the algorithm does not just want to expand totally randomly because first, this might take an extremely long time, and second, the views generated might not even be looking at the region of interest.

The spatial data structure is used to optimize expansion by ensuring the algorithm

expands towards a region that has been visited the least amount of times. As shown on line 8 of Algorithm 3, this is done by selecting a node from the voxel in the spatial data structure that contains the least amount of nodes. The spatial data structure is made up of voxels that contain the nodes with locations in the three-dimensional regions they represent. Thus, it is trivial to select a random node from the voxel that contains the least amount of nodes. This is done for every other node selected to encourage exploration for coverage of the configuration space.

For the other half of the loop iterations, the algorithm focuses on getting nodes with high weights. To do this, any node with a weight above a given threshold is selected for expansion. How the value for the threshold is initially set using the node weight average and standard deviation is discussed in detail in Section 4.4.

If the weight of the selected node is greater than the high weight threshold, the weight of the new node is added to the previously used node weight total and a new average and standard deviation are calculated. These values are added together to create a new, higher valued threshold to encourage even higher weight nodes for future selections. By expanding the roadmap towards higher quality nodes, the algorithm effectively prunes low quality directions and decreases the potential size of the roadmap.

4.8 Node Generation

After selecting a node as described in Section 4.7, a new node is generated based off of it. The velocity of the old node is taken and a new velocity with a different pitch and yaw is added to it. A delta value, within a range of 0 and π of the old node's pitch is randomly chosen. The new delta value is then added to phi. The same process happens for yaw with a delta and added to theta. This is shown on lines 5 through 8 of Algorithm 8. The new, semi-random pitch and yaw are then used to calculate the new velocity as seen on lines 9 through 13. On line 14, the new velocity is added to the previous position, creating the position of the new node. If that is, in fact, a

valid position and not colliding with the region of interest, then the orientation for the robot at that position is calculated as shown on lines 15 through 22. The node is then generated with degrees of freedom in x, y, z, pitch, and yaw. Having roll also be free would not make sense as AUVs cannot move in that way.

Algorithm 8: High Level Algorithm for New Node Generation

Result: A new position and orientation for the new node

```
1  while NOT new validPosition do
2      vec3 prevVelocity = parentNode.velocity;
3      float phi = asin(prevVelocity.y);
4      float theta = atan2(prevVelocity.x, prevVelocity.z) - PI;
5      float pitchDelta = random(0, PI);
6      float yawDelta = random(0, PI);
7      phi += pitchDelta;
8      theta += yawDelta;
9      vec3 newVelocity;
10     newVelocity.x = cos(phi) * sin(PI + theta);
11     newVelocity.y = sin(phi);
12     newVelocity.z = cos(phi) * cos(PI - theta);
13     newVelocity *= auvSpeed;
14     vec3 pos = prevNode.pos + newVelocity;
15     if pos NOT colliding with the wreck then
16         newNode.velocity = newVelocity;
17         newNode.position = pos;
18         vec3 up = vec3(0, 1, 0);
19         vec3 right = cross(newVelocity, up);
20         newNode.lookAt = cross(newVelocity, right);
21         validPosition = true;
22     end
23 end
```

4.9 Path Completion

Our algorithm terminates based a path on one of three things: a time limit, a path length, or if all sides of the region of interest have been seen. The time limit and path length options are only used for debugging purposes as the goal of this thesis is to create paths that see all sides of the region of interest in order to reconstruct a 3D model. Each time a node is added to the roadmap, the algorithm evaluates if all sides of the bounding box around the region of interest have been seen; when all sides have been seen the algorithm begins its termination sequence. After the final side has been seen, the algorithm runs for a finite amount of time (a random time between 10 and 15 seconds) to ensure there is good coverage of the final side. After the time limit is hit, we have a complete path. The time limit is an experimental choice and other methods to ensure further coverage of the final side of the region of interest could be explored. This is left to future work.

After the algorithm has explored the space and found a path that sees all sides of the region of interest, we need to extract the exact path from the rapidly-exploring random tree in order to use this path for the AUV trajectory (or in our case to replay a virtual trajectory to gather images for reconstruction). In order to extract the path from the tree, each node holds the index of its parent node. The path is reversed from that terminating node to the root node.

Then, important information for each node is written out to a text file so that the path can be replayed later as discussed in Section 4.10. For each node, its weight, position, and camera information are written out. The camera information is how the AUV knows its orientation at each position. Statistics like the roadmap size, average weight of nodes in the path, minimum weighted node, and maximum weighted node are also written to the file.

Finally, the virtual world is rendered to the screen. The positions are then interpolated between using a cubic Hermite interpolation function creating a spline [8]. The camera can then move along the smooth spline instead of jumping from node to

node. An image is taken every third frame by the virtual camera as it follows the spline. These images are all saved to be used for reconstruction later.

4.10 Replaying a Complete Path

If a path has completed and a file has been successfully output as discussed in Section 4.9, that file can be used to replay the path. Instead of generating a new path, if the command line arguments provided to the application specify the name of a previously generated path, the old path will be played back. Each node (with position and camera information) in the path is read in from the file and put into a vector. The positions are then interpolated between using a cubic Hermite interpolation function creating a spline [8]. The camera can then move along the smooth spline instead of jumping from node to node. Figure 4.8, below, shows the nodes that the path visits with each node representing a configuration of the AUV.

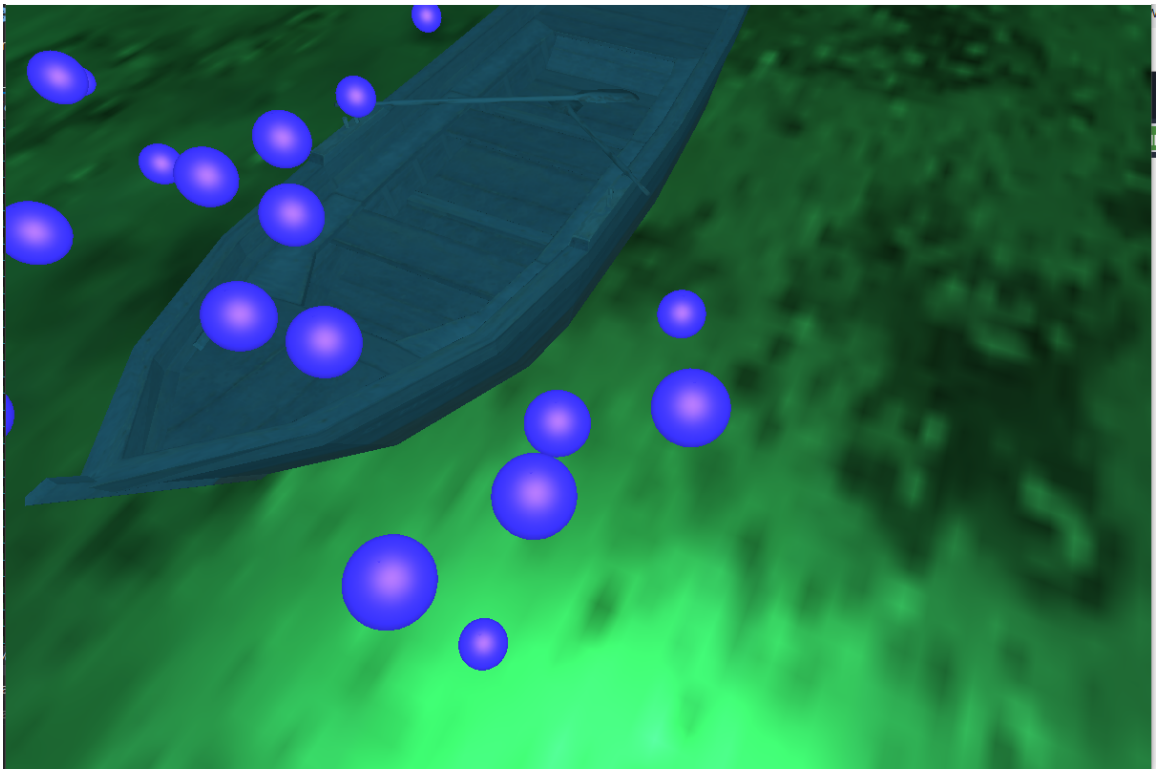

Figure 4.8: Each node (blue sphere) represents a configuration of the AUV while following the path.

