Approaches for Autonomous Multi-Vehicle Underwater Inspections

Marina A. Ruediger

A dissertation
submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:
Ashis G. Banerjee, Chair
Santosh Devasia
Aaron Marburg

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Approaches for Autonomous Multi-Vehicle Underwater Inspections

Marina A. Ruediger

Chair of the Supervisory Committee:
Ashis G. Banerjee
Department of Industrial & Systems Engineering and Department of Mechanical Engineering

Underwater robotic inspections are an ongoing challenge for large naval platforms. Although many approaches to specific aspects of robotic inspection tasks have been previously explored, the limitations of specific approaches leave large gaps in underwater inspection capabilities. By developing and implementing a cooperative multi-robot system for underwater inspections, we aim to bridge these gaps in existing methods. Cooperative underwater robotic systems face the unique challenges of highly dynamic environmental conditions, extremely limited communications and sensor ranges, expensive and easily damaged hardware, and limited simulation options. This dissertation focuses on bringing together variations on existing methods for task allocation, task discovery, SLAM, and motion planning, modifying them to fit the unique underwater challenges, and integrating them into a system that takes advantages of the similarities and strengths of different methods to build a cooperative robotic system for underwater inspection and maintenance.

To begin the testing and integration of various task allocation, SLAM, and motion planning, these methods must first be tested on simulations and datasets. By leveraging aspects of several tools, we are able to develop a basic simulation that allows testing of coordination methods while allowing a controlled environment with the basic tools needed to evaluate the performance of a decentralized task allocation method. By modifying the d-CBBA method

to account for task additions and communication limitations, we enable our simulated robots to distribute dynamic sets of tasks.

Without prior knowledge of the geometry of the objects being inspected, a method to generate appropriate inspection locations during exploration is needed to determine which tasks should be passed on to be allocated between the robots. With the extreme limitations on communications, there is no way to communicate environment maps between robots in real-time. Individual robots can generate their own maps through onboard SLAM processes, but the only environmental information they receive from other robots in the system is the location of tasks passed between robots. To approach the task discovery problem, we develop Map-TIDAL, which uses the individual robot maps to generate candidate tasks, then compares them to the set of existing tasks made available to the robots through the task allocation to determine the best tasks to add. By integrating task allocation, SLAM, and task discovery, the robots are able to coordinate their efforts with significantly reduced communication bandwidth.

By adding motion planning methods into the Map-TIDAL framework, the multi-robot system will be able to perform inspections without risking collisions that could damage the robots or the vessels being inspected. Analysis is performed on several available planners, to determine which planners would be most applicable to the underwater inspection domain.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables v

Glossary vi

Chapter 1: Introduction 1
1.1 Motivation 1
1.2 Underwater Robotics 3
1.3 Dissertation Outline 7

Chapter 2: Related Literature 9
2.1 Simulations and Datasets 9
2.2 Cooperative Robotics 12
2.3 SLAM 13
2.4 Coverage Based Methods 13
2.5 Motion Planning 14

Chapter 3: Underwater Cooperation Simulation 17
3.1 Simulation Components 17
3.2 Multi-Robot Task Allocation 21
3.3 Conclusions 30

Chapter 4: Map-TIDAL 31
4.1 Method 31
4.2 Experimental Results 39
4.3 Discussion 52
4.4 Conclusions 53

Chapter 5: Analysis of Motion Planning in Underwater Contexts 54
5.1 Summary of Methods 54
5.2 Adaptations for the Underwater Environment 59
5.3 Analysis and Results 62
5.4 Conclusions 69

Chapter 6: Conclusions 71
6.1 Contributions 71
6.2 Impacts 73
6.3 Future Work 74

Bibliography 76

Appendix A: Proof of DMG Property of Task Allocation Bid Function 82

LIST OF FIGURES

Figure Number		Page
1.1	Size comparison of several naval vessels to common references.	2
1.2	System diagram of all the cyber-physical processes in a simulated underwater vehicle.	4
3.1	Simulation of a generic underwater vessel, featuring weld lines, sea chests, and an internal tank access point.	18
3.2	Partial map of an underwater environment generated by a BlueROV.	20
3.3	Process diagram for the task manager, including colors for which areas of memory are changed during different operations.	22
3.4	Allocation and execution of nine inspection tasks on the exterior hull of an underwater vessel by a team of three robots.	28
4.1	Flowchart of the task discovery and task allocation processes.	32
4.2	**a.** Inspection point (orange) generation from mesh (grey) using the centroid (blue) and normal (green). **b.** Comparison of inspection points (orange) to existing tasks (blue), point A has no tasks inside the radius of exclusion, point B has a task in the radius of inclusion, but the angle is outside of the angle of exclusion, point C has a task that is within both the radius and angle of exclusion, and will not be removed. **c.** Calculation of keypoint scores based on the camera field of view. The score is the total of the keypoints (green) that fall in the field of view for the inspection point (orange) normalized by the number of keypoints. **d.** Final pruning on remaining inspection points, done in alphabetical order tasks E and H are removed, the remainder become tasks.	35
4.3	**A.** Left camera image with tracked keypoints overlaid in green and untracked keypoints in red. **B.** Kimera mesh with A. as texture. **C.** Initial tasks shown as arrows in red, mesh in blue, keypoints as rainbow dots. **D.** The 326 CIPs generated in teal. **E.** The 165 CIPs remaining after the first pruning in teal. **F.** The 49 tasks kept from their keypoint scores in green, with the remaining 116 CIPs in red. **G.** The 98 tasks kept after the final pruning.	40

4.4 Total number of tasks over seven frames with: Top: varied r_{ex} values, $\theta_{ex} = 0.3$ rad, and $t = 75\%$; Middle: varied θ_{ex} values, $r_{ex} = 1.5$ m, and $t = 75\%$; Bottom: varied t values, $\theta_{ex} = 0.3$ rad, and $r_{ex} = 1.5$ m. 41

4.5 **A.** Top view of the test tank with dimensions. Solid lines show geometry used to generate inspection points for Voronoi and sweeping pattern methods, dashed lines show the edges of the voxelized region used to evaluate method effectiveness. **B.** Test tank with two modified BlueROV2s with the main robot at the center. 42

4.6 Simulated coverage analysis for **A** Voronoi, **B** Map-TIDAL, and **C** sweeping patterns with task locations shown in blue and voxels colored brighter red with each time they are viewed. 45

4.7 Histogram showing normalized views per voxel for different task generation methods compared with the discretized ideal gamma distribution. 46

4.8 Task discovery with steps shown over the first two mesh frames in pool testing. CIPs are colored blue initially. In the case of the first mesh, the final tasks are shown in black, which remain when they are used in the second mesh as the existing tasks. For the second mesh, during the score separation step, ones that are selected are green, while the ones subject to final pruning are red, and the final selection is shown in light green. These light green tasks are then combined with the existing list of tasks for the next mesh frame. 48

4.9 Simulated coverage analysis of a pool section for three methods with task locations shown in blue and voxels colored brighter red with each time they are viewed. 49

4.10 Histogram showing normalized views per voxel for different task generation methods compared with the discretized ideal gamma distribution on a pool section. 50

5.1 Graph comparing the average path lengths of the various planners for each set of trials. 64

5.2 Sample of paths generated by each planner. 66

5.3 Sample of paths generated by each planner. 67

A.1 Schematic illustration of task bidding by a robot. 82

LIST OF TABLES

Table Number Page

4.1 Average Views per Voxel, Number of Tasks, and Percent Uninspected 43

5.1 Average Distance Between Final Location and Goal (m) 63

GLOSSARY

AUV: Autonomous Underwater Vehicles, typically used to refer to torpedo shaped UUVs.

CAD: Computer Aided Design, the process of using a computer system to create and analyze digital models. Also used as an adjective to describe the models created in such a process or the computer software and tools that are used in the CAD process.

DOF: Degree of Freedom, the number of different directions or states that can be separately controlled in a dynamic system.

DVL: Doppler Velocity Log, an underwater velocity sensor based on measuring the frequency shift of reflected sound waves (the Doppler effect).

ENU: East North Up, the standard coordinate system for air and land based robotics, with the x-axis pointing east (or to the right in body fixed systems), the y-axis pointing north (or forward in body facing systems), and the z-axis pointing up.

GPS: Global Positioning System, a position sensor based on triangulating the position of an object using a signal received at multiple locations.

IMU: Inertial Measurement Unit, an acceleration sensor.

NED: North East Down, the standard coordinate system used in underwater robotics, with the x-axis pointing north (or forward in body-fixed systems), the y-axis pointing east (or to the right in body-fixed), and the z axis pointing down.

RGB: Red, Green, Blue, one method of conveying digital color images using integer values for the red, green, and blue intensity information for each pixel of the image.

RGBD: An extension of RGB images that also includes depth (distance) information for each pixel to create a 3D image.

ROV: Remotely Operated Vehicle, a vehicle typically operated by a remote control process. Can be used to refer to autonomous vehicles that can be remotely operated even if they are not doing that while acting autonomously.

SLAM: Simultaneous Localization and Mapping, a process in which sensor data is used to determine both where a robot is located and to create a map of where the robot has been.

UUV: Unmanned Underwater Vehicle, an underwater vehicle operated without any humans on board, either remotely or autonomously.

ACKNOWLEDGMENTS

I sincerely thank all the people who have helped me through my academic journey, whether you are listed here specifically or not. My first thanks to Prof. Ashis Banerjee for your invaluable guidance through the snarls of research, funding, and technical writing. I have had the great opportunity to pursue the sort of research I have always wanted to explore because you chose me for this project and gave me the chance to succeed.

Thank you to Prof. Santosh Devasia, Dr. Aaron Marburg, and Dr. Abhishek Gupta for agreeing to serve on my supervisory committee, I look forward to your feedback and advice, and have no doubt they will improve my work tremendously.

Thank you to the Mechanical Engineering Department at the University of Washington for the allowing me to expand me education here, providing me with the opportunity to teach, and facilitating the connections I have relied on throughout my work here. Thank you to all my lab mates, especially Ben, Tommy, Ekta, Apoorva, and Diya and many more- your help and friendship kept me and my robots going through the trials of learning new software, long days of research, and cold days at the test tank.

A special thanks to my family who has supported me through this journey, from start to end. To my mom, Lucy, you have been unendingly patient with me and supported me through all my struggles, I know I can always count on you to be there and listen. To my dad, Bill, your practical advice and problem solving have helped me build all the skills I need to succeed. To my brother, John, I'm proud of you and the person you are becoming, I hope you can be proud of me too; a large part of my drive to succeed comes from wanting to show you the best I can be, and inspire you to do the same. To my grandparents, Bill and Julia, I thank you for all you have done to pave the way for me to succeed- from sharing

your love of candy to being there for the holidays and vacations and fishing trips, I know I can always count on your loving support. To my grandparents, Theo and John, I am proud to have had the chance to see your love and determination to make something better for the future, whether through hard work or taking care of those around you, I hope that I can do the same, and thank you for the opportunities you built for me. To all my cousins, aunts, and uncles, I feel truly blessed to have such a large and loving family, thank you for your love and support through all my years, it means more to me than you can know. To Piper, I am so grateful have you in my family, I can't wait to see how you grow and hope I can be an auntie that inspires you.

Lastly I would like to thank all of my friends, Deanta, Jon, Marissa, Ania, Abby, and all my Belegrim friends especially, your friendship has kept me afloat through this PhD, and I hope to be good to you as you have been to me.

DEDICATION

to all my family, from those who have gone before to those yet to come

Chapter 1

INTRODUCTION

1.1 Motivation

Underwater inspections of hulls and interior spaces are important in assessing the safety and readiness of seagoing vessels and are used to detect corrosion, biofouling, cracks, coating failures, and other damages. A summary of the existing tools and methods for underwater inspections can be found in [1, 2]. Human divers are unable to keep up with the demand for inspections due to necessary safety precautions such as limited dive times and large support teams [3], as well as from the sheer scale of naval vessels as shown in Figure 1.1. Human divers do excellent work and their skills and adaptability are needed for more complex underwater operations. However, large scale, generalized survey inspections are not a good use of their time and efforts when these can be done by a team of robots. Confined space inspections are also extremely dangerous both for divers if the tanks are filled, or for inspectors if they are emptied first. Between 1996 and 2024, 310 people have died in maritime confined space accidents, 31 of those were in 2023 [4]. By using a robotic solution in areas where the unique human abilities are not needed, we can reduce the workload on human divers and inspectors, which will in turn reduce the risks to human lives. Other benefits of underwater inspections include faster completion of work, better guidance for where maintenance will be required, and improved dry dock scheduling and estimation, which will all provide huge cost savings and improve fleet readiness.