4.11 The Virtual World

This thesis includes the algorithm to compute a path in 3D space given a region of interest and an application to test these paths via the creation of a virtual scene for the virtual AUV/camera to travel through. The virtual camera captures an image every third frame for use in reconstruction. This virtual world is an OpenGL application which in its current implementation includes simple geometric models. The ground plane represents the seafloor. It is textured with an image of a sandy seafloor like that of the Mediterranean. The texture does not repeat to make the captured images more unique so image alignment discussed in Chapter 5 is more accurate. There are no rocks as the Mediterranean is a sandy environment. Figure 4.9 shows a view of a 3D modeled shipwreck in the virtual world.

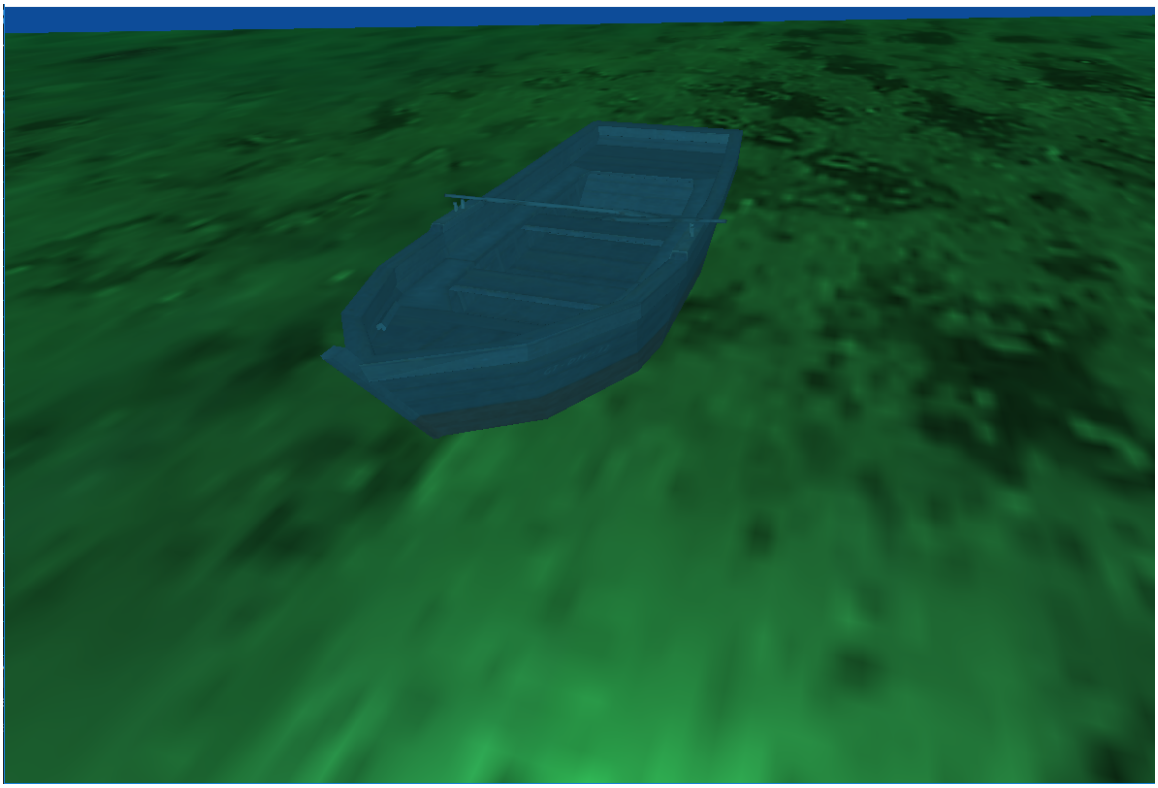

Figure 4.9: The Virtual World.

The 3D model shipwreck captured in Figure 4.9 is a simple "Old Boat" model.

The geometry of the wreck is loaded from an .obj file and the texture loaded from a .jpg. The texture is applied to the model via the texture coordinates. Other 3D models could be loaded in place of the old boat, and the algorithm would still be able to generate paths around it.

This thesis is not specific to any one model. The algorithm was run with multiple other 3D models including a 3D reconstruction of the "Bristol Beaufighter". Figure 4.10 is a frame captured from a path generated around the beaufighter model.

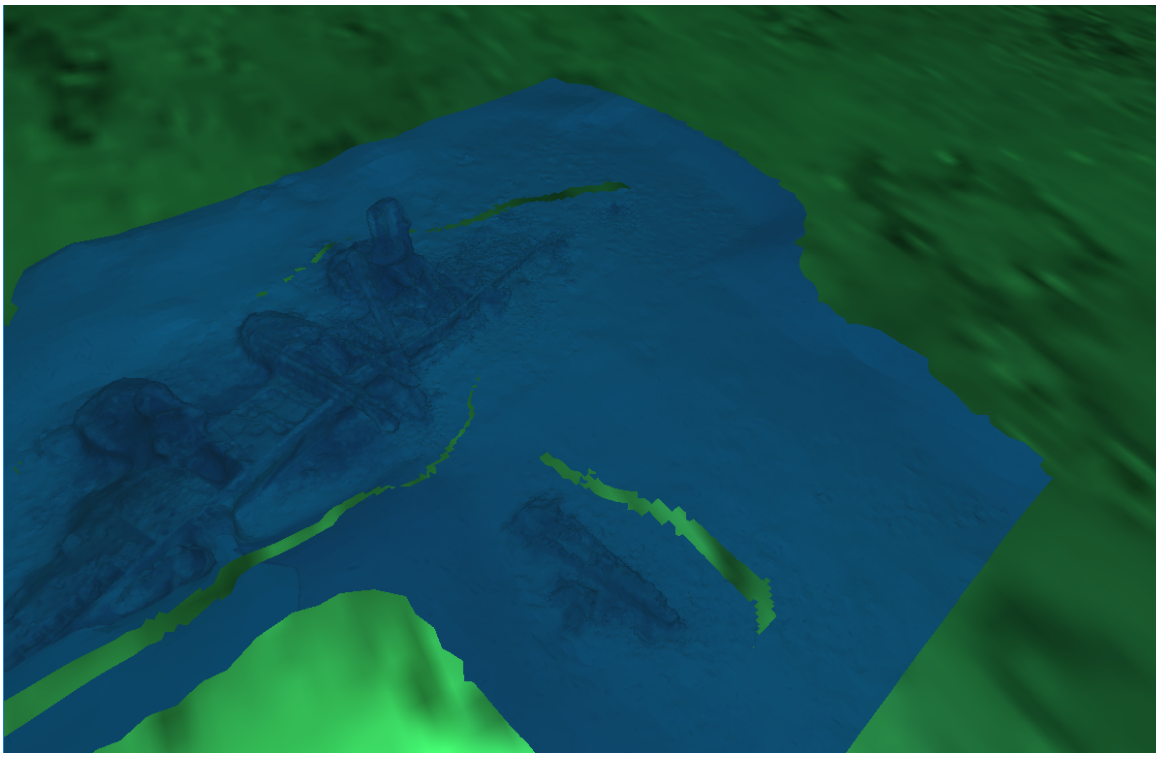

Figure 4.10: The Bristol Beaufighter.

4.12 AUV Pathing

For actual AUV trajectories, the AUV can only locate itself at the surface of the water. Therefore, in order to convert the automatically generated paths to paths that could be used with the AUV for ICEX missions, additional way-points (nodes) would have to be added. Pairs of nodes with similar velocities would be selected and

corresponding surface way-points appropriately spaced to reach the correct depth for each node before and after the node pairs would be added. This process would be repeated until all nodes are paired and associated with surface way-points. The AUV would then travel linearly to each way-point until the path is complete. This is left to future work.

4.13 Testing Details

This thesis was executed for testing on a 2019 MSI GS63 Stealth with an Intel 8th Generation Core i7-8750H processor. The computer also has a dedicated graphics card - NVIDIA Geforce GTX 1060. The code was written in C++14 along with the Open Graphics Library (OpenGL) version 4.6. The OpenGL Mathematics library, GLM, version 0.9.8.5 and GLFW3 version 3.2 were also used.

4.14 Implementation Summary

The path generation system implemented in this thesis was described in detail in this chapter. Paths are generated by a probabilistic roadmap algorithm that uses a rapidly-exploring random tree to quickly explore space. In each iteration of the algorithm, a node to expand off of is selected based on its location in space or its high weight, a new node with a given amount of freedom is generated, the objective function determines the weight of the new node, and then it is added to the roadmap. Once all sides of the shipwreck have been viewed, a path is determined to be complete, it is extracted from the rapidly-exploring random tree, and is written to a file. This file can replay the completed paths.

Chapter 5

RESULTS AND VALIDATION

This thesis presents a probabilistic roadmap algorithm for the generation of paths for an AUV to travel in order to capture multiple views of a region of interest. In addition, we present an application to test these paths in a virtual environment which uses simple geometry but allows for virtual frames to be written out and then used with a photogrammetry application. The algorithm features a rapidly-exploring random tree to quickly cover the volume of exploration space and generate small maps with good coverage. The roadmap is constructed out of nodes, each having its own weight. The weight of a given node is calculated using an objective function which measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. The algorithm was tested in a virtual world where the virtual camera acted as the AUV. All of the images collected from our automatically generated path were used to create 3D models and point clouds using photogrammetry. To measure the effectiveness of our paths versus the pre-packaged lawnmower paths, the 3D models and point clouds created from our algorithm were compared to those generated from running a standard lawnmower pattern. The paths generated by our algorithm captured images that could be used in a 3D reconstruction, that were more detailed, and that showed better coverage of the region of interest than those from the lawnmower pattern.

5.1 Expected Results

It is expected that the probabilistic roadmap algorithm will be able to produce paths based on path length, time limit, and viewing sides of the region of interest. Viewing all sides of the region of interest means the camera has seen each of the sides of the region of interest (front, right, left, back, top) with good coverage. The virtual camera/AUV then follows the generated path through a virtual world. Every third

frame, an image is captured. These images are used in photogrammetry where a reconstruction of the site of interest is created. The reconstructions created by the generated paths can then be compared to those created by following a lawnmower pattern. Point clouds of each reconstruction can also be compared to determine the level of detail of each. It is expected that reconstruction created from data gathered by the generated paths will be more detailed and offer more interesting views and coverage of the site of interest than those from the lawnmower pattern.

5.2 Hypothesis

If a PRM algorithm is used to generate paths, then the reconstructions produced using photogrammetry from the images collected when following the path with a camera/AUV will be more detailed, more interesting, and provide better coverage than those made by following a lawnmower pattern.

5.3 Variables

In this thesis, the independent variables include the type of algorithm being used - either the PRM algorithm paths or the lawnmower pattern. Both algorithms are used to survey the same location. The dependent variables include the resulting reconstructions and point clouds each running each algorithm, collecting image data, and using photogrammetry. The reconstructions and point clouds can be compared for best details, views, and coverage of the site of interest.

5.4 Measures

Both algorithms output an image every third frame that can be used to create a reconstruction using photogrammetry. The quality of the images, surface area of the site of interest covered, and distance from the site of interest, all affect the fidelity of the reconstruction. After the PRM algorithm automatically generates a path, the virtual

camera/AUV follows it taking an image every third frame. Software called Agisoft Photoscan® can then be used for photogrammetry - where a 3D model is created from aligning images. This reconstruction can then be qualitatively compared to one made from running a standard lawnmower pattern. A reconstruction is considered to have better views and coverage if all sides of the site of interest can be seen (front, right, left, back, top) in the 3D model. Point clouds of each reconstruction can also be compared to determine the level of detail of each. Point clouds can be created and compared with an open source software called "CloudCompare". The most important metric for level-of-detail comparison is the mean distance between each point of the point cloud. The less distance between each point, the denser and more detailed the point cloud.

Both the 3D models and point clouds are also compared to the original 3D model of the shipwreck (the site of interest) used in the virtual world and the point cloud of that model. The reconstruction that provides the most information about the shipwreck with the best detail and that is most similar to the original 3D model of the shipwreck can then be decided on.

5.5 Experiment Protocol

The first step of this thesis is to automatically generate paths using the PRM algorithm. The success or failure of this algorithm is left to how random the different configurations generated are and to the nature of the configuration space itself. Therefore, some of the paths automatically generated may not be worth moving forward with. A path has good potential for gathering data for a reconstruction if the randomness moves it all around the site of interest so all sides can be seen, while not moving too far away from it to not lose any detail or coverage. A path with a high average weight is decidedly one with good potential.

After a path has been determined to have good potential, it can be run in a virtual world where the virtual camera acts as the AUV. An image is taken every

third frame as the camera follows the path. The images can then be given to Agisoft Photoscan® to create a 3D model of the site with photogrammetry. First, Agisoft Photoscan® aligns the collected images. If not enough images are aligned, it is not worth continuing on to create a 3D model. The standard lawnmower pattern is also run over the shipwreck model in the virtual world to collect images and build another reconstruction.

The PRM reconstruction can then be compared to the lawnmower reconstruction. The 3D models are analyzed for level of detail and if all sides of the site were reconstructed with good coverage. Multiple reconstructions can be made for comparison. CloudCompare is also used to create point clouds from each 3D model. The mean distance between each point in the point cloud determines the density and, therefore, detail of each reconstruction. If the PRM algorithm continuously leads to denser point clouds and better reconstructions, then it is possible to conclude that it is the algorithm that should be used in both virtual and real-world scenarios as opposed to the standard lawnmower pattern.

5.6 Results and Validation

Of all of the paths generated by the PRM algorithm, twelve had above average weights. The averages were computed by adding the weights of all of the nodes in the generated path and dividing by the number of nodes. These twelve paths were run in the virtual world and the resulting images were given to Agisoft Photoscan®. Of the paths, the one with a high percent of images aligned, PRM 3, was selected for reconstruction. Two paths with only about 50% of images aligned, PRM 6 and PRM 11, were also selected. This is to show that even average PRM paths can produce better results than a reconstruction following the lawnmower pattern. The data for all of the paths is summarized in Table 5.1. Three lawnmower pattern paths were also run. While one of them led to a good reconstruction and point cloud as shown in Figure 5.2 and Figure 5.5, the other two, including an attempt to run the lawnmower pattern created by Yamafune et al. in their work [24], did not produce usable models.

Path	Images Aligned	Percent Aligned	Average Weight
PRM 1	599/608	98.52	0.53825
PRM 2	121/666	18.17	0.523266
PRM 3	468/551	84.94	0.577008
PRM 4	198/614	32.25	0.507342
PRM 5	296/641	46.18	0.566364
PRM 6	790/1322	59.76	0.574813
PRM 7	476/988	48.18	0.563158
PRM 8	429/743	57.74	0.541949
PRM 9	544/646	84.21	0.569361
PRM 10	83/268	30.97	0.561669
PRM 11	295/573	51.48	0.57375
PRM 12	478/592	80.74	0.553023

Table 5.1: Summary of Paths with Good Potential. Includes the images aligned out of how many were inputted to Agisoft Photoscan ®, the percent of those actually aligned, and the average weight of each path.

By comparing both reconstructions from running path PRM 6 and from the lawnmower pattern to the original 3D model of the shipwreck used in the virtual world, it is easy to see which of the reconstructions was more successful - PRM 6. Figure 5.1 shows the side profiles of the reconstructions alongside the original 3D model. The lawnmower pattern clearly did not get good views or coverage of the sides of the shipwreck as they are completely missing from the reconstruction. PRM 6, however, followed a path with freedoms in pitch and yaw, so all of the sides of the shipwreck were seen and reconstructed very similarly to the original 3D model. All of the paths, PRM and lawnmower, produced good reconstructions of the top of the shipwreck as shown in Figure 5.2 and Figure 5.3.