Autonomous inspection challenges persist due to high operational costs (expensive, highly customized platforms); huge scales of operations (detecting small defects in large vessels while operating in the open waters); and lack of system robustness and resilience (robot hardware and autonomous decision-making failures). While ground and aerial robots have

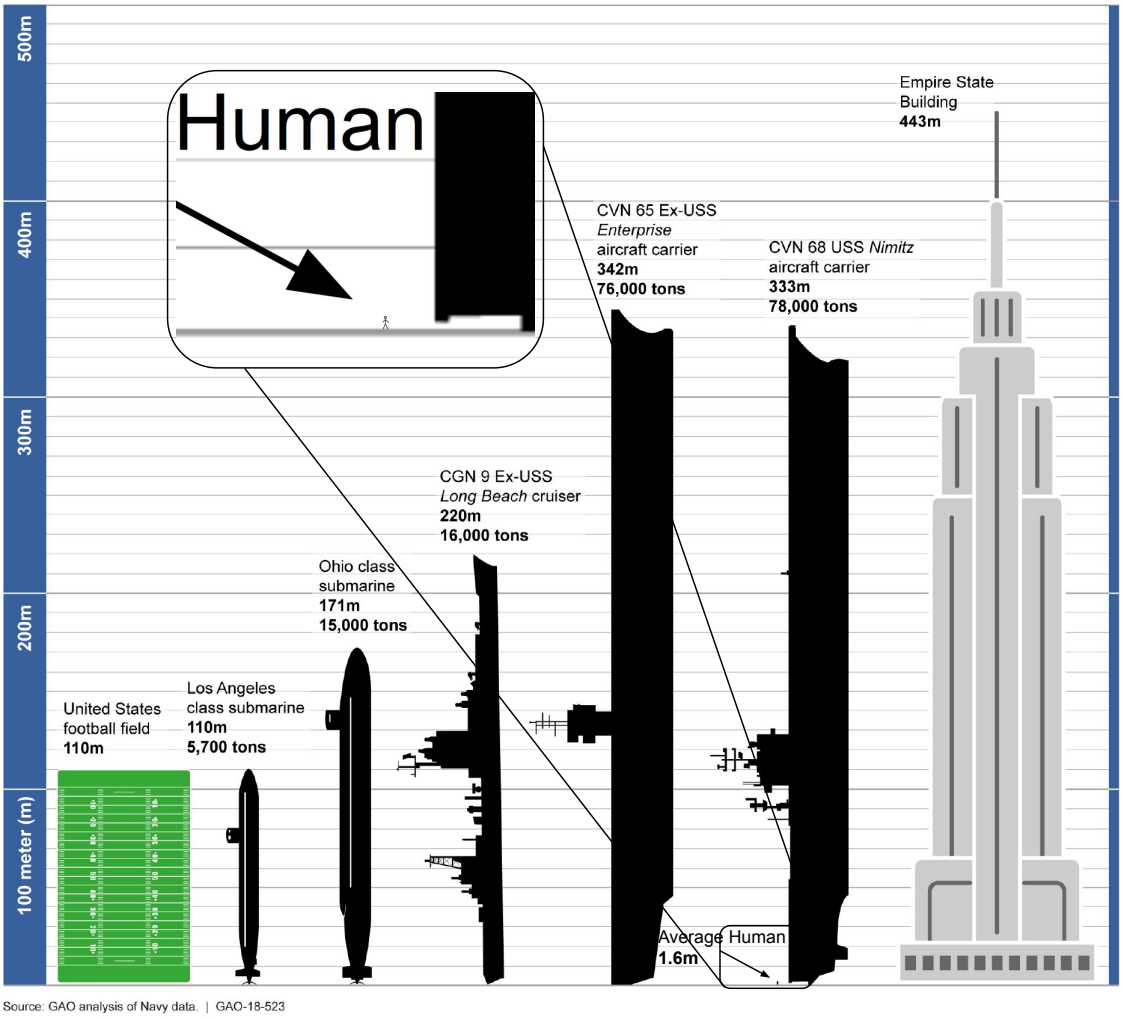


Figure 1.1: Size comparison of several naval vessels to common references.

highly functional platforms available on many different budget scales, the same is not true for underwater robots. There are very few underwater robots that are available on lower budgets, and those that are available are often not autonomous or have very specific, limited functionality. Underwater sensors are less readily available than their air based counterparts, in part due to the need for waterproofing and pressure tolerance. Maritime industries are plentiful (fishing, drilling, transportation, military, ecological, meteorological, etc.) and there are numerous applications for underwater robotics at the industrial level, but very few at

the consumer level, which leaves the lower budget options lacking.

Due to the high cost of underwater robots and the limits on underwater communication speed, many systems focus on the ability of a single robot rather than multi-robot systems. Shipyard conditions are very rough on robots, in addition to the hazards of saltwater immersion for electronics, robots will be dropped, kicked, shoved, dragged, and thrown, regardless of instructions otherwise. This treatment will result in robot breakdowns, leaving single robot systems non-functional while repairs and recalibrations are completed. Multi-robot systems are more robust to this kind of breakdown; even with several robots out of order, the system as a whole can still continue to function with the remaining robots. Multiple robots are also able to divide the work of large scale inspections, leading to faster completion overall. Another advantage of multi-robot systems comes from the ability to manage different robots with different capabilities all working together, allowing less capable robots to still be helpful in guiding more advanced robots to areas where their capabilities are required.

1.2 Underwater Robotics

1.2.1 Simulations

Accurate simulations of unmanned underwater vehicles (UUVs) are widely used in the design of multi-robot systems for the completion of underwater missions. However, there are only a few environments that also simulate the hull of a submarine or surface vessel along with the pertinent features for inspection and maintenance tasks. To the best of the our knowledge, there is no publicly available open environment for high-fidelity, large-scale simulation studies that require an entire hull of a vessel with access to internal structures such as water tanks. In 3, we report our effort to build upon the work of Project Dave [5] and others to create this rich simulated environment for future research and potential collaboration. We also discuss our preliminary work on using the simulation for a simple multi-robot inspection task of a realistic vessel hull.

Two different robots are featured: the Caracara from project Dave [5], and the BlueROV [6],

which is a widely available test vehicle. The simulation captures some of the important dynamics of the problem by including a vehicle motion model with parameters that are modified to reflect the unique dynamics of each vehicle. The various vehicle thruster and sensor configurations are also modeled. This modeling allows us to demonstrate task allocation and realistic navigation for a simple inspection routine involving a group of three robots, where each member of this group must visit all the allocated task locations on the outer hull of a submerged vessel.

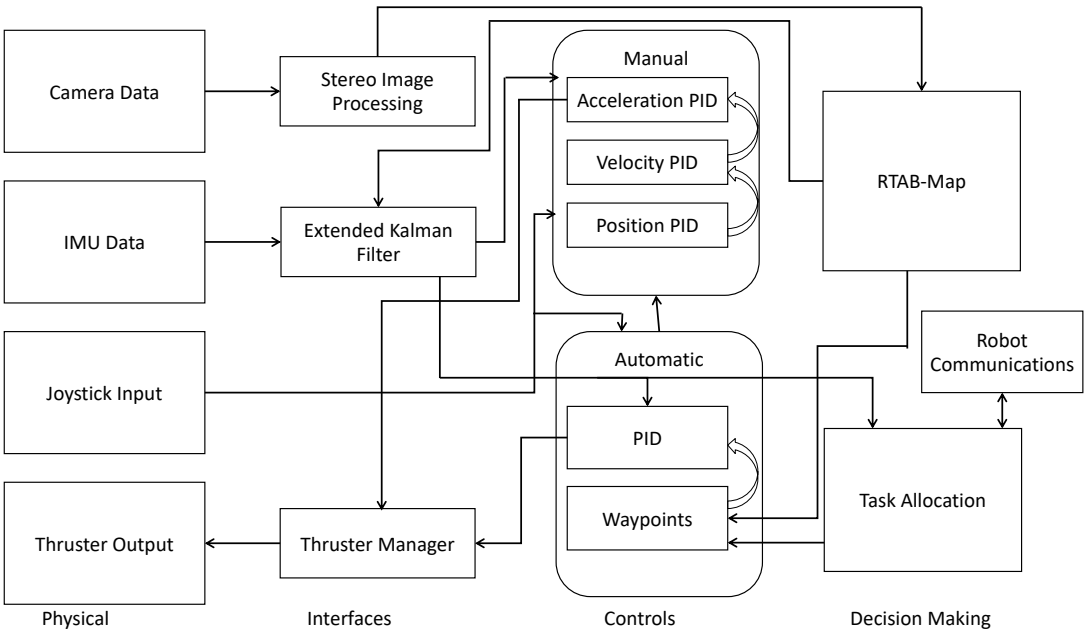


Figure 1.2: System diagram of all the cyber-physical processes in a simulated underwater vehicle.

A large part of preparing this open-source simulation is based on integrating existing ROS modules to work together, rather than duplicating the effort of others. Figure 1.2 shows a system-level diagram of all the cyber-physical processes that need to be executed together for the vehicle simulation. Stereo image processing is provided by the stereo_image_proc ROS package [7], the extended Kalman filter (EKF) is a part of the robot_localization ROS package

[8], and the Simultaneous Localization and Mapping (SLAM) is performed by RTAB-Map [9]. Outside these packages, most other features are from either Project Dave or the UUV Simulator, or altered versions of the ones provided by those sources. However, we develop a new task allocation framework to investigate modifications to state-of-the-art decentralized allocation algorithms to suit problems with these types of constraints. The task allocation framework also serves to connect the messages from the various other packages.

1.2.2 Sensors and Hardware

The range of underwater sensors varies from those used in air-based robotics due to the differences of how light and sound move through water as opposed to how they move through air. In air, sensors frequently use light to both detect the environment and communicate to other sensors. In water, light is quickly scattered rendering such sensors useless, however sound travels very well through water and is frequently used in similar ways to how light can be used in air-based environments. This idea is what leads to sensors such as underwater GPS (position is determined by triangulating an acoustic signal from an antenna or an array of receivers). Other frequently used underwater sensors include pressure sensors (for depth), DVLs (for velocity measurements calculated using sound reflected from the seafloor and the Doppler effect), side scan sonars (SSS) (for mapping surfaces using echos received as the vehicle moves forward), 2D sonar (for mapping one 2D layer of the environment using reflections of a single sound beam), and 3D multibeam imaging sonar (for 3D environment mapping, similar to the 2D sonar, but featuring multiple beams to create a point cloud of the environment). With proper waterproofing, calibration, lighting, and filtering, cameras are also used underwater. Stereo and monocular cameras work well and color cameras require some color correction but are also very frequently used. However, cameras that use structured light (ex. infrared) to determine 3D distances to images do not work underwater, due to the light scattering properties of water. Camera range is often significantly reduced underwater due to turbidity and lighting limitations. Many typical internal state sensors work under water, such as IMUs, encoders, and gyroscopes, though when conducting inspections near

steel hulls or tanks, it is important to use non-magnetic options.

Underwater communications are a particular challenge when dealing with underwater multi-robot systems. Wifi, bluetooth, and radio do not work well underwater due to light scattering. Underwater acoustic modems often only work directionally at very low speeds and are both large and expensive for use on UUVs. Wired connections are fast, but can easily become tangled or damaged. Tethers also add additional controls challenges as they create more drag depending on how much is in the water, especially when moving through currents. If wireless methods are used in an underwater robotic system, it is important that the system be powered onboard, fully autonomous and decentralized, since underwater wireless communications are incredibly slow and often unreliable. Tethered systems allow for remote operation and power delivery, and present some options for a centralized system, though with a multi-robot system, it is still best for each robot to be autonomous since human attention can only be on one thing at a time.

1.2.3 Cooperative Robotics

We consider a low-cost cooperative multi-robot system, and present a SLAM-based task discovery and allocation algorithm to generate the inspection tasks from environment meshes without prior geometric knowledge. This approach aims to address key underwater challenges: recovery from the loss or addition of perceptual and/or motor capabilities of robots, extremely slow and unreliable communications, and limited visibility due to turbidity and sparse features that complicate localization and mapping.

When considering the challenges of robotic coordination, the way robots decide which robots should do what portion of the job depends on the conditions the system needs to operate under. In the underwater inspections here, we consider each task as something that can be done by a single robot, one at a time. In many air based systems, reliable, high-speed communications can be assumed, allowing the task allocation to be centralized, where one computer or robot delegates tasks to the other robots in the system, allowing individual robots to have a lower computing power onboard. This does not work in situations with slow

and unreliable communications. Decentralized methods, with each robot making decisions based on their own algorithms and knowledge, even if incomplete, are needed for underwater inspections.

Our method is based on a decentralized multi-robot task allocation framework that uses very small message sizes and intermittent communications [10]. We extend this framework with an adaptive task discovery method that focuses the inspection effort on geometrically interesting areas identified during SLAM mesh generation. Specifically, we contribute a novel algorithm that automatically discovers inspection tasks from SLAM-generated meshes for heterogeneous multi-robot teams performing visual inspections of hulls or fluid-filled interior tanks. The inspection consists of visiting locations defined by position and orientation and capturing color images. Selected locations are referred to as tasks, while potential locations during discovery are candidate inspection points (CIPs).

1.3 Dissertation Outline

This chapter has discussed the importance and challenges of underwater naval inspections. It began with an overview of the reasons for underwater inspections, highlighting the dangers and scale of such operations. Then introduced a simulation tool further developed in 3. The discussion then turned towards challenges of underwater hardware, with a brief overview of various underwater sensors and communication challenges. After an introduction to underwater cooperative robots, this outline for the rest of this work is presented.

Chapter 2 covers literature related to the topics under discussion. Section 2.1 gives an introduction to the motion models, sensor configurations, and simulation options found in literature. Section 2.2 focuses on cooperation methods used both in and out of the water, especially those related to multi-robot tasking. In section 2.3, an overview of the principles of SLAM is presented, followed by a discussion of several relevant methods used in this work. After a discussion of works concerning underwater inspection coverage in section 2.4, chapter 2 is concluded with a discussion of path planning methods in section 2.5.

Chapter 3 of this dissertation presents our work on a simulation specifically for underwa-

ter multi-robot cooperation. This simulation is then used to test a task allocation algorithm based on a modified decentralized Consensus Based Bundle Adjustment algorithm. Chapter 4 builds on the task allocation algorithm to present Mapping-based Tasks for Inspection: Discovery and ALlocation (Map-TIDAL), which uses SLAM meshes to determine an appropriate set of inspection tasks for a set of robots, allocates them between the robots, dynamically adjusts bundle generation as additional tasks are added and uses the full set of tasks as a shared representation of the environment. Chapter 5 features an analysis of existing motion planning tools, within the context of underwater inspections and potential for integration into Map-TIDAL. Chapter 6 concludes the dissertation with a summary of the contributions, potential impacts, and future directions for continuing this research.

Chapter 2
RELATED LITERATURE

In this dissertation, methods for underwater multi-robot inspections are presented, often through expanding on, modifying and integrating existing methods to work with the unique challenges of underwater inspection. Section 2.1 presents relevant simulation tools and datasets. With those basics covered, section 2.2 provides an overview of robotic cooperation methods both in and out of the water, as an introduction to the task allocation algorithms developed later in the dissertation. As Map-TIDAL is based on the positioning and mapping done in the SLAM process, a general overview of SLAM with a specific focus on methods for underwater SLAM is presented in section 2.3. Map-TIDAL also features aspects of coverage based planning methods, these methods are discussed in section 2.4. This chapter finishes off with a review of path planning and collision avoidance algorithms in section 2.5 that will be relevant to our planned research.