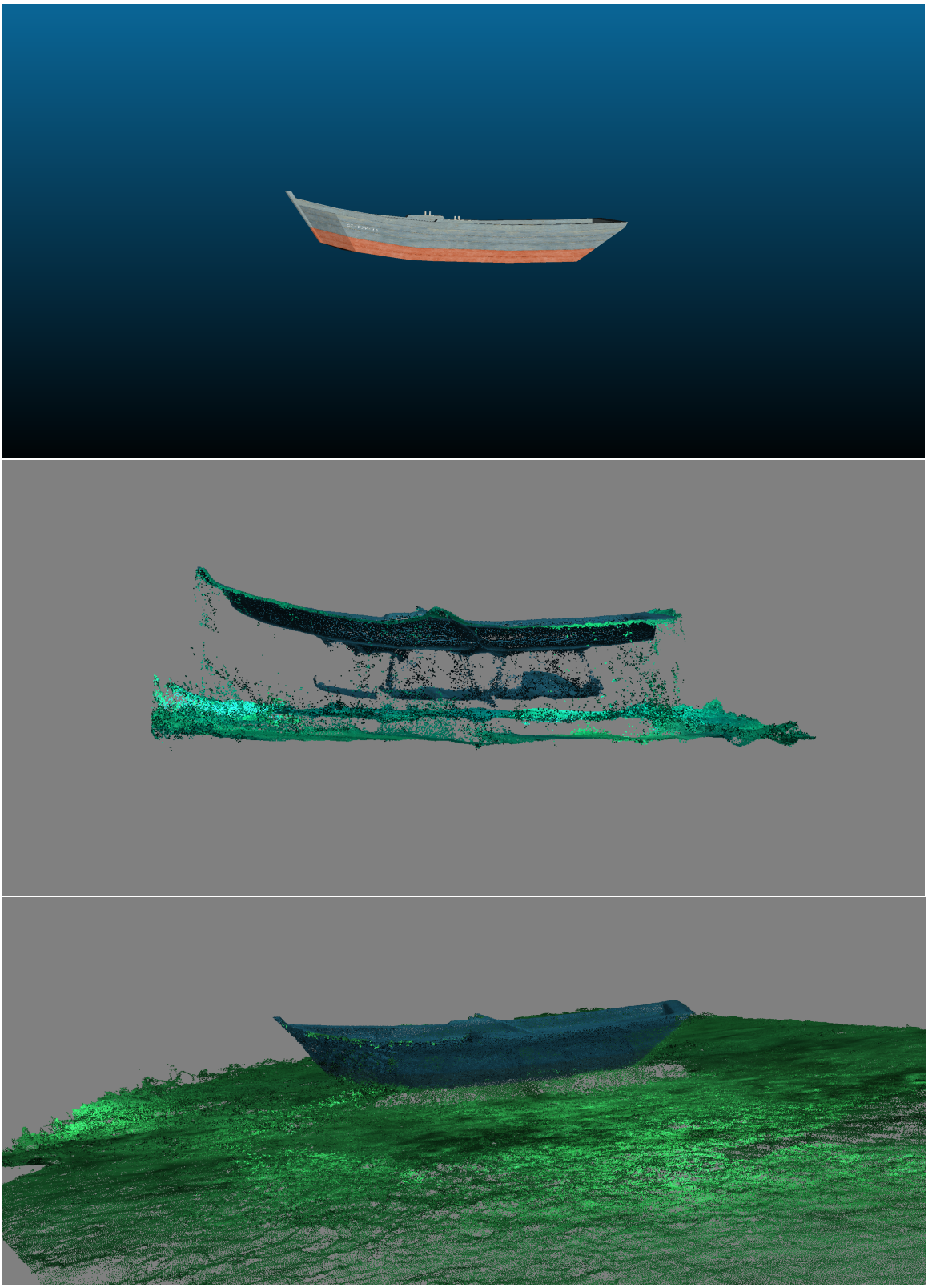

Figure 5.1: 3D Models from the Original Shipwreck, the lawnmower pattern, and PRM Path 6

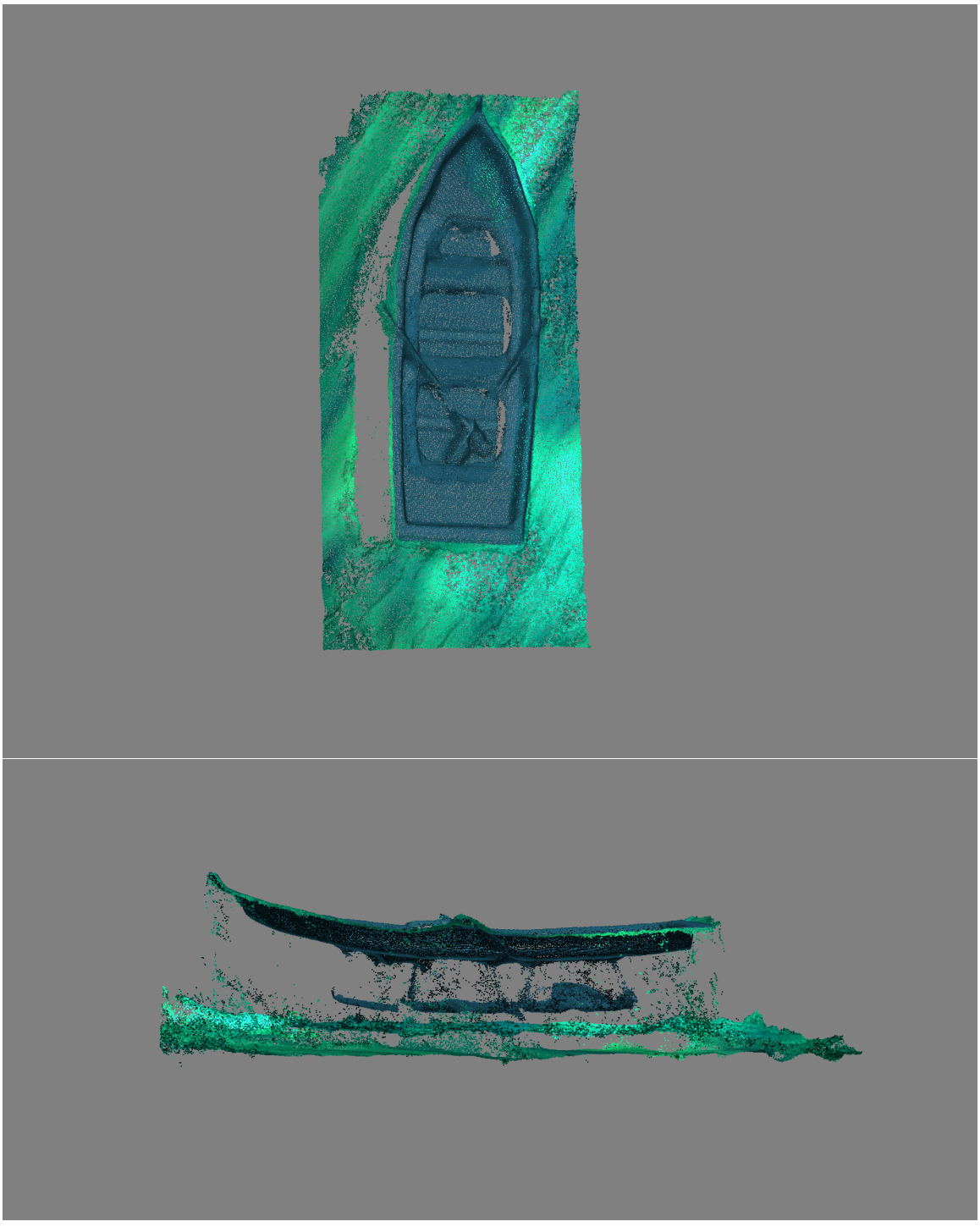

Figure 5.2: 3D Model created by Lawnmower Pattern

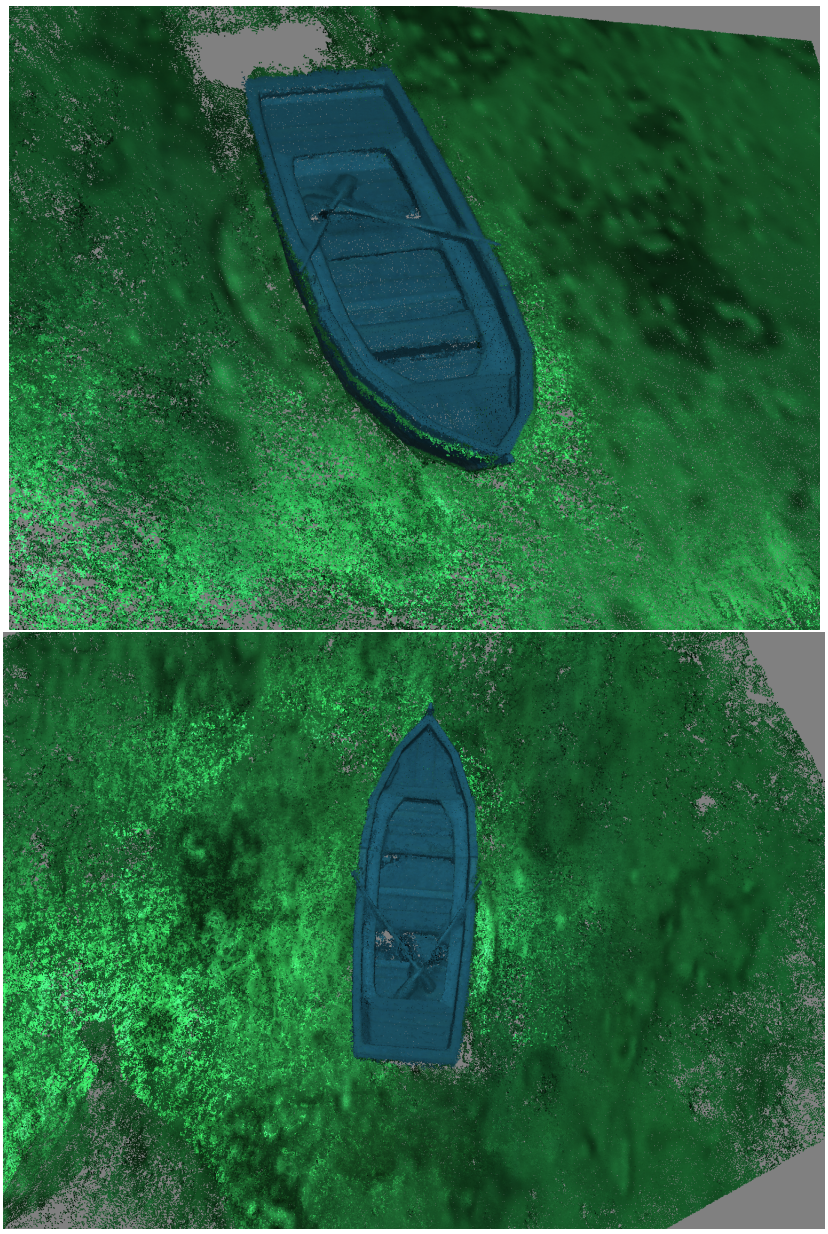

Figure 5.3: 3D Models created by PRM 11 and PRM 6

All of the reconstructed 3D models were then given to CloudCompare to create point clouds for further comparison. The denser the point cloud, the more detailed the reconstruction. The density can be measured by the mean distance between points where the smaller distance between points means greater density. All of the point clouds are compared to each other and to the point cloud of the original shipwreck used in the virtual world.

In the images, Figure 5.4 shows the point cloud of the original 3D model in gray. In the point clouds created from the lawnmower pattern and the PRM paths, the point cloud of the original model is left in for comparison purposes. The original model's point cloud is colored gray while the lawnmower and PRM point clouds are rendered over it in blue. Figure 5.5, Figure 5.6, and Figure 5.7 show their own point clouds in blue over the gray point cloud created from the original model. As shown in the images and reinforced in the data displayed in Table 5.2, the point clouds created by the PRM paths are significantly more dense than the point clouds created by the lawnmower pattern. When comparing mean distance of points in point clouds, it is better if the value is smaller because it means the point cloud is denser. The mean distance of the PRM paths' point clouds range from being 1.5 to 2.3 times better than that of the lawnmower pattern.