2.1 Simulations and Datasets

Underwater robotics are typically very expensive, and break down easily. In addition, underwater environments are very challenging to test in, both due to unpredictable effects of weather, tides, currents, waves, and turbidity that are uncontrollable, and because any indoor facilities are increasingly costly with the more control one needs over the environment. Because of this, underwater simulations are needed to enable algorithm development and proof of concept before the investment is made into hardware and facility access. Another tool frequently used in early development is datasets, and while many options exist for air-based environments, those for underwater environments are not as common, harder to find, often very platform specific, and frequently lack ground truth verification tools. Underwater

simulation tools often focus on limited aspects of underwater robotics, simply due to the huge complexity of underwater environments. These typically come in three main forms: simulations and datasets focused on image processing and sensor realism, simulations focused on obtaining detailed motion models and accurate underwater dynamics based on fluid flow analysis, and generalized all purpose simulations that feature some aspects of both. Since this dissertation is not about detailed fluid dynamics analysis, this section focuses on sensor based and general purpose underwater simulations and datasets.

The VAROS dataset presented in [11] is an excellent example of a synthetic underwater dataset that focuses on accuracy of images and sensor data. Using the Blender modeling and rendering engine, they create a realistic underwater environment with both natural and man-made features, generate sensor data for several sensors including appropriate image adjustments for water conditions and light scattering, and generate a sequence of poses. They use a simplified motion model of a spherical body with a simple linear quadratic regulator based control system. While the simulation itself is not available, the dataset has been made available and provides simulated monocular RGB images, surface normal vector images, depth images, ground truth pose data, IMU measurements, and depth gauge data. The provided data is useful in many kinds of early stage testing, such as verification of SLAM techniques or task generation algorithms. However as it is a constant dataset on a single environment there are challenges it cannot address, including dynamic motion planning and controls, as the simulation does not feature things such as tides or currents or waves, or advanced vehicle configurations. Most importantly, the sensor data is limited to the sensors that were already modeled with the system, which does not include several staples of underwater sensing, such as DVL or any variety of sonar data, or stereo camera data.

Datasets on real underwater scenes are available, but most feature a limited set of sensors. The AQUALOC dataset described in [12] features greyscale monocular camera images, IMU data, and pressure data from three underwater scenes at varied depths. One of the biggest challenges in underwater datasets is developing accurate ground truth data to use as a

comparative baseline. The AQUALOC dataset does this through offline reconstruction of the environment using structure from motion and robust image matching. While this is not exactly ground truth data, evaluation of the average reprojection error shows the data to be high quality.

One of the most frequently used open-source underwater simulators is the UUV simulator presented in [13]. The UUV simulator uses ROS and Gazebo to enable modular underwater robotics simulations, including a wide variety of plugins for various underwater sensors (IMUs, DVLs, various cameras, multi-beam sonars, positioning transponders, pressure, etc.), hydrodynamics (thrusters, fins, currents, etc), controls and manipulation, and multi-robot options. Unlike previous simulators such as those described in [14, 15], the UUV simulator includes full consideration of the added mass, Coriolis, and centrifugal forces, and of hydrodynamic drag. By using these plugins rather than rigid models, the UUV simulator allows for complete customizability of any robot with known parameters and configurations, making it highly versatile. This simulator is a good option, but still has some gaps in functionality. One of the biggest challenges using the UUV simulator with new robots is finding the parameters for the added mass, Coriolis, and hydrodynamic drag forces, in addition, it does not take into consideration surface wave conditions or tether forces, and has limited simulation of underwater light sources and water turbidity effects such as floating particles.

Project Dave, [5], is built on top of the UUV simulator to add additional sensing and modelling ability. It adds increased sensor functionality such as 3D underwater LiDAR, Ultra-Short Base-Line (USBL) positioning systems, bottom occlusion models, object degredation models, an array of underwater manipulation tools, forward facing sonar, more complex ocean current models, and the ability to use bathymetry models to create underwater environments. Their underwater camera models accounts for distortions due to turbidity, light attenuation, and backscattering.

2.2 Cooperative Robotics

Allocation of tasks between sets of robots is necessary when robots are working together, whether in or out of the water. The multiple traveling salesman problem (mTSP) is a frequently used method to formulate the problem of task allocation, based on the traveling salesman problem (TSP). The TSP can be summarized as finding the minimal total path length when visiting a set of locations. One method of solving the 3D TSP can be found in [16], which uses a neighborhoods based approach to provide a feasible solution. Branch and bound techniques [17], mixed integer linear programming (MILP) [18,19], and heuristics based solutions [20] have been proposed as well. In the multi-robot case, less computationally demanding auction based methods are commonly used.

The decentralized consensus based bundle adjustment (d-CBBA) [21] is an auction based method using a two step process to allocate a set of single robot tasks between a set of robots that perform one task at a time. In this method, each robot completes an auction stage individually based on their current knowledge of the other robots, then communicates their bid information to the other robots in the system. With this new information, conflict resolution is performed to determine if the new information is relevant or not given the previous information stored on each receiving robot. If required, the process is then repeated taking into account the most up to date information. This process is described in more detail in chapter 3.

Similar auction based methods have been used to account for task reallocation as in [22], where auctions are used to distribute tasks that were discarded due to path conflict during airplane wing assembly. In [23], task scheduling with precedence was taken into account with a similar wing assembly problem involving multiple tools. Moving into underwater environments, [24] uses several variations of CBBA algorithms to account for path limitations of heterogenous AUV systems in mine countermeasure missions.

2.3 SLAM

Many advances have been made in the field of underwater SLAM, as summarized in [25, 26]. Due to the extremity of the underwater environment, many typical sensors used in aerial or ground SLAM solutions are unavailable or have limitations. Vision-based sensors that work well include both monocular [27] and stereo cameras, but care must be taken with shifts in color, water clarity [28], and image distortion [29]. Sonar systems are widely used for range finding-based SLAM [30–32], as acoustic waves have significantly lower attenuation rates in water and are not affected by changes in illumination conditions. However, they are often expensive. As an alternative, LiDAR systems have been used, although they work at different frequencies of light than their aerial counterparts [33].

Internal state-based sensors are often fused with external environmental sensors to overcome external sensing challenges. Doppler velocity logs (DVLs) use acoustic pulses to calculate velocity, compasses provide orientation data, inertial measurement units (IMUs) measure both linear and angular acceleration, and depth sensors calculate depth based on water pressure [25]. Often, multiple sensors are fused together to provide a robust solution. For example, [34, 35] combine visual, inertial, and sonar data; [36] uses visual, inertial and depth data; [37] couples DVL, visual, and inertial data; [38] fuses sonar, visual, inertial, and water-pressure data; and [39] presents a new dataset along with a solution encompassing stereo camera, IMU, DVL, and pressure sensor data.

2.4 Coverage Based Methods

Multi-robot (or more generally multi-agent) coverage problems appear in a wide variety of domains, including inspection and surveillance, search and rescue, agriculture, manufacturing, traffic management, law enforcement, and more. Although many solutions have been proposed, typically, the coverage problem is addressed either through a detailed consideration of the choice of agents' viewpoints over a complex but well-known geometry or through the partitioning of a simple bounded region for a set of agents with known capabilities.

For example, measurement uncertainties based on optical parameters are used to generate a set of viewpoints in [40]. An enhanced rapidly-exploring random tree is then used to compute the optimal subset of viewpoints that maintains full coverage and visibility of all the measurement points. A semiautomatic view planner is presented in [41], which uses 3D triangulation between a camera and a laser to actively calculate the viewpoints from an existing mesh model. A multi-robot boustrophedon coverage method for unknown 2D environments using line-of-sight communications among the explorer robots is developed in [42]. This method relies on cell splitting around obstacles to enable coverage using a heterogeneous system of a team of coverer robots and up to two teams of explorer robots. A sensor-based coverage path planning method that uses a generalized Voronoi diagram-based graph to model the energy capacity of multiple robots is presented in [43].

2.5 Motion Planning

Path planning is the necessary process of planning an exact, collision free path for a robot between two points. The open motion planning library (OMPL) [44] provides implementations of a number of different motion planning algorithms for situations with various constraints. OMPL divides their available planners into 3 categories: geometric planners (based on kinematic and geometric constraints), control-based planners (based on state propagation), and multilevel-based planners (based on simplifying state space representations), though some planners fall into multiple categories. The SPArse Roadmap Spanner algorithm (SPARS) [45] and one of the many variations on Rapidly-exploring Random Trees (RRT) [46] seem most promising initially for integration under the constraints of underwater inspections.

Rapid-exploring Random Tree (RRT) is an often adapted planning method for exploration based on computing collision-free trajectories considering both the kinematics and dynamics of the system and environment [46]. The RRT uses a tree grown forward from and initial position. At each iteration, a random state is generated and the nearest neighbor of the random state in the tree is calculated. By applying possible controls starting at the nearest neighbor, a set of successor states are generated. If a successor state satisfies velocity

constraints, has no collision and minimizes a metric relating the position to the random state, it is added to the tree. For either tree, if an added state is within a small neighborhood of any node in the other tree, a candidate solution trajectory has been found and is recorded. This method was tested on several models, including situations such as a 3D body with rotation featuring a 12 dimensional state space, similar to that used in the simulation of underwater robotics. While this method is simple and effective, it comes with no optimality conditions and can take longer computation times.

Variations of RRT that are analyzed in 5 are RRT-Connect [47] and RRT* [48]. RRT-Connect invloves growing two trees simultaneously. The forward tree generates successor states as in the original algorithm, but with extended control durations until an obstacle or boundary is reached, while the backwards tree generates successor states by integration over the small Δt, likewise extended to meet obstacles or boundaries as possible. RRT* features tree pruning to see the chosen successor state can be connected to other previous states to form a shorter path and pruning the nodes bypassed by these shorter routes to optimize the tree.

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) [49] is a planner based on discretization of the environment, optionally on several levels. The cells are sampled so that exterior cells are more highly weighted than interior cells, to promote exploration of the environment. After a new cell is chosen, motions are sampled and progress is determined by the increase in exploration of the cells.

The SPARS method for motion planning is based on building a roadmap that can be queried multiple times, rather than a tree that must be built new each time a new path is required. This method [45] uses both a sparse roadmap of the continuous space, and a dense roadmap of the local visible space using the $\delta-$PRM (probability road map) method. By leveraging the connections between visible regions, the spanner is able to build a graph based representation of the space, and use the visibility properties of the regions to ensure collision free navigation. In the implementation, it proves to equal the abilities of other roadmap based planners in terms of path quality, and exceed them significantly in reducing roadmap

memory space and query resolution time.

Chapter 3

UNDERWATER COOPERATION SIMULATION

3.1 Simulation Components

The simulation makes extensive use of several available tools for simulating robots in ROS Noetic [50] and Gazebo 11. The first of these is Project Dave [5], which is built upon the UUV Simulator [13]. This provides the framework for several robots, and simulates a variety of undersea conditions, hydrodynamics, sensors, and manipulators. The BlueROV simulation is specifically based on the UUV Simulator implementation found in [51], edited to function within the Project Dave environment.

3.1.1 CAD Model of Vessel Hull

The marine hull used in the simulation is a CAD model of a generic underwater vessel, as shown in Fig. 3.1. It was publicly released by the Naval Undersea Warfare Center (NUWC) Division Keyport for research purposes. This vessel model was developed through interviews with Navy inspection personnel, but is not an exact replica of any specific hull. The model, like a real hull, has limited features to use for robot localization, but does include sea chests and weld seams. It also contains an external access point to an internal water tank, which is another area of considerable interest in vessel inspection.

3.1.2 Robot Modeling

Although the files to represent Caracara and BlueROV vary, each robot requires detailed meshes for both visualization and collision avoidance. Note that bounding planes can be used for collision checking in the case of the RexROV. These collision elements allow the robots to interact with the simulated underwater environment, including external disturbances such

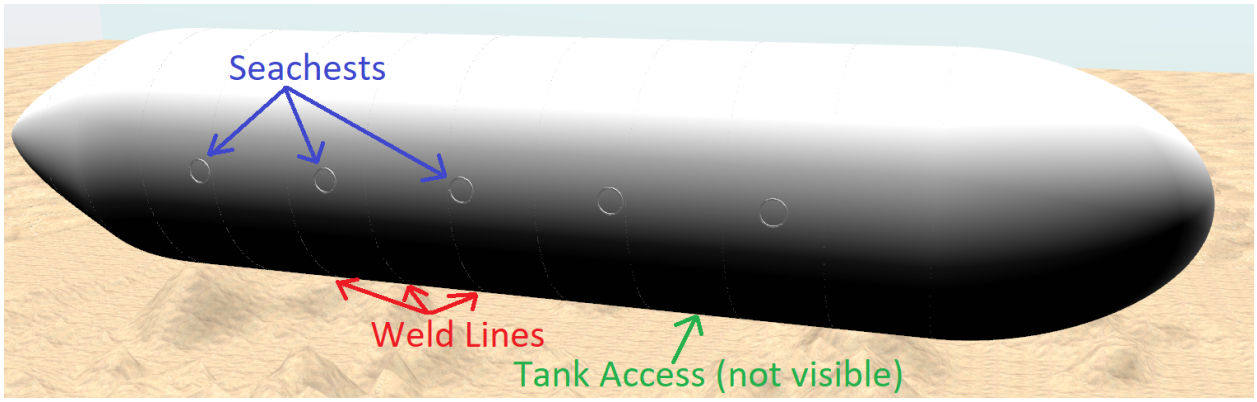


Figure 3.1: Simulation of a generic underwater vessel, featuring weld lines, sea chests, and an internal tank access point.

as waves and currents, and deliberate interactions such as control surfaces and thrusters. To facilitate these interactions, the hydrodynamic properties must be specified for the robots. This is accomplished by including an added mass matrix, and linear and quadratic damping terms per the Fossen model [52, 53]. It is also important to consider that the thruster responses are unique to each robot. The Caracara model is based on the scaled versions of those values specified in the existing program, while the BlueROV thruster responses are found from [6] with the added mass and drag matrices estimated based on the similar ellipse method from [54].

We then obtain a thruster allocation matrix to transform the thruster forces to the forces and moments acting on the six degrees-of-freedom robot body. This allows us to compute the individual thruster commands from the control inputs. The onboard sensors, namely an inertial measurement unit (IMU) and a stereo camera system, are then integrated into the robot models, and the range and noise parameters are set to match the available hardware capabilities. Due to limitations of the dynamics models, it is important that when the sensors are added, the balance of the robot is preserved; otherwise, the small weight added due to the sensors causes the robots to not behave properly, even if the robot's mass matrix already includes the weight of the sensors. We recommend making the inertial element for

the Gazebo instances of the sensors as small as possible in order to minimize this behavior, since the sensors' true weights should be already included in the robot. Unfortunately, the inertial mass cannot be made zero, as the sensor is not simulated in that case.