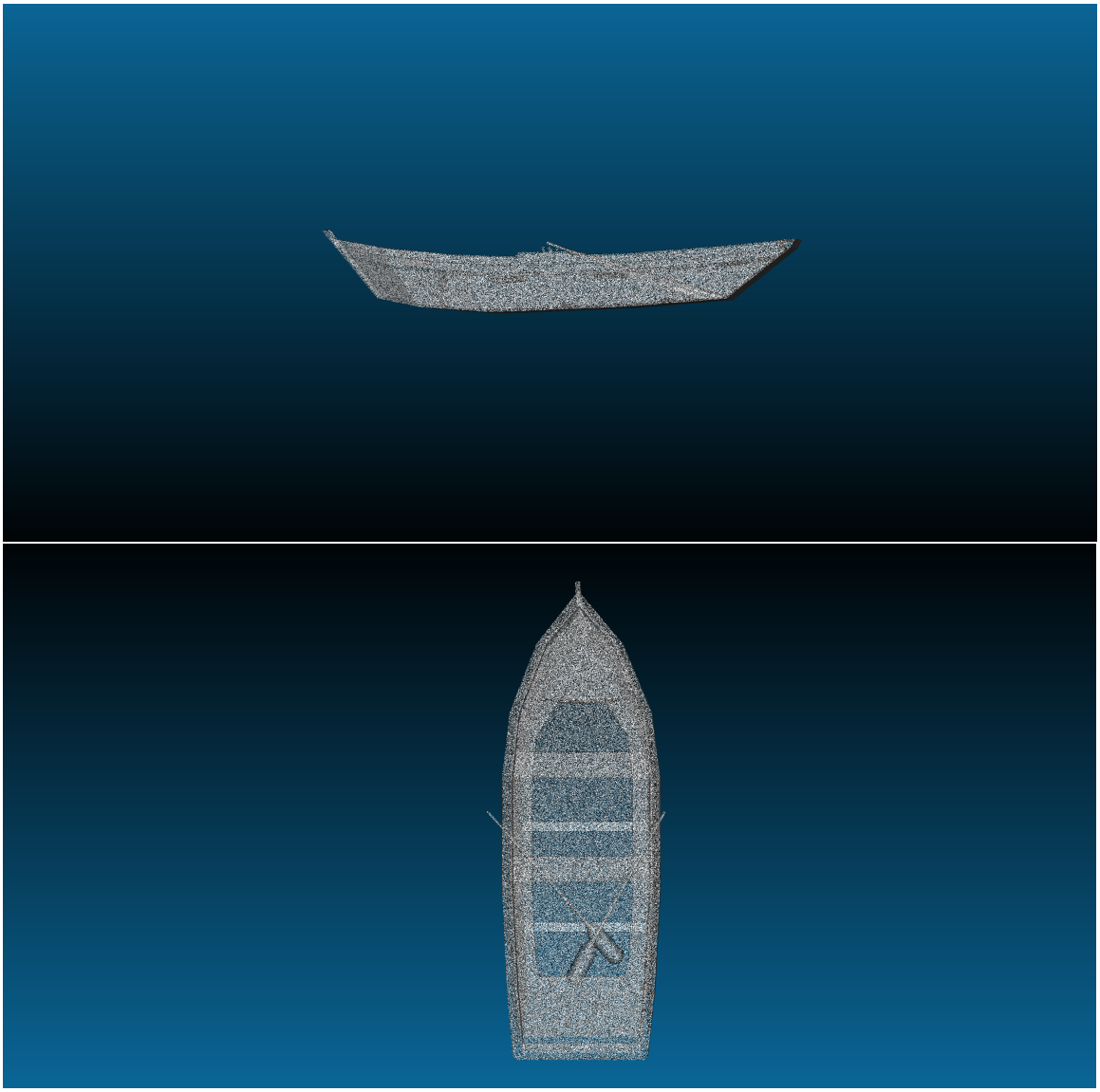

Figure 5.4: Point Cloud of Original Shipwreck

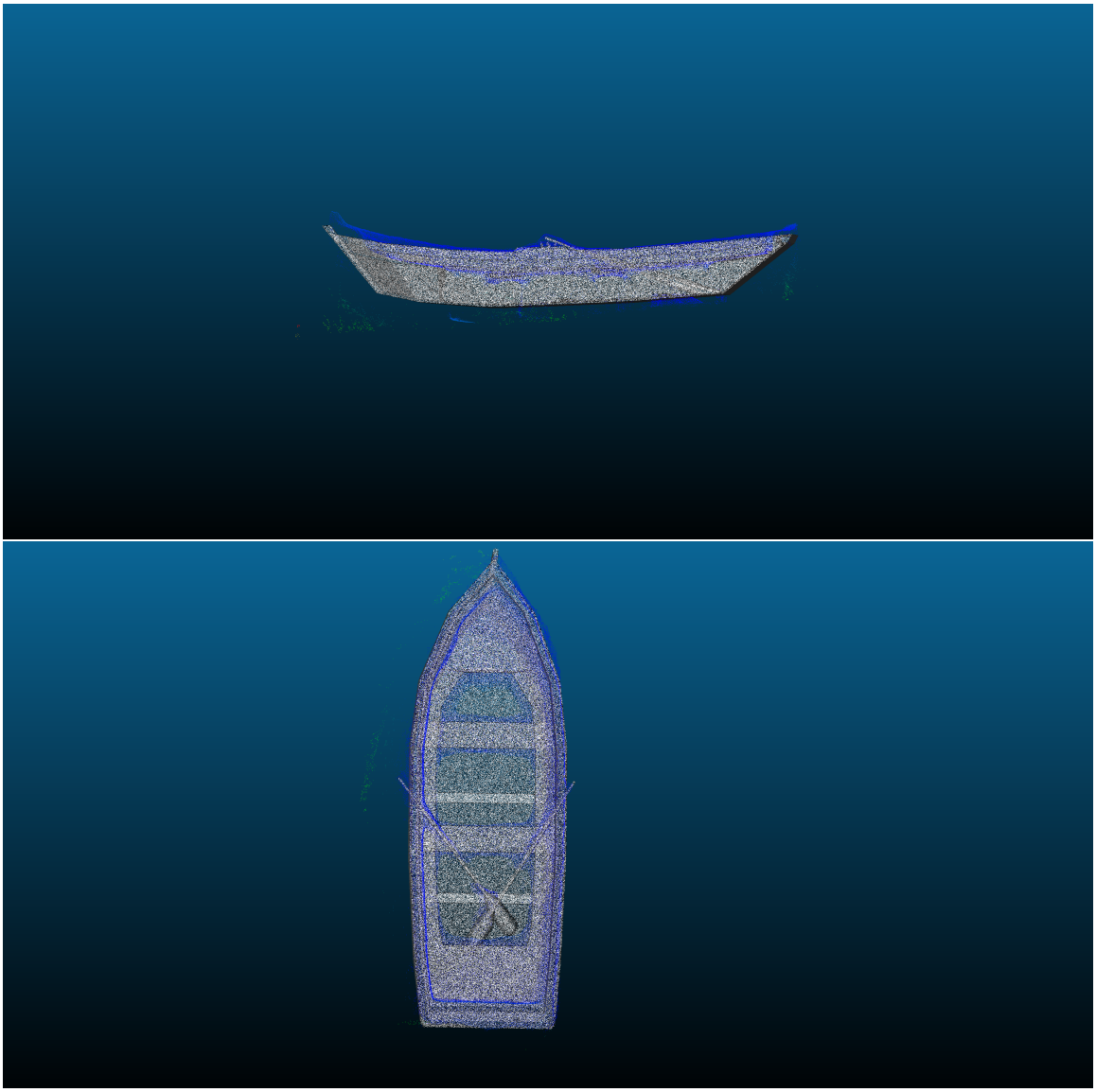

Figure 5.5: Point Cloud from Lawnmower Pattern

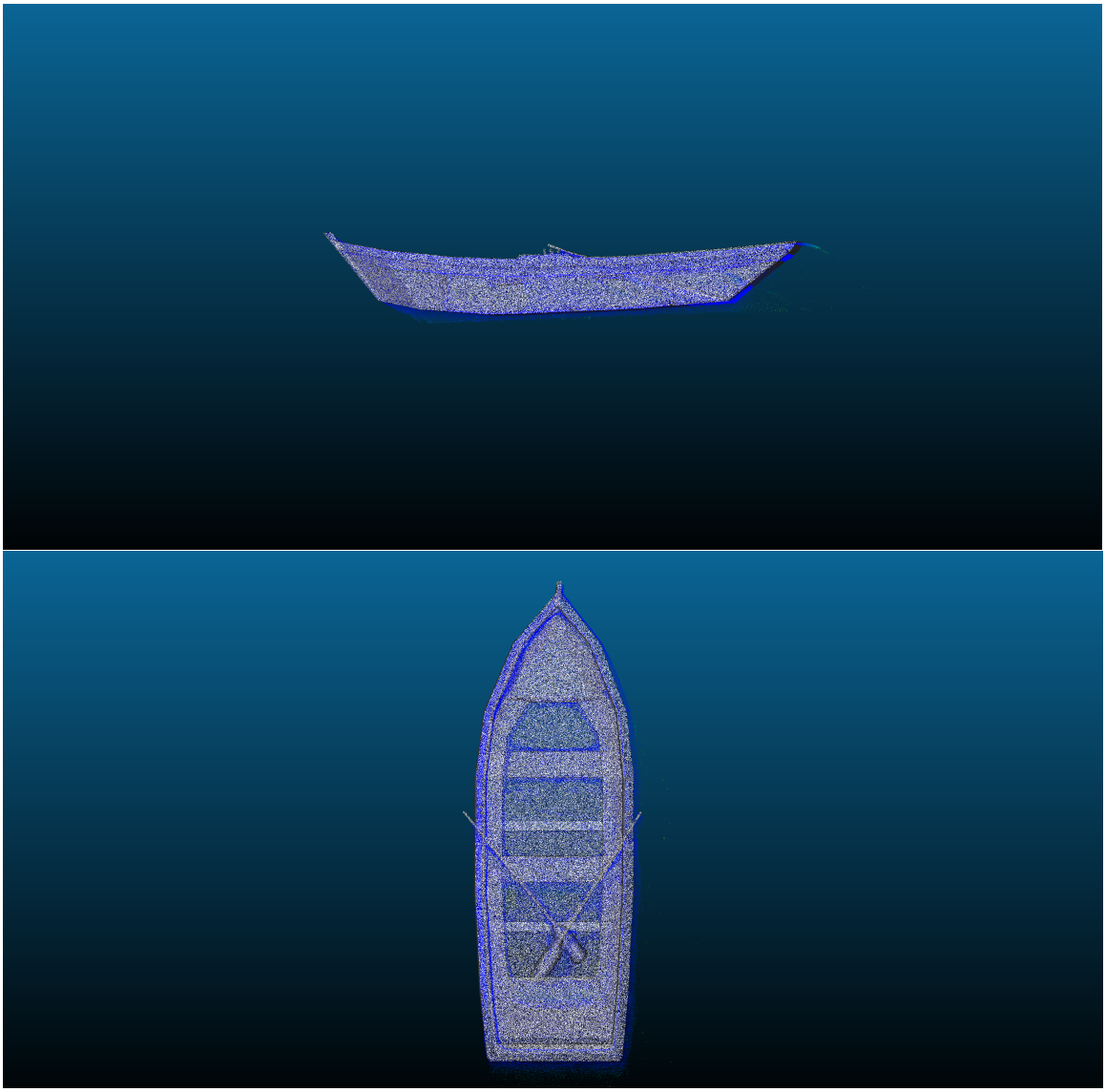

Figure 5.6: Point Clouds from paths PRM 3

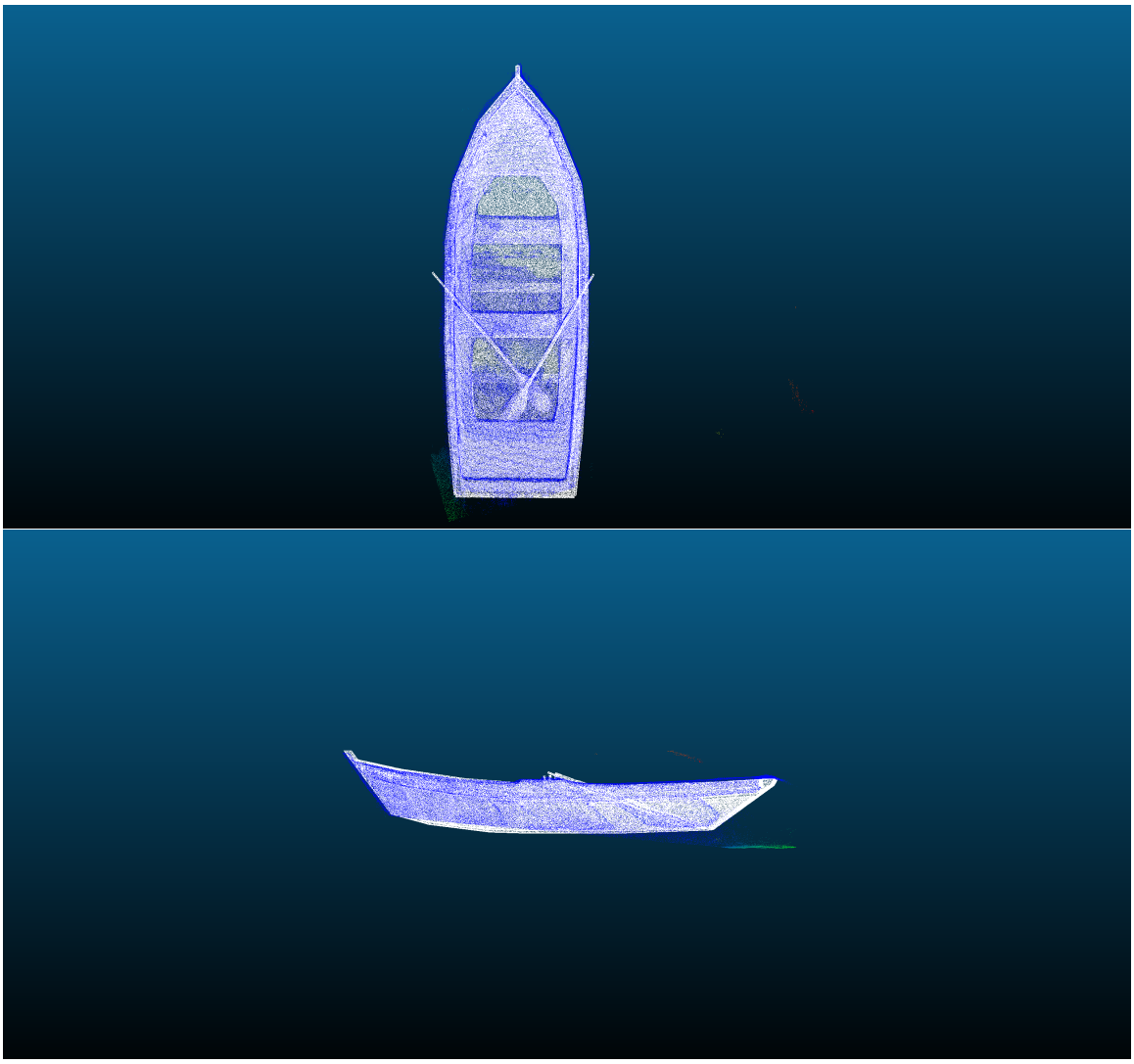

Figure 5.7: Point Clouds from paths PRM 11

Path	Mean Distance	Standard Deviation
PRM 3	0.14937	0.18863
PRM 6	0.120192	0.387108
PRM 11	0.232785	0.590656
Lawnmower	0.336414	0.431753

Table 5.2: Point Cloud data for PRM and lawnmower paths

It is also important to note that the distance covered by each path does not correlate to the quality of the reconstruction. Table 5.3 shows the distance each path covered in the virtual world. The lawnmower pattern covers a distance of almost twice as far as path PRM 11, but PRM 11 is a better reconstruction overall.

Path	Distance Covered
PRM 3	1678.94
PRM 6	2270.19
PRM 11	664.24
Lawnmower	1128.49

Table 5.3: Distance covered by the selected paths in virtual world units.

5.7 Result and Validation Conclusions

The 3D models created by the generated PRM paths are more detailed and show better views with better coverage of all sides of the shipwreck than the 3D models created by the lawnmower pattern. While all paths produced good reconstructions of the top of the shipwreck, the lawnmower pattern almost completely missed the sides of the shipwreck in its reconstruction. The PRM paths also created significantly denser point clouds than the lawnmower paths, with a mean distance between points that ranges from 1.5 to 2.3 times better than that of the lawnmower pattern.

5.8 Algorithm Performance

5.8.1 The Objective Function

The objective function, as detailed in Section 4.3, had a significant effect on performance. Initially, the algorithm was allowed to expand by choosing a node completely randomly from the roadmap. On multiple occasions, the algorithm was left to run

overnight without generating a complete path because it either never saw all sides of the region of interest or the computer ran out of memory. In Section 4.6, on "Using the Spatial Data Structure", and Section 4.7, on "Node Selection", how the algorithm expands to less explored areas with the spatial data structure is discussed. This was the primary form of node selection for awhile, but it was still very slow. The algorithm often took hours, but it did generate complete paths. When the high weight threshold was added to the node selection process, path generation went from taking hours to minutes.

The average, minimum, and maximum weights for each path are summarized in Table 5.4. The weights in this table accurately reflect the node selection process of the algorithm. The nodes selected from less explored areas within the spatial data structure are more likely to have lower weights. The objective function creates regions of space with many high weight nodes. Often the nodes not in these denser high weight regions have lower weights. As the spatial data structure de-clusters the roadmap by expanding to unexplored areas, it selects low weight nodes. The other half of the nodes selected for roadmap expansion are those above a high weight threshold. Therefore, we have an equal distribution of low weight and high weight nodes leading to paths with average weight overall.

Path	Average Weight	Minimum Weight	Maximum Weight
1	0.53825	0.12375	0.985
2	0.523266	0.005	1
3	0.577008	0.0075	1
4	0.507342	0	1.19375
5	0.566364	0.125	1.25
6	0.574813	0	1.225
7	0.563158	0.12375	1.25
8	0.541949	0.1075	1
9	0.569361	0	1
10	0.561669	0	1
11	0.57375	0.1275	1.25