3.1.3 Robot Control

Two different control schemes can be used to operate the robots in the simulation: remote control through a joystick ('manual'), or trajectory control based on autonomy control processes ('automatic'). At any point, the user can enter any joystick command and resume manual control of the robots. This is provided as a part of the safety tools when switching between the simulation and real robot testing.

In the manual mode, the robot is controlled through a cascaded proportional-integral-derivative (PID) controller, provided as a part of the UUV Simulator. This can be used to control position, velocity, or acceleration from the human operated joystick. The regulated input signals are sent to the thruster manager to be appropriately allocated among the thrusters on the robot, depending on robot configurations.

In the automatic mode, the robot is controlled by the trajectory tracker from the UUV Simulator, which relies on both the task allocation and the SLAM modules. The SLAM module is responsible for providing accurate world knowledge, including robot odometry and optionally a cost-map for the environment. Task allocation provides a list of waypoints the robot needs to visit in order to complete the desired tasks. Depending on the chosen methods, the trajectory generator creates a path via the waypoints, then commands the robot based on the odometry feedback and a simple PID controller. Similar to the manual mode, the input signals are sent to the thruster manager for actuation of the robot.

3.1.4 Single Robot SLAM

Since we want this simulation to reflect the behavior of actual robots during an underwater inspection routine, all localization has to be separate from the ground truth. In order to model this properly, SLAM (and, in turn navigation) relies only on IMU and stereo camera

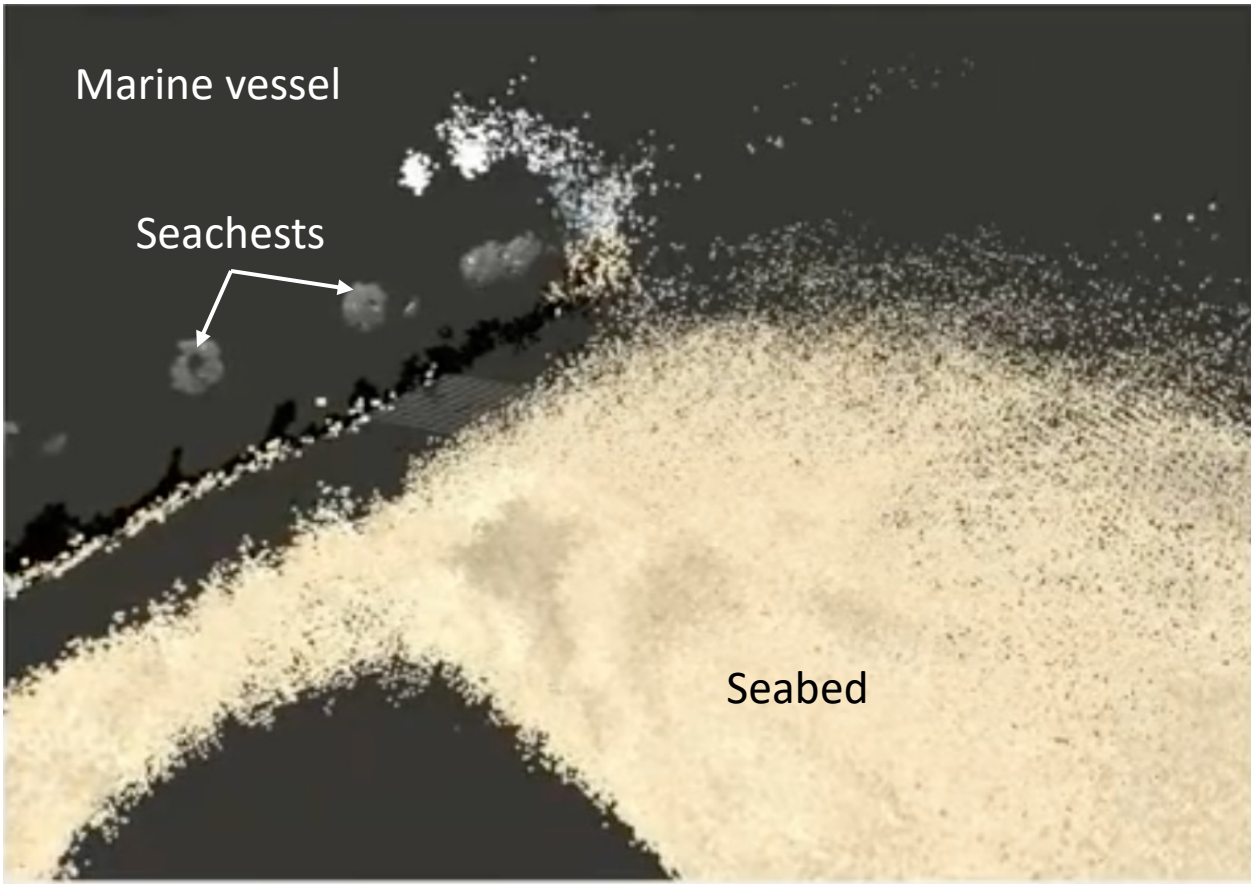


Figure 3.2: Partial map of an underwater environment generated by a BlueROV.

data. Other sources of positioning data, such as depth, and even a range measurement from a sonar buoy, will likely be included in the future depending on the performance of the other sensors.

We are currently using RTAB-Map [9] to perform visual-inertial SLAM, where the camera inputs are merged with the IMU data using the robot localization ROS package [8]. We are only using two of the default cameras and the default IMU provided in the UUV Simulator. We use the stereo image processing package [7] in ROS to handle the raw camera data before sending it to RTAB-Map. The speed of the IMU data allows immediate feedback for the processing of the control signals, while the SLAM data is used to construct the map and

correct the positional discrepancy that builds up from the inaccuracy of the IMU. Figure 3.2 shows a partial point cloud generated from a short run of the simulation.

3.2 Multi-Robot Task Allocation

3.2.1 Approach

Currently, we use a simple auctioning of pre-defined tasks on the outside of a vessel, where each task can be done by a single robot. Any task is considered complete as soon as the robot reaches the known location specified by the task. These assumptions represent the properties of simple tasks such as taking an image of a region of interest, which only requires maintaining a certain pose for less than approximately two seconds. Each robot submits a bid for a task that is proportional to the Euclidean distance of the task location from its current location. The bids also take into account factors such as the speed of the robots and the locations of the other assigned tasks. Future implementations will take into account the time to complete more complex tasks, tasks that require specialized tooling, and the possibility of multiple robots being needed for a given task.

The task manager package is implemented to connect messages from the different ROS packages, and simulates anticipated realistic restrictions on underwater inter-robot communication. Figure 3.3 shows the basic structure and flow of the task manager. The following sub-sections describe the important features of the task manager program that runs on each robot.

Task Manager Initialization and Overview

The task manager is a custom package that includes a launch file which launches a node for each robot in the system. Each node is responsible for managing several lists of tasks for the individual robot. The node also maintains important information about the capabilities of the robot. As shown in Figure 3.3, the robot stores lists of known tasks (colored in grey), bundle (blue), path (green), and completed tasks. The task manager also stores a

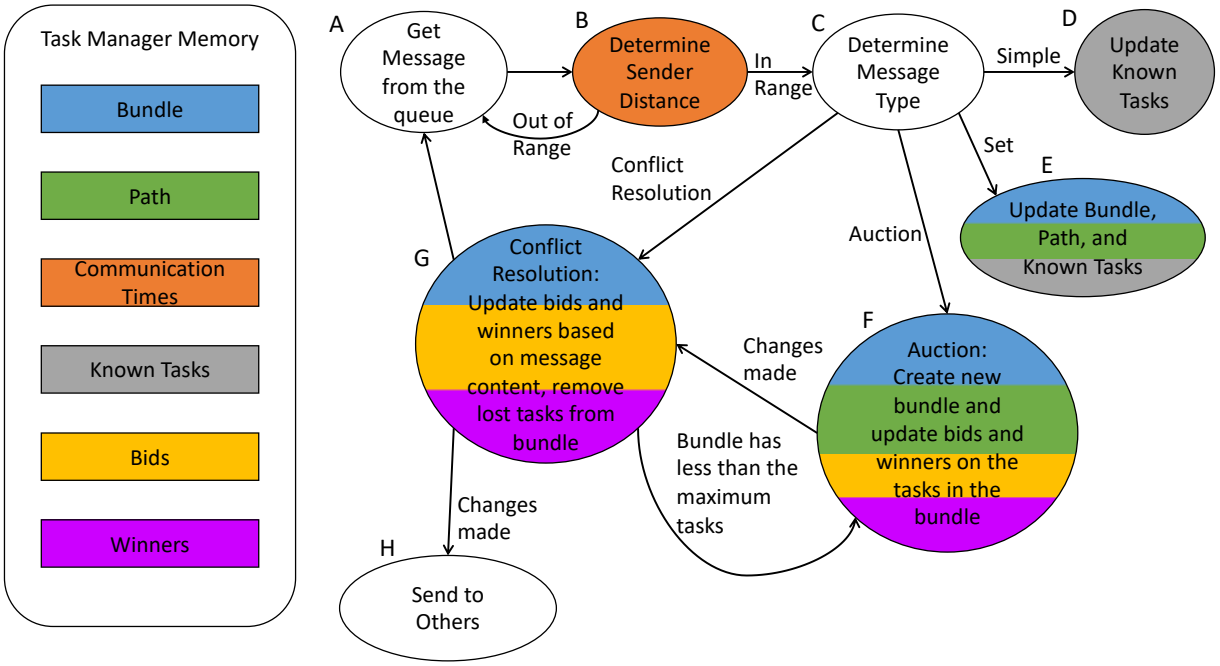


Figure 3.3: Process diagram for the task manager, including colors for which areas of memory are changed during different operations.

list of winning bids (yellow), the robots that made the winning bids (purple), and the times of the latest communications with the other robots in the system (orange). It also stores information about the robots, with each robot namespace corresponding to a particular robot ID, communication range, the number of tasks that can be present in the bundle, and the robot position.

Several ROS topics are used to facilitate communication among the robots: one subscriber topic to accept incoming lists from the other robots, another subscriber to receive the robot's odometry, and one publisher topic per robot to send outgoing lists - which are received via the other robot's subscribers, plus an additional topic for the user (either for debugging, or to be used in a future application).

Upon launch, each robot starts an instance of the task manager node, which monitors

and handles messages. When a message is received, it is added to the queue for handling. Depending on the message type, different actions are triggered, and the allocation process proceeds according to the diagram in Figure 3.3.

Task Message Structures

There four main message types: the Task, the Basic Task, the Task_list and the Basic Task_list. The format of the Task message type is designed to facilitate the auctioning process presented in [21], while allowing for new algorithms to be explored in future research. A Task message contains the ID of the robot who found the task ('FoundID'), the ID assigned by that robot to the task ('TaskID'), the pose that the robot needs to reach to complete the task, and a Boolean value to indicate task completion. To prepare for future implementations of the algorithms that will involve the discovery and addition of new tasks to the existing sets, both the IDs are needed to distinguish between the different tasks and avoid robots overwriting the unknown new tasks that the other robots may discover.

The second important message structure is the Task_list message type, which is used whenever tasks need to be transferred among the robots. In addition to a list of tasks, the Task_list message also includes the ID of the sending robot, the current location of the sending robot, a string representing the type of the message, that robot's list of bids and winners, and when that robot has last heard from each of the other robots. This information is important for the cycle of auctioning and conflict resolution. The current location of the robot is used in the process of simulating the limited communication range. The task manager can be configured to discard a message from a robot that is too far away based on the senders location (B in the Figure 3.3).

The last of the task message structures is the Basic Task, which can be used to reduce the size of the messages sent among the robots. This feature is especially important when coordinating with a limited bandwidth connection. Unlike the full Task message, the Basic Task only contains the FoundID and the TaskID, which can be sent in a message as small as 2 bytes (compared to the 30-31 bytes of a standard Task message). In implementations

that use the basic task structure, the full Task message would be sent only if requested by a robot. This situation would occur if the robot discovers that its neighbors have found new tasks while out of communication range - tasks that now need to be added to the allocation cycle.

This setup also requires a different list message type for the Basic Tasks under the same structure, and additional ROS topics set up to transfer these Basic Task_lists. The Basic List types also have reduced sizes, using rounded integers for the sender's position and bid values and an integer number code for the type string, rather than the strings and floats used in the full messages. In this paper, while the full Task message types are used by all the auction, conflict resolution and bidding algorithms, only a few small changes are necessary to recreate the full messages from the smaller messages when all the tasks are known to the robots. As such, the rest of this section refers to the full Task messages and lists, but would apply just the same to the Basic Task messages after reconstruction of the full Task message before processing.

Processing Messages

Every incoming message is placed in the queue for processing. When using *rospy*, a new thread is created for each publisher, which can lead to unwanted behaviors. For example, if two auctions are running simultaneously, both can add the same task to the bundle at the same time. This results in a duplicate task in the bundle, which leads to an under-allocation of actual tasks as the bundle contains duplicates rather than unique tasks. We prevent these overlaps by creating a single thread to handle the messages by pulling them from the queue (which is thread locked), rather than directly from the messages published on the ROS topic (which is not locked).

The following paragraphs describe the sequence depicted in Figure 3.3. When a message is pulled from the queue (A), the range of the sender is checked to see if the message can be successfully transmitted to the receiving robot (B). For testing purposes, we choose a range of 200 meters, which is based on the WaterLinked Modem M64, but other modems ranges

can also be used. When simulating the communication speed limitations, the bandwidth of this modem is 64 bits/second. Once the message is confirmed to be in range, the type of the message determines what the robot does with the message (C).

Currently there are four main functional types that are present in the Task_list messages: simple, set, auction, and conflict resolution. For the simple type (D), the tasks included in the message are added to the robot's list of known tasks if they are not previously known, and their bid and winning robot values are set to 0. This message type is most frequently used by the user to initialize the system, and in the future it will be used to add tasks to the robot's list as they are discovered. The set option (E) is similar to the simple type, but it also adds the list of tasks to the robot's bundle. This action is meant to allow the user to override the current list of tasks for a robot, but can also be used to initialize a set of tasks that will be acted on immediately by the robot.