12	0.553023	0	1.25

Table 5.4: Average, minimum, and maximum weight from each generated path.

5.8.2 Cubic Hermite Interpolation

The final algorithm produced paths with anywhere from 20 to 200 nodes as shown in Table 5.5. More nodes means a higher memory cost and higher times to generate. The virtual camera moves through the world by visiting each node, but this creates very jumpy motion. Significantly more nodes would have to be added between the original node in order for the virtual camera to move smoothly along the final path. Instead, to keep the number of nodes in a path from increasing when generating the final path, a cubic Hermite spline was created to interpolate between the nodes. The camera can then move along the spline instead of jumping from node to node.

5.8.3 The Hash Map

The largest improvement on performance, by far, was the addition of a hash map. Previously, the spatial data structure was looping over all of 3D space to find which voxel had the least amount of nodes. This was an $O(n^3)$ operation. With the hash map, the algorithm could look up the voxel with the least amount of nodes in $O(1)$ time. After the implementation of the hash map, paths could be generated in seconds. As shown below in Table 5.5, all of the generated paths were between 10 and 27 seconds.

5.9 Time Performance

As mentioned above, before the addition of features such as the objective function and the hash map, the algorithm took hours to find a path that completed our goals. Now, it only takes seconds. Table 5.5 shows a summary of the generated paths including the length of each path, the number of nodes in each respective roadmap, and the time it took to generate them. The larger the roadmap, the more time the algorithm took to create a path. However, even the largest roadmap with 538 nodes took only 27 seconds to generate a path - Path 12.

Path	Path Length	Roadmap Size	Time (s)
1	20	36	10
2	62	99	15
3	160	261	23
4	87	193	17
5	33	48	11
6	207	407	26
7	76	178	16
8	118	189	20
9	92	137	20
10	155	234	22
11	61	100	14
12	189	538	27

Table 5.5: Summary of generated paths. Includes the length of each path, the number of nodes in each roadmap, and the time to generate each path in seconds.

5.10 Analysis of Parameters

There were numerous factors that contributed to choosing the values for the parameters used in the algorithm. The most important parameters include delta phi, delta theta, root node configuration, the high weight threshold, bounding box volume encasing the region of interest, the volume of exploration area, and what contributes to node weight with the objective function.

The deltas used in node generation mentioned in Section 4.8, phi and theta, could vary from $-\pi$ to π allowing pitch and yaw to be completely free. When the roadmap was having trouble expanding throughout the entire space and clustering on one or two sides of the region of interest, it was first thought that maybe the deltas needed

to be limited to encourage more circular expansion. Instead, both the volume the algorithm was able to explore and the bounding box around the region of interest were increased. Increasing the size of the bounding box around the region of interest seems counterproductive, but it was actually very beneficial. By forcing the virtual AUV to off from the region of interest, it had a much broader view of the region of interest and was able to see more sides more quickly. The volume the algorithm could explore was also increased thus making it easier to get around the sides of the region of interest by creating more space to expand.

The root node was positioned at the top left corner of this volume and oriented to face the center of the region of interest creating an initial high weight node that realistically reflected where an AUV would be positioned at the beginning of a survey.

The objective function measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. We started with casting only six rays out from the camera to ensure weighting this way was effective. Once the paths started to move towards higher weight nodes and we were sure this was an effective method to determine weight, we began increasing the amount of rays. The chosen amount of rays to get an accurate percentage of how much of the region of interest is visible without slowing down performance was determined to be 200 rays. In addition, to encourage expansion towards new sides of the region of interest, the weight of a node is increased if this node allows the AUV to see a new side of the region of interest and if the new node has a higher weight than its parent node. Without careful consideration of all of these different parameters, the algorithm would not be able to generate paths.

5.11 Limitations

The parameters above are set specifically to work with this virtual scene. However, as shown in Section 4.11 with the Bristol Beaufighter model, the algorithm can work with any other virtual scene with either no or only minor adjustments. How the

algorithm determines if it sees every side of the region of interest is specific to one bounding box. Therefore, the algorithm can work with multiple 3D models, but only if they are all put into one larger bounding box. It would be possible to provide each 3D model with its own bounding box, creating more interesting paths, in future iterations of this work.

Chapter 6

CONCLUSIONS

6.1 Conclusion

While most of the ocean is still unexplored, this thesis offers a way to speed up surveying sites of potential historical significance. Paths are generated by a probabilistic roadmap algorithm that uses a rapidly-exploring random tree to quickly cover the volume of exploration space and generate small maps with good coverage. The roadmap is constructed out of nodes, each having its own weight. The weight of a given node is calculated using an objective function which measures an approximate view coverage by casting rays from the virtual view and intersecting them with the region of interest. In addition, the weight of a node is increased if this node allows the AUV to see a new side of the region of interest. In each iteration of the algorithm, a node to expand off of is selected based on its location in space or its high weight, a new node with a given amount of freedom is generated, and then added to the roadmap. The algorithm has degrees of freedom in position, pitch, and yaw as well as the objective function to encourage the path to see all sides of the region of interest. Once all sides of the region of interest have been viewed, a path is determined to be complete.

The algorithm was tested in a virtual world where the virtual camera acted as the AUV. All of the images collected from our automatically generated path were used to create 3D models and point clouds using photogrammetry with Agisoft Photoscan®. The reconstructions could then be used in CloudCompare to create point clouds. To measure the effectiveness of our paths versus the pre-packaged lawnmower paths, the 3D models and point clouds created from our algorithm were compared to those generated from running a standard lawnmower pattern. In both the reconstruction having more detailed sides with good coverage and in point cloud density, the PRM algorithm outperformed the lawnmower pattern when compared to the original shipwreck model.

6.2 Future Work

This thesis introduces a path planning algorithm for AUVs using a probabilistic roadmap. When compared to the standard lawnmower pattern, the generated PRM paths performed better. However, there is always room for improvement. The following areas allow for potential future work:

- Expand the PRM algorithm to account for multiple bounding boxes.
- As mentioned in Section 4.12, more accurately reflect AUV pathing by adding surface way-points (nodes). Nodes would be placed at the water surface and the path would be linear lines between each node.
- Instead of casting rays from the camera to the bounding box, render the bounding box in a given color. Then, count the colors in the frame buffer object from the virtual camera configured at a given node.
- Analysis of the number of rays being cast. There is potentially a better number of rays to cast than 200 rays.
- For node generation, analyze lessening the delta values to a smaller range.
- Bit encode sections of the sides of the bounding box to count how many rays have hit which sections. This could be used to encourage not oversampling/ over-viewing a certain region of the site.
- Instead of having the algorithm run for a finite amount of time after the last side of the bounding box has been seen, count how many rays have intersected it and terminate the path after a certain amount. This could ensure good coverage of the final side for any size region of interest.
- Expand the PRM algorithm to work with a sixth degree of freedom to potentially create more interesting paths.

- Explore other methods of interpolating between nodes besides cubic-Hermite splines.
- Deploying an AUV off the coast of Malta to follow the PRM generated paths from this thesis.

BIBLIOGRAPHY

[1] V. Alcazar, M. M. Veloso, and D. Borrajo. Adapting a rapidly-exploring random tree for automated planning. January 2011.

[2] A. Burgess and T. Gambin. The underwater aviation heritage of the second siege of malta. *Malta Archaeological Review*, 10:53–60, 2010-2011.

[3] K. Carroll, S. McClaran, E. Nelson, D. Barnett, D. Friesen, and G. William. Auv path planning: an a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones. In *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, pages 79–84. IEEE, June 1992.

[4] C. Clark. E160 - lecture 13 autonomous robot navigation. `http://www.hmc.edu/lair/E160/E160-Lecture14-MotionPlanning.pdf`, June 2016.

[5] C. Clark, A. Lewis, S. Freed, J. Rutledge, and Z. Wood. Auv and graphics research motivated by marine archaeology: From development to discovery. In *International Conference on Aviation Archaeology and Heritage (ICAAH)*, Valetta, Malta, November 2017. Heritage Malta.

[6] K. Davis. Probabilitic roadmaps for virtual camera pathing with cinematographic principles. Master's thesis, California Polytechnic State University - San Luis Obispo, 1 Grand Ave. San Luis Obispo, CA 93405, 2017.

[7] P. Debertolis, N. Earl, and M. Zivic. Archaeoacoustic analysis of tarxien temples in malta. *Journal of Anthropology and Archaeology*, 4(1):7–27, June 2016.

[8] Demofox. Cubic hermite interpolation. `https:`

//blog.demofox.org/2015/08/08/cubic-hermite-interpolation/, 2015.

[9] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge, 2010.

[10] E. Dunn, J. v. d. Berg, and J. Frahm. Developing visual sensing strategies through next best view planning. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4001–4008, October 2009.

[11] K. Frischkorn. Why the first complete map of the ocean floor is stirring controversial waters. https://www.smithsonianmag.com/science-nature/first-complete-map-ocean-floor-stirring-controversial-waters-180963993/, July 2017. Last accessed 21 January 2019.

[12] T. Gambin. The maritme heritage of malta: Past, present and future. *Maritme Heritage: Advances in Architecture*, 65, 2003.

[13] B. Garau, A. Alvarez, and G. Oliver. Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an a* approach. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 194–198. IEEE, January 2006.

[14] M. Rantanen. *Improving Probabilistic Roadmap Methods for Fast Motion*. PhD thesis, School of Information Sciences, University of Tampere, 2014.

[15] D. Rao and S. B. Williams. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA)*, December 2009.

[16] J. Rutledge, W. Yuan, J. Wu, A. Lewis, S. Freed, Z. Wood, and C. Clark. Intelligent shipwreck search using autonomous underwater vehicles. *IEEE*

International Conference on Robotics and Automation (ICRA), pages 1–8, September 2018.

[17] H. Samet. Spatial data structures. *Modern Database Systems, The Object Model, Interoperability and Beyond*, pages 361–385, 1995.

[18] Scratchapixel. Ray-box intersection. `http://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection`, 2004.

[19] C. S. Tan, R. Sutton, and J. Chudley. An incremental stochastic motion planning technique for autonomous underwater vehicles. volume 37, pages 483–488, May 2017.

[20] V. Viswanathan, Z. Lobo, J. Lupanow, S. von Frock, T. Gambin, Z. Wood, and C. Clark. Auv motion-planning for photogrammetric reconstruction of marine archaeological sites. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5096–5103, May 2017.

[21] S. von Fock, K. Davis, S. Bilich, V. Viswanathan, Z. Lobo, J. Lupanow, T. Gambin, Z. Wood, and C. Clark. Pipeline for reconstruction and visualization of underwater archaeology sites using photogrammetry. In *International Conference on Computers and Their Applications (ISCA)*, March 2017.

[22] A. Williams, S. Barrus, R. Keith, and M. P. Shirley. An efficient and robust ray-box intersection algorithm. *Journal of Graphics Tools*, 10:54, 2004.

[23] Xojo. Tutorial 12: Quaternions. `http://www.xojo3d.com/tut012.php`.

[24] K. Yamafune, R. Torres, and F. Castro. Multi-image photogrammetry to record and reconstruct underwater shipwreck sites. *Journal of Archaeological Method and Theory*, 2016.