The auction message (F) initializes the auction process for the robot, based on the robot's list of known tasks. This message type ignores whatever list is sent with the message, and only triggers the start of an auction, the end of an auction, and when the conflict resolution message is sent to the other robots in the system. The conflict resolution message type (G) begins the conflict resolution algorithm based on the content of the message. If necessary, the conflict resolution performs another auction, and if the order of composition of the allocated tasks changes significantly, it sends another round of conflict resolution to the rest of the robots. In addition to the functional message types that cause changes to the robot's planned trajectory or task list, other string messages can be programmed to allow the user to interject, monitor, or debug. This allows the user to understand what the system is planning, or evaluate the performance of different algorithms.

Auctioning

In our current simulation, we use a modified version of the consensus-based bundle algorithm (CBBA) presented in [21]. This method consists of two steps: the first one builds a bundle, path, and finds the optimal bids, and is referred as the auction; the second step handles the

decentralized conflict resolution. Algorithm 1 shows this auctioning algorithm for agent i at time t. We use $\mathbf{z}_i$ to represent the list of winning robots, $\mathbf{y}_i$ as the list of winning bids, $\mathbf{b}_i$ as the bundle, $\mathbf{p}_i$ as the path, L_t as the maximum length of the bundle, c_{ij} as the reward for robot i completing task j, k_{ij} as the bid robot i places on task j, J_i as the chosen task to bid on, and $\mathcal{J}$ as the list of tasks being bid on. Further, $n_{i,j}$ denotes the optimal location of the chosen task on the path, R is the completion value of each task, l_a^b is the distance from task a to task b, l_0^1 is the distance from the robot's current location to the first task in the bundle, $S_i^{\mathbf{p}_{i,n}(t)}$ is the reward for the path up through index n, $S_i^{\mathbf{p}_i(t)}$ as the total reward of the path without adding a task to it as defined in (3.1), and $S_i^{\mathbf{p}_i(t)\bigoplus_n\{j\}}$ is the total reward of the path with task j inserted into the path at location n. $\mathbb{I}()$ is an indicator function that returns 1 if the statement is true and 0 if the statement is false.

Algorithm 1 CBBA Auction phase for agent i at iteration t

$\mathbf{z}_i(t) = \mathbf{z}_i(t-1)$

$\mathbf{y}_i(t) = \mathbf{y}_i(t-1)$

$\mathbf{b}_i(t) = \mathbf{b}_i(t-1)$

$\mathbf{p}_i(t) = \mathbf{p}_i(t-1)$

while $|\mathbf{b}_i(t)| < L_t$ **do**

  $c_{ij} = \max_{n \leqslant |\mathbf{p}_i(t)|+1} \left(S_i^{\mathbf{p}_i(t)\bigoplus_n\{j\}} - S_i^{\mathbf{p}_i(t)} \right), \forall j \in \mathcal{J} \backslash \mathbf{b}_i(t)$

  $n_{ij} = argmax_n S_i^{\mathbf{p}_i(t)\bigoplus_n\{j\}}$

  $k_{ij} = R - \sum_{a=1}^{n_{ij}-1} \left(l_a^{a+1} \right) + l_0^1$

  $h_{ij} = \mathbb{I}(k_{ij} \geqslant y_{ij}), \forall j \in \mathcal{J} \backslash \mathbf{b}_i(t)$

  $\mathcal{J}_i = argmax_j c_{ij} \cdot h_{ij}$

  $\mathbf{b}_i(t) = \mathbf{b}_i(t) \bigoplus_{end} \{J_i\}$

  $\mathbf{p}_i(t) = \mathbf{p}_i(t) \bigoplus_{n_{i,J_i}} \{J_i\}$

  $y_{i,J_i}(t) = k_{i,J_i}$

  $z_{i,J_i}(t) = i$

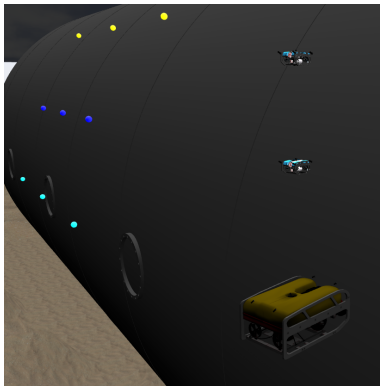
Algorithm 1 is modified from the original version in [21] to include a bid that is different from the reward function. This auction process involves calculating the optimal reward and best insertion point for each task on their list, and the transmitted bid for each task. The availability function h_{ij} is used to determine whether the robot is able to outbid the other robots in the system, based on the robot's knowledge of the existing bids. The best task is determined as the one that offers the highest difference in reward functions. This task is then added to the end of the bundle, and inserted at the best location in the path, and the bid and winner lists are updated. After this auction is completed, the conflict resolution is sent to the other robots in the system.

Bid Determination

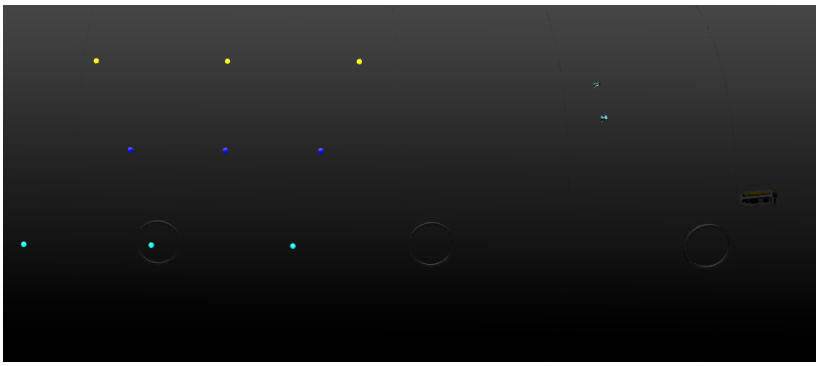
One important requirement for the convergence of the CBBA auction and conflict resolution steps is that the bid values for each task must have diminishing marginal gains (DMG), where the bid value is monotonically non-increasing as the task are placed towards the end of the path. Since our motion planner is not performing detailed trajectory prediction for obstacle avoidance at this stage, to reduce the computational complexity, we choose a simple reward and bidding function based on the Euclidean distance between the robot and the tasks. Each task is assigned a completion value, R, which, for now, is chosen to be the length of the tether, $l = 100$m. The total reward for a path is then

$$S_i^{\mathbf{p}_i(t)} = m(t) * R - \sum_{i=1}^{m(t)-1} \left(l_i^{i+1}\right) + l_0^1 \tag{3.1}$$

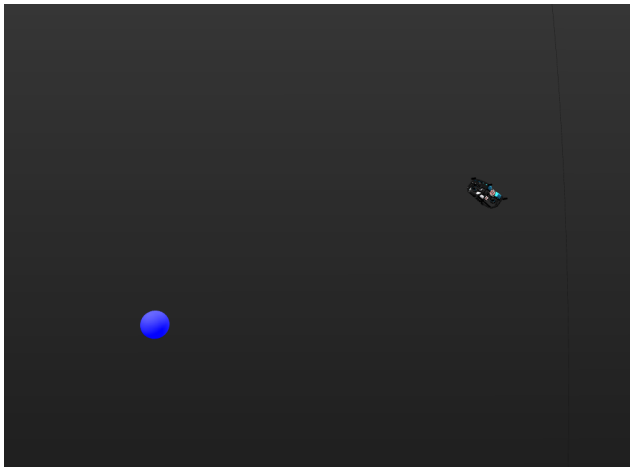
where, $m(t)$ is the number of tasks in the path $\mathbf{p}_i(t)$. Then, during the auction algorithm, the reward for adding an individual task to the list at a location is the difference between the total path rewards with and without adding the task. This reward is used to determine which task to bid on, while the bid is used to determine if the robot takes the task in its bundle. The value for the bid, k_{ij}, differs from the reward in that it only considers the completion value of the single task, and the sum of the distances to that task.



(a) Task allocation using auctioning



(b) Robots move to begin completing tasks



(c) A BlueROV approaches a task for completion

Figure 3.4: Allocation and execution of nine inspection tasks on the exterior hull of an underwater vessel by a team of three robots.

Since we want the robot to achieve some synergy among the tasks, the reward of adding a given task to the bundle can be higher at some later point in the path than at the first position, if we use just the Euclidean distance as the reward. Using the differences in the rewards as bids results in the bid function not being DMG; however, these values lead to

a more optimal assignment of tasks, since the robot always bids on a task that gives the shortest total travel distance. In order to make the bidding mechanism DMG, we change the bid values of the tasks to reflect the distance that has to be traveled along the path to reach the task at the given location. However, we retain the total path reward to choose which task the robot bids on. This preserves convergence in the algorithm (see Appendix A), while still providing a simple representation of the effort needed by the robot to complete the tasks.

In future implementations, we plan to consider more complex bidding functions that take into account additional factors such as avoidance of obstacles and tangling tethers, underwater currents, and changes in orientations to transition from one task to another.

Conflict Resolution

Our conflict resolution algorithm is the same as that presented in [21]. It starts when a robot sends its list of tasks, bids, winners, and communication times to a receiving robot. The receiving robot then compares this list to its own list, using the times the sender has heard from the other robots to determine who has the most up-to-date information. Afterward, it either leaves everything unchanged (in the case of agreement, or after determining it has the most recent information), or updates the task (after determining that the sender has more up-to-date information), or resets the task (if the values are conflicting that indicates both the sender and the receiver are wrong about who has the task). When a task is removed from the receiving robot's bundle and path, all the tasks added to the bundle after that task are also removed from the bundle and the path, since their bids are no longer accurate. If the bundle changes during the conflict resolution stage, a new auction is started. If changes are made to the bundle, or conflicts are ignored since the receiver has the most up-to-date information, the receiving robot send its updated list and information to the other robots and begins completing the assigned tasks.

3.2.2 Simulation Results

Figure 3.4 shows the results of a simple inspection scenario within our simulation, in which a team of three different robots must complete a given set of nine tasks by visiting each task location. The tasks are initially unassigned and the location of each task is represented by a small circular marker. After the task allocation is complete, the color of each circle changes to reflect the winner of that task. These task markers are simply visual representations for the user with no physical properties within the simulation. The allocation is seen in Figure 3.4a, with each robot being assigned a total of three tasks. Figure 3.4b shows the robots heading toward their first tasks. Figure 3.4c shows a BlueROV in the simulation approaching a task for completion. The auctioning and conflict resolution for this set of tasks, with each robot initially knowing about only 3 of the tasks, is completed within 28 conflict resolution messages.

3.3 Conclusions

While our simulation platform is quite preliminary, it demonstrates both the complexity of simulating underwater multi-robot inspection of marine hulls and the utility of doing so for autonomous robotics research. Using our platform as an initial test bed, researchers can investigate a variety of algorithms for multi-robot task allocation, simultaneous localization and mapping (SLAM), and motion planning and control, without having to purchase expensive underwater hardware. As this research continues, we anticipate our platform to be used extensively as we test more complex, geometry-based decentralized task detection, (re)-allocation [22], and prioritization [23]; cooperative visual SLAM [55]; and coordinated motion planning [56] methods. We also plan to consider additional features that are crucial to the success of the overall inspection mission, such as the communication bandwidth among the robots, water turbidity that clouds the field of view of the cameras, and so on.

Chapter 4

MAP-TIDAL

4.1 Method

4.1.1 Task Allocation

Map-TIDAL employs a decentralized task allocation framework based on the consensus-based bundle algorithm (CBBA) with domain-specific modifications for underwater multi-robot inspection [10]. The task allocation system operates through a distributed auction mechanism, where each robot maintains autonomous decision-making capabilities while coordinating with the team through as few message exchanges as possible to achieve consensus — a critical requirement for bandwidth-constrained underwater communication environments.

Each robot executes a local task manager responsible for maintaining: (1) known task inventories, (2) bid histories from neighboring robots, (3) communication timing logs, and (4) conflict resolution protocols. The auction process utilizes a reward function based on path optimization, where the reward for incorporating a given task into a robot's bundle is calculated as the difference in total path distance when the task is optimally inserted versus the original path configuration.

The bidding mechanism employs distance-based metrics, where each robot's bid represents the cumulative distance to reach that task location. This approach ensures convergence properties while providing computationally efficient effort estimation for task completion. Upon completion of the local auction phase, bid information is disseminated to neighboring robots for decentralized conflict resolution following the consensus protocols established in [21].

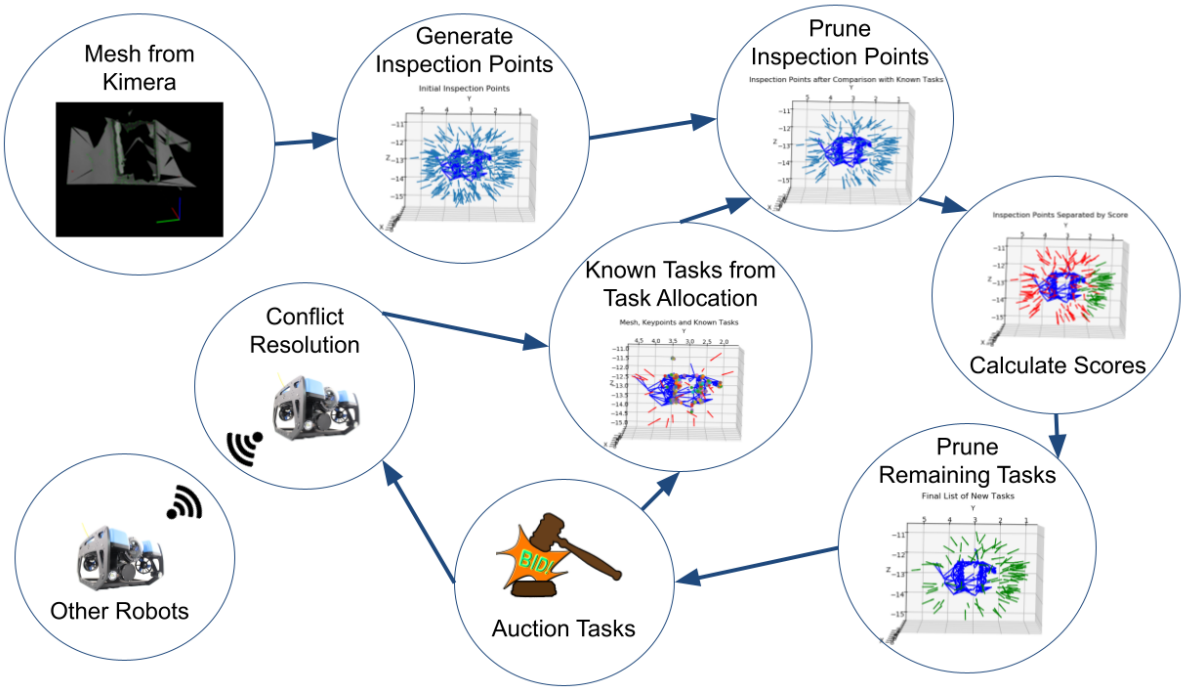


Figure 4.1: Flowchart of the task discovery and task allocation processes.

The integration of task allocation with SLAM and task discovery processes in Map-TIDAL enables fully autonomous operation, where newly discovered tasks are seamlessly incorporated into the existing allocation framework and communicated efficiently throughout the robot network. Figure 4.1 shows the integration between the task allocation and task discovery steps in the pipeline from the SLAM keyframe mesh through the communication and conflict resolution stages. The consensus-based task sharing inherently creates a distributed understanding of spatial coverage across the inspection area. This shared awareness of existing task locations enables intelligent mesh analysis during task discovery, where newly generated candidate inspection points can be evaluated not only for their current viewpoint coverage but also for their contribution to overall coverage gaps identified through the collective task inventory.

4.1.2 Task Discovery

In this work, we use Kimera-Multi [57, 58] to fuse IMU, stereo vision, and DVL data. While originally intended for aerial and ground environments, Kimera-Multi is able to handle underwater sensor fusion. With properly calibrated cameras, Kimera-Multi calculates location and generates a lightweight 3D mesh from very sparse keypoint data using structural regularities [59]. This mesh is ideal for underwater inspection since visual features underwater are typically clustered only in areas with strong visual differences, such as patches of barnacles or algae, a scratch or rust damaged area, or a piece of complex geometry like a weld line or sea chest. These visual features are the exact things that need the most attention during inspections. In areas with large, smooth sections with very few, very sparse features, the robots will rely on the DVL and IMU for localization and the mesh will be less dense and will generate fewer tasks.

There are four main steps in the task discovery part of the Map-TIDAL process which begins whenever a keyframe mesh ($\mathcal{M}$) is produced by Kimera.

1. **Candidate inspection point generation:** For each triangle in the mesh $\mathcal{M}$, candidate inspection points (CIPs) are generated by projecting along the surface normal by the ideal length (L_i), as shown in Fig. 4.2a. L_i is optimized based on camera focal length and water turbidity to maximize image quality while maintaining safe standoff distance from the vessel surface.

2. (a) **Pruning based on existing tasks** (when $T_e \neq \emptyset$): Each CIP undergoes spatial filtering against existing tasks T_e using dual geometric constraints: radius of exclusion (r_{ex}) and angle of exclusion (θ_{ex}). A CIP is discarded if any existing task lies within both the spherical radius r_{ex} and the angular cone θ_{ex}, ensuring that retained points provide sufficiently distinct viewpoints for inspection coverage, as shown in Fig. 4.2b. If T_e contains only points outside of r_{ex} from any CIP, step 2b is performed instead.

OR

(b) **Pruning based on self-comparison** (when $T_e = \emptyset$): When no prior tasks exist within r_{ex} of any CIP, CIPs undergo self-comparison using reduced thresholds ($p_s = 70\%$ of r_{ex} and θ_{ex}) to eliminate redundant inspection points within the candidate set, preventing over-dense task generation in highly featured areas.

3. **Keypoint scoring:** Each remaining CIP is evaluated by projecting the camera field-of-view (FOV) cone and counting intersected mesh keypoints shown in Fig. 4.2c. CIPs scoring above the threshold (t) percentage of maximum possible keypoints in the local mesh region are promoted to confirmed tasks, prioritizing geometrically complex areas that correlate with potential defects.

4. **Final spacial pruning:** Remaining sub-threshold CIPs undergo iterative comparison against the newly confirmed task set using the spatial exclusion criteria from Step 2a. Tasks are processed sequentially, with each addition expanding the exclusion zones for subsequent candidates, ensuring optimal spatial distribution in the final task set as shown in Fig. 4.2d.

Once the new task list has been finalized, it is sent to the auction phase of Map-TIDAL, which in turn sends the new task information to the other robots in conflict resolution, and updates the master list of tasks. This full Map-TIDAL process is described in algorithm 2 and in Fig. 4.1.

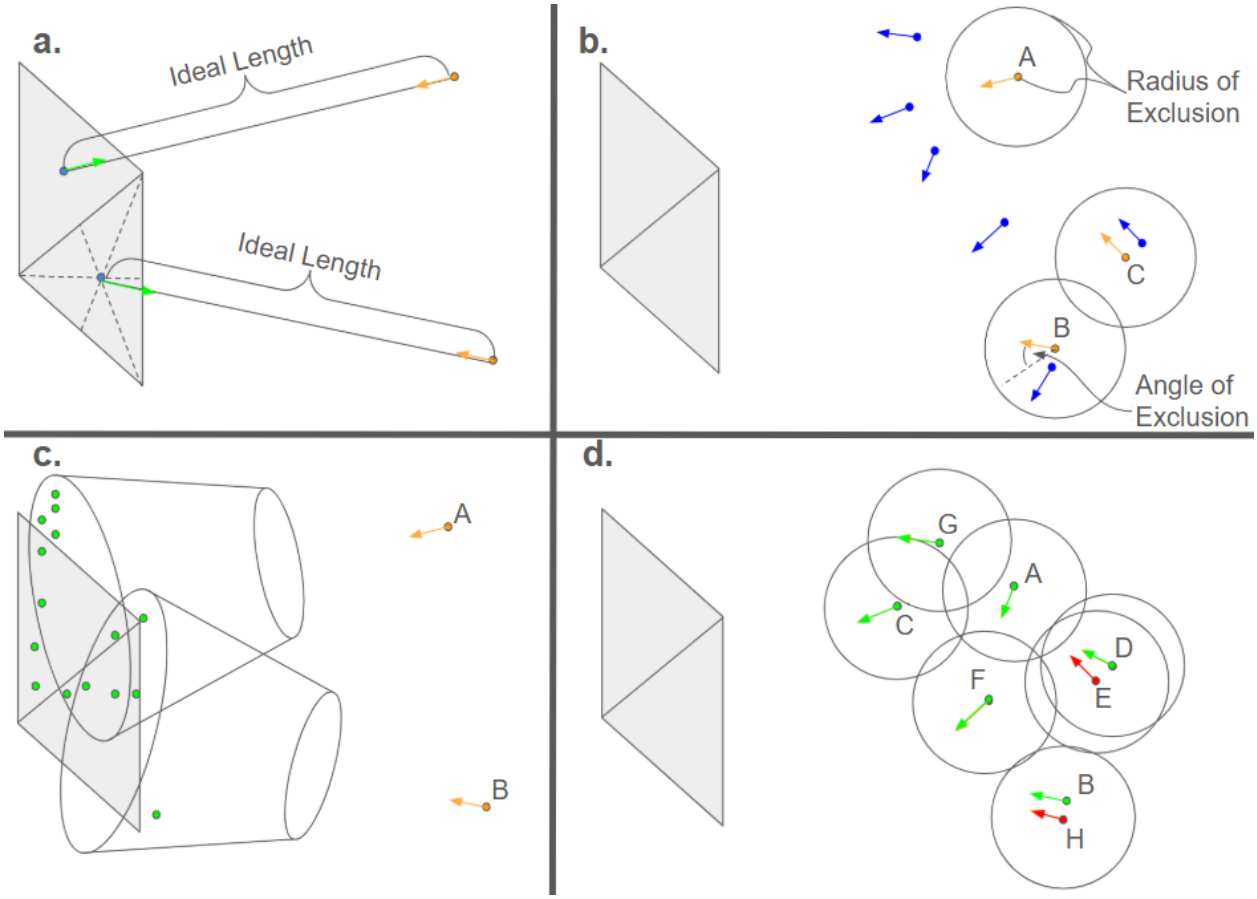


Figure 4.2: **a.** Inspection point (orange) generation from mesh (grey) using the centroid (blue) and normal (green). **b.** Comparison of inspection points (orange) to existing tasks (blue), point A has no tasks inside the radius of exclusion, point B has a task in the radius of inclusion, but the angle is outside of the angle of exclusion, point C has a task that is within both the radius and angle of exclusion, and will not be removed. **c.** Calculation of keypoint scores based on the camera field of view. The score is the total of the keypoints (green) that fall in the field of view for the inspection point (orange) normalized by the number of keypoints. **d.** Final pruning on remaining inspection points, done in alphabetical order tasks E and H are removed, the remainder become tasks.

Algorithm 2 Task Discovery Process

Inputs: Kimera mesh ($\mathcal{M}$), existing task list (T_e)

Parameters: L_i, r_{ex}, θ_{ex}, t, FOV

Output: New task list (T_{new})

▷ $\hookleftarrow$ indicates appending to the end of a list

▷ $\emptyset$ indicates an empty list (ordered)

from $\mathcal{M}$ extract keypoints ($\mathcal{K}$) and polygons ($\mathcal{P}$)

$C \leftarrow$ GENERATE POINTS($\mathcal{P}$) ▷ step 1

if T_e is $\emptyset$ **then**

    $C \leftarrow$ PRUNE(C,$\emptyset$,2) ▷ step 2a

else

    $C \leftarrow$ PRUNE(C,T_e,0) ▷ step 2b

$T_{new} \leftarrow \emptyset$

$C_{remains} \leftarrow \emptyset$

$S \leftarrow$ CALCULATE SCORES($C, \mathcal{K}$) ▷ step 3

for i **in** $length(S)$ **do**

    if $S[i] \geq t$ **then**

        $T_{new} \hookleftarrow C[i]$

    else

        $C_{remains} \hookleftarrow C[i]$

$T_{new} \hookleftarrow$ PRUNE($C_{remains}, T_{new}, 1$) ▷ step 4

return T_{new}

Algorithm 3 Pruning

Parameters: L_i, r_{ex}, θ_{ex}, FOV, p_s

function PRUNE(C, C_{comp}, n)
  $precheck \leftarrow 1$ ▷ Boolean that marks if prior tasks exist within r_{ex}
  $C_{pruned} \leftarrow \emptyset$
  $C_{compare} \leftarrow \emptyset$
  if $n = 0$ **then** ▷ C is compared to only C_{comp}
    $C_{compare} \leftarrow C_{comp}$
  else if $n = 1$ **then** ▷ C is compared to C_{comp} w.r.t. own tasks
    $C_{pruned} \leftarrow C_{comp}$
    $C_{compare} \leftarrow C_{comp}$
  else if $n = 2$ **then** ▷ C is compared to itself
    $C_{pruned} \hookleftarrow C[0]$
    $C_{compare} \hookleftarrow C[0]$
    $r_{ex} \leftarrow r_{ex} \times p_s$ ▷ precheck strength
    $\theta_{ex} \leftarrow \theta_{ex} \times p_s$
  for X **in** C **do**
    $d \leftarrow \emptyset \quad \alpha \leftarrow \emptyset$
    $d \hookleftarrow distance(X, Y)$ **for** Y **in** $C_{compare}$
    $\alpha \hookleftarrow angle(X, Y)$ **for** Y **in** $C_{compare}$
    $keep \leftarrow 1$ ▷ Boolean indicating whether to keep a task
    for i **in** 0 to $length(d)$ **do**
      if $d[i] < r_{ex}$ **then**
        $precheck \leftarrow 0$
        if $\alpha[i] < \theta_{ex}$ **then**
          $keep \leftarrow 0$
          break

```
        if keep = 1 then
            C_pruned ↩ X
            if n = 1 or n = 2 then
                C_compare ↩ X
    if n = 0 and precheck = 1 then
        C_pruned ← PRUNE(C, ∅, 2)
    return C_pruned
```

Algorithm 4 Additional Functions

```
function GENERATE POINTS(P)
    for x in P do
        c ← centroid(x)
        n ← normal(x)
        p ← c + L_i × n
        θ ← −1 × n                          ▷ angle expressed as a quaternion
        C ↩ (p, θ)
    return C

function CALCULATE SCORES(C, K)
    S ← ∅
    for X in C do
        FOV_t ← transform(FOV, X)
        T ← Σ (Delaunay(FOV_t).find_simplex(K) ≥ 0)
        S ↩ T/length(K)
    return S
```

4.2 Experimental Results

4.2.1 Physical Setup and Data Collection

To validate Map-TIDAL, we collected two sets of in-situ data in an OSB test tank with saline water. We used a team of two BlueROV2 [6] robots customized with an NVIDIA Jetson Xavier AGX on-board processor, a WaterLinked A50 DVL (only on the robot of interest), and an OAK-D stereo depth camera. While the tether is used as a safety measure for monitoring and manual control takeover if required, all necessary processing and recording are done on the Jetson.

In the first test, a short video and IMU data with the robot manually held in a stable position were collected. This video data was then used with offline Kimera processing to determine initial feasibility and algorithm development. Figure 4.3 shows the results of this initial algorithm development on a single mesh frame featuring 326 triangles. The initial parameters were chosen as $L_i = 1.5$ m, $r_{ex} = 0.8$ m, $\theta_{ex} = 0.1$ rad, and $t = 0.4$. The left image with keypoints and the mesh are shown in Fig. 4.3A and Fig. 4.3B, showing the concentration of the keypoints on the geometrically and visually interesting places such as the ladder feet and edges of the window and frame, and the two visible edges of the corner. Using these values and an initial list of 21 tasks selected randomly from the CIPs, the number of CIPs is reduced from 326 to 98. This can be further reduced with appropriate parameter tuning.

In the second test, the robot was allowed to rise slowly a distance of about 0.5 m, and the first 7 keyframes were analyzed to find the ideal parameters for this environment. In pretest observations, with high water clarity and good, indoor lighting at the saltwater tank, we found the OAK-D provided the most accurate point clouds when it was positioned approximately 1 meter from the surface it observed, this was set as the L_i value.

Parameters r_{ex}, θ_{ex}, and t were varied separately with the goals of minimal uninspected area, appropriate levels of overcoverage, aiming for an average number of views in the 2-2.8 range, and between 15-25 tasks. We observed that θ_{ex} produces the number of tasks in that

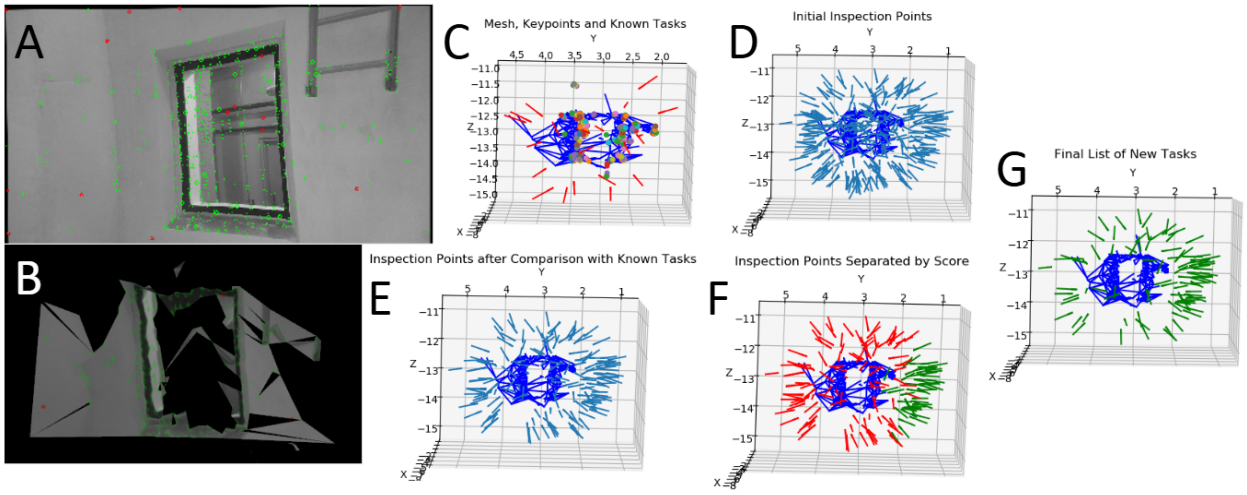


Figure 4.3: **A.** Left camera image with tracked keypoints overlaid in green and untracked keypoints in red. **B.** Kimera mesh with A. as texture. **C.** Initial tasks shown as arrows in red, mesh in blue, keypoints as rainbow dots. **D.** The 326 CIPs generated in teal. **E.** The 165 CIPs remaining after the first pruning in teal. **F.** The 49 tasks kept from their keypoint scores in green, with the remaining 116 CIPs in red. **G.** The 98 tasks kept after the final pruning.

range for values between 0.25 and 0.45 rad, r_{ex} produces tasks in that range between 1.5 and 2.0 m, with the rate of adding additional tasks higher with lower values of r_{ex} and θ_{ex}. The threshold has little effect on the rate of adding tasks, but a large effect on the initial number of tasks, with higher thresholds producing fewer tasks overall, and very little change in the number of tasks produced for any threshold variance over 80%. The results of varying each individual parameter are presented in figure 4.4.

4.2.2 Analysis

To analyze the effective coverage of the new task locations, we modeled the region of the test tank used during testing. This region includes a flat section with an inset window, two angled sections, and a short part of the longer sides of the tank, as shown in Fig. 4.5. The

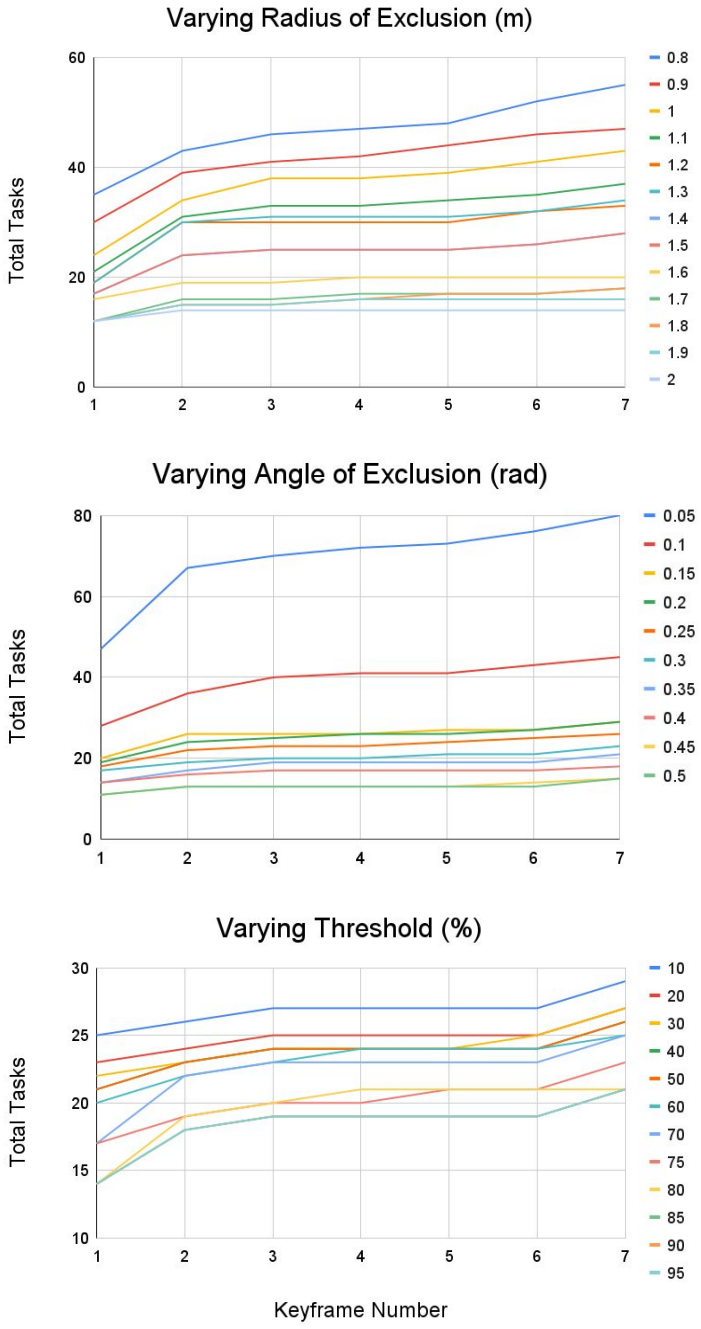


Figure 4.4: Total number of tasks over seven frames with: Top: varied r_{ex} values, $\theta_{ex} = 0.3$ rad, and $t = 75\%$; Middle: varied θ_{ex} values, $r_{ex} = 1.5$ m, and $t = 75\%$; Bottom: varied t values, $\theta_{ex} = 0.3$ rad, and $r_{ex} = 1.5$ m.

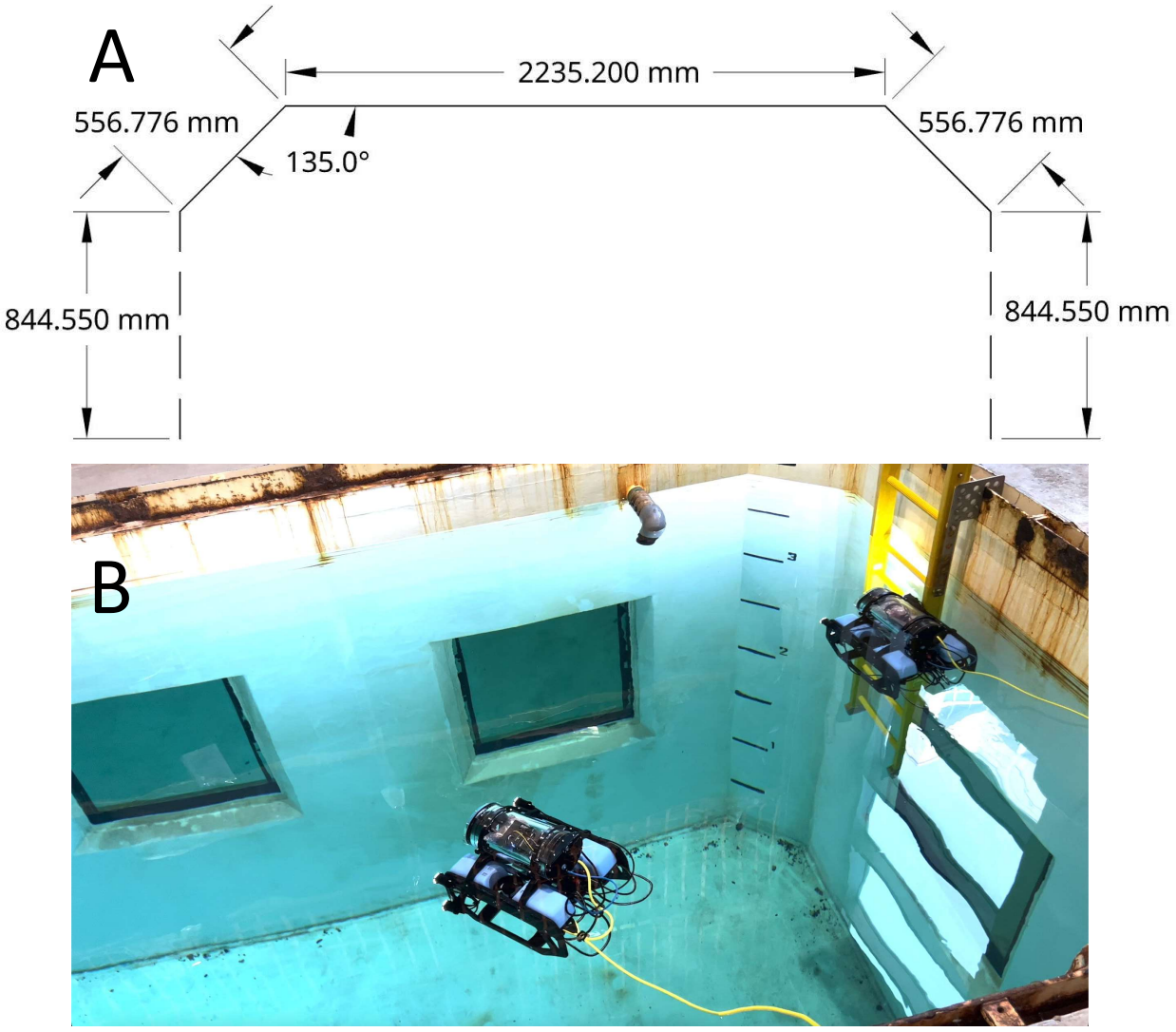


Figure 4.5: **A.** Top view of the test tank with dimensions. Solid lines show geometry used to generate inspection points for Voronoi and sweeping pattern methods, dashed lines show the edges of the voxelized region used to evaluate method effectiveness. **B.** Test tank with two modified BlueROV2s with the main robot at the center.

bottom of the tank and the surface of the water are not included in the model. The model is voxelized using Open3D libraries.

For each point in the discovered task sets, the FOV cone is transformed, and any voxels within the cone are shaded red. If a voxel is seen from multiple positions, the shade of red is made brighter with each point that views it. By analyzing the distribution of the number of times each voxel is seen, we can compare the effectiveness of the variations of each parameter within their effective ranges.

Table 4.1: Average Views per Voxel, Number of Tasks, and Percent Uninspected

t	r_{ex} (m)	θ_{ex} (rad)					
		0.25			0.3		
		avg	num	%	avg	num	%
70%	1.5	4.26	27	1.49	3.97	25	3.71
	1.6	3.63	24	7.60	3.19	21	8.06
	1.7	2.85	22	4.28	2.30	18	21.16
	1.8	2.71	19	6.18	2.79	19	5.77
75%	1.5	4.05	26	3.05	3.50	23	5.69
	1.6	3.12	22	11.73	2.83	20	9.53
	1.7	2.56	20	3.81	2.38	18	6.79
	1.8	2.48	18	6.84	2.55	18	6.42
80%	1.5	3.26	23	4.06	3.06	21	4.40
	1.6	2.62	22	11.90	2.31	19	8.07
	1.7	4.06	25	3.71	3.15	21	5.13
	1.8	3.78	23	6.35	3.09	20	6.45

Avg indicates the average views per voxel, num the number of tasks generated with that set of parameters, and % the percentage of voxels that receive no views. Green values indicate optimal performance, yellow values acceptable performance, and red values indicate significant overcoverage.

Table 4.1 shows the average number of views per voxel, the number of tasks generated, and the percentage of voxels with zero views for various values of r_{ex}, θ_{ex} and t. Ideal values with less than 5% uninspected are shown in green, and for these values, between 15 and 25 tasks, and an average of less than 2.5 views per voxel are also shown in green. Yellow values indicate values outside of that range, but under 30 tasks or with an average lower than 3.5, while red values indicate averages over 3.5. The best coverage is achieved at $r_{ex} = 1.7\ m$, $\theta_{ex} = 0.25\ rad$, and $t = 75\%$ but other values also show acceptable results. In general, lower r_{ex} and θ_{ex} values lead to higher numbers of tasks with better coverage, but a higher average coverage.

4.2.3 Comparison Metrics

Metrics on inspection quality such as the coverage percent and averages can be easily compared, but finding a metric that also shows the distribution of overcoverage can be a challenge. Some overcoverage can be desired, especially around complex geometry to ensure that all angles are viewed to reveal all defects, but too much overcoverage is a sign that there are more inspection points than needed- leading to wasted time and energy. But what is the ideal balance that ensures coverage without too much overcoverage? While many options are present, we have chosen to examine the distribution of number of views per voxel compared to the gamma distribution. While this function is typically used for wait time variables, the inherent 0 minimum, left skew, and exponentially decreasing tail make the gamma distribution good for our purposes as well. The gamma distribution has the probability density function in equation 4.1.

$$f(x;\alpha,\beta) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} x^{\alpha-1} e^{-x/\beta}. \tag{4.1}$$

By choosing the shape parameter α and the scale parameter β, we can modify the shape of the distribution to fit our desired inspection requirements- that there be a the least number of voxels inspected 0 times, that the peak value be between 1 and 2, and that the number of voxels with large amounts of overcoverage be low. The peak value of the gamma distribution

is at $\beta(\alpha-1)$, so if we set a desired peak value, either parameter can be used to tune the whole distribution. Using 2 as the peak value and setting $\alpha = 4$ gives $\beta = \frac{2}{3}$, and checking with the cumulative density function, this choice of parameters gives less than 1% of voxels being inspected less than 0.5 times. Since the gamma distribution is continuous while the number of views is discrete integer values, when discretizing data from the gamma distribution, this would be the cut-off for voxels that would be inspected 0 times using integer centered bins.

4.2.4 Comparisons

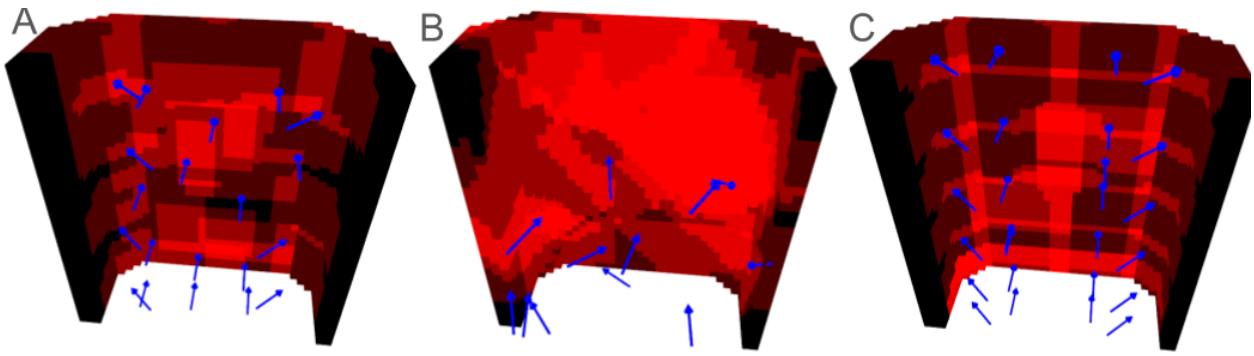


Figure 4.6: Simulated coverage analysis for **A** Voronoi, **B** Map-TIDAL, and **C** sweeping patterns with task locations shown in blue and voxels colored brighter red with each time they are viewed.

Some understanding of other prevalent area coverage methods is required to compare them to the performance of MAP-Tidal. Voronoi diagrams are made by dividing a space into regions that contain all the points closest to a point within a set of points. A survey of Voronoi methods for robotic coverage is provided in [60]. The generation of the points used to generate the Voronoi diagram can be generated by using robot start positions, random generation based on various distributions, or other various methods. The Voronoi partitions can be generated with a constant cost function or on a distributed cost function over the region based on previous knowledge of the environment. From the starting positions, the

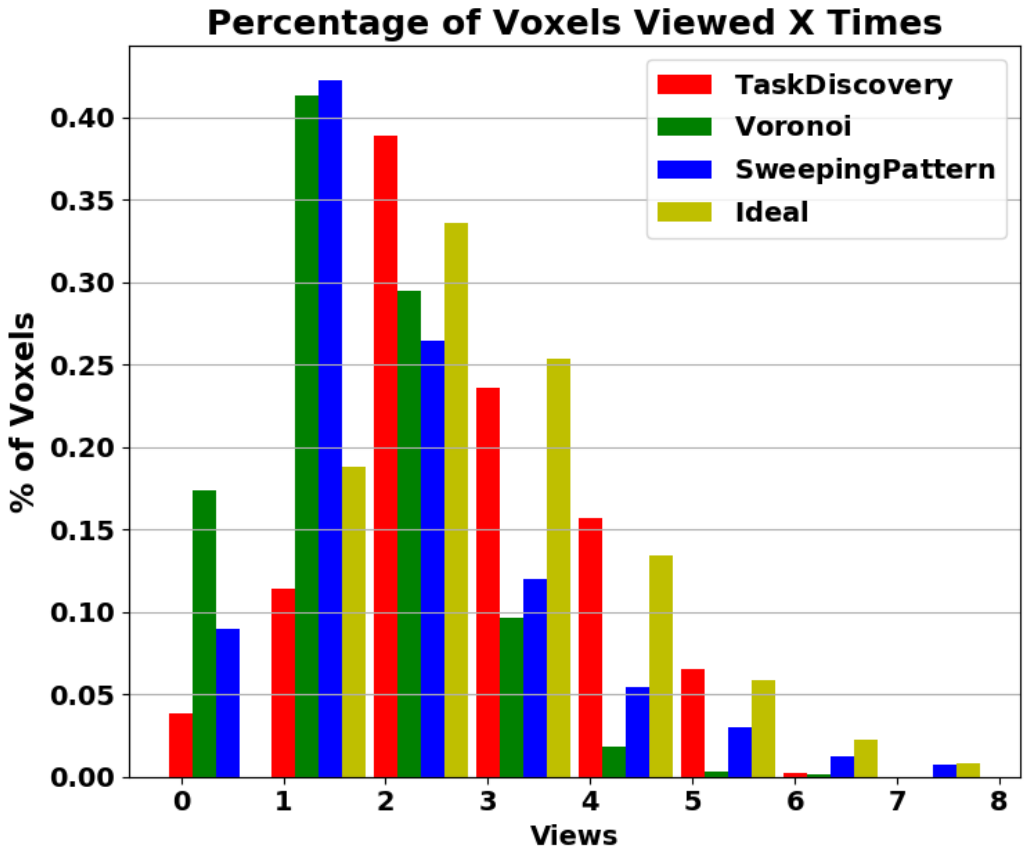


Figure 4.7: Histogram showing normalized views per voxel for different task generation methods compared with the discretized ideal gamma distribution.

diagram can be optimized using Lloyd's method, which moves each point toward the centroid of the corresponding Voronoi partition to recompute the new partitions. For our comparisons, we use WebSVG's Voronoi editor [61], which uses a seed generation method that controls the spread and regularity and allows for weighted distributions and wall avoidance.

Boustrophedon patterns are based on sweeping linearly back and forth over a space. In [62], the optimization of sweeping patterns occurs over a space that can be represented by a 2D space with holes as no-fly zones. Bahnemann et al. develop a tool to calculate optimized

paths using a generalized traveling salesman problem framework that can be modified during runtime as a user adds additional no-fly zones. While this implementation was originally demonstrated using flying robots searching a hillside for mines with a set altitude offset, it can be applied to other situations where a 3D space can be represented by a surface with an offset robot, such as our inspection problem.

For both the Voronoi and sweeping patterns, we use a projection of the rectangular surface onto the wall of the tank with a distance offset of 1 m and an angle perpendicular to the surface of the tank. A top-down section of the portion of the tank considered is shown in Fig. 4.5. Similar to the evaluation of coverage used for parameter adjustment, the same tank model is voxelized and the same parameters can be used to evaluate the effectiveness of the coverage of these methods.

Fig. 4.6 shows the best version of each generation method and Fig. 4.7 shows a histogram chart with the views per voxel overlaid with the ideal gamma based distribution generated in part IV.C. The Boustrophedon pattern was generated using two interpolation points between sections longer than 1 m, with a lateral overlap of 0.1 m. The Voronoi partition is generated with 20 points including wall avoidance. Map-TIDAL used $r_{ex} = 1.7$ m, $\theta_{ex} = 0.25$ rad, and $t = 75\%$.

From the histogram in Fig. 4.7, it can be seen that Map-TIDAL achieves 3.81% uninspected voxels with an average of 2.56 views per voxel in 20 tasks as compared to the Voronoi (17.32% uninspected, 1.39 average views, 20 tasks) and boustrophedon (8.94% uninspected, 1.80 average views, 30 tasks). The average of absolute differences between Map-TIDAL and the ideal is 0.029, while the Voronoi (0.100) and boustrophedon (0.081) display larger differences.

4.2.5 Additional Experimental Verification

The third test was conducted in a fresh water pool with no additional parameter tuning, as the pool visibility was similar to that in the OSB test tank. This test involved more movement of the robot, with a surface not directly perpendicular to the robot, similar to

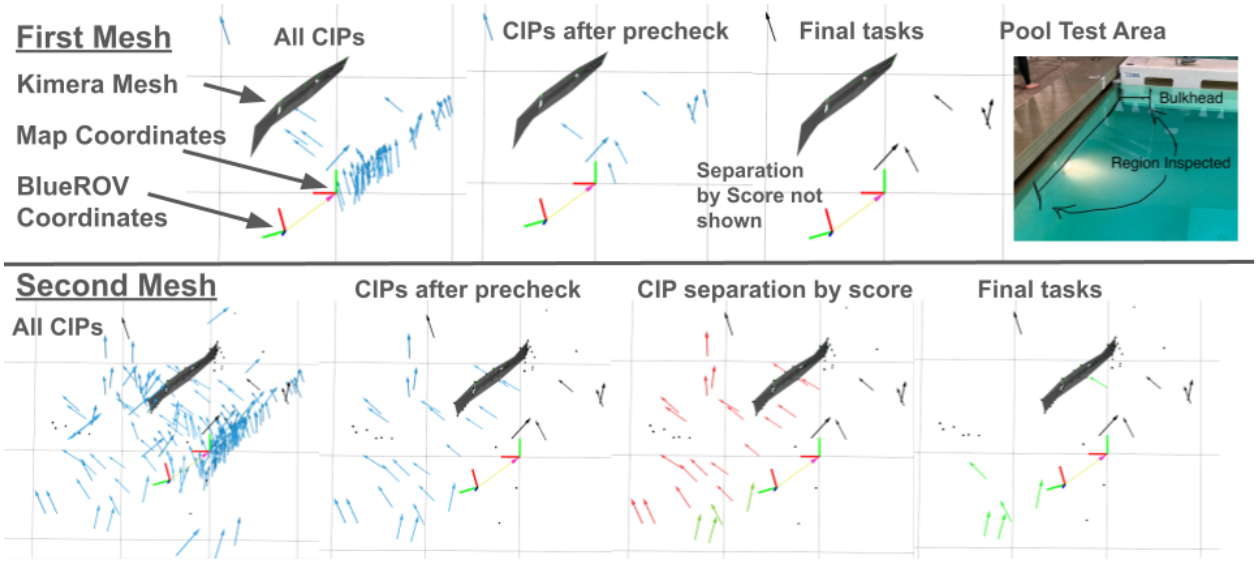
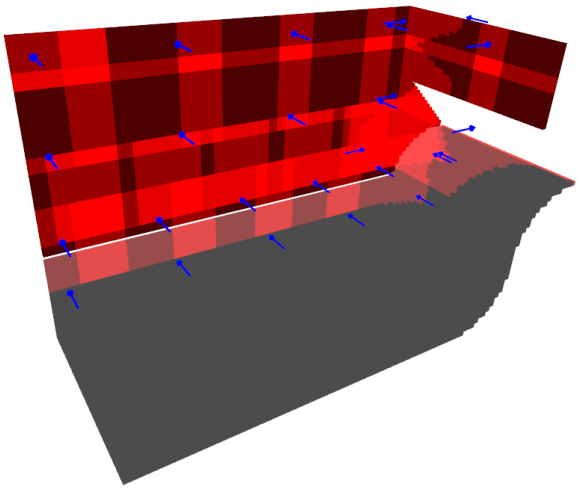


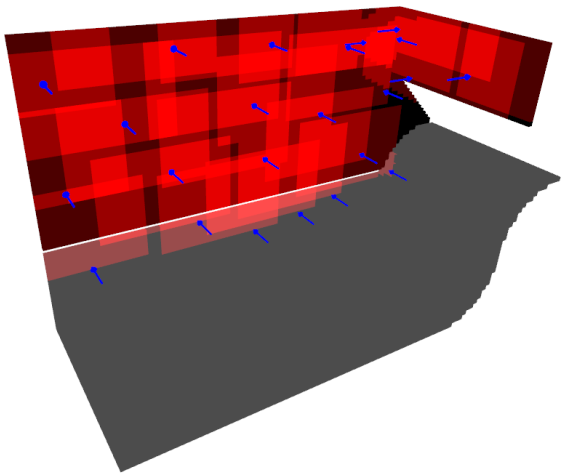
Figure 4.8: Task discovery with steps shown over the first two mesh frames in pool testing. CIPs are colored blue initially. In the case of the first mesh, the final tasks are shown in black, which remain when they are used in the second mesh as the existing tasks. For the second mesh, during the score separation step, ones that are selected are green, while the ones subject to final pruning are red, and the final selection is shown in light green. These light green tasks are then combined with the existing list of tasks for the next mesh frame.

what it would see if moving along a hull at an angle.

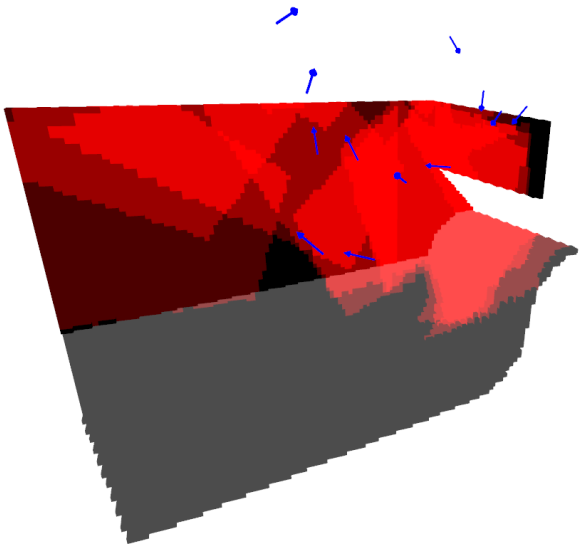
Figure 4.8 shows the results of the first and second mesh frames in one instance of pool based testing. Based on observations made during testing, this set of tasks appears well distributed over the environment, though it can be noted that the Kimera mesh visualization is not complete. The mesh includes all triangles in the mesh while the visualization includes only ones over a certain size. It can also be noted in later mesh frames (not shown) that the reflection of the wall onto the surface of the water causes additional mesh irregularities, indicating that additional visual processing should be developed to handle regions near the surface. Overall, this third test was very successful, demonstrating the usefulness of the Map-TIDAL algorithm in different environments with varying geometric complexities.



(a) Sweeping Pattern: 28 Tasks



(b) Voronoi: 24 Tasks



(c) Map-TIDAL: 12 Tasks

Figure 4.9: Simulated coverage analysis of a pool section for three methods with task locations shown in blue and voxels colored brighter red with each time they are viewed.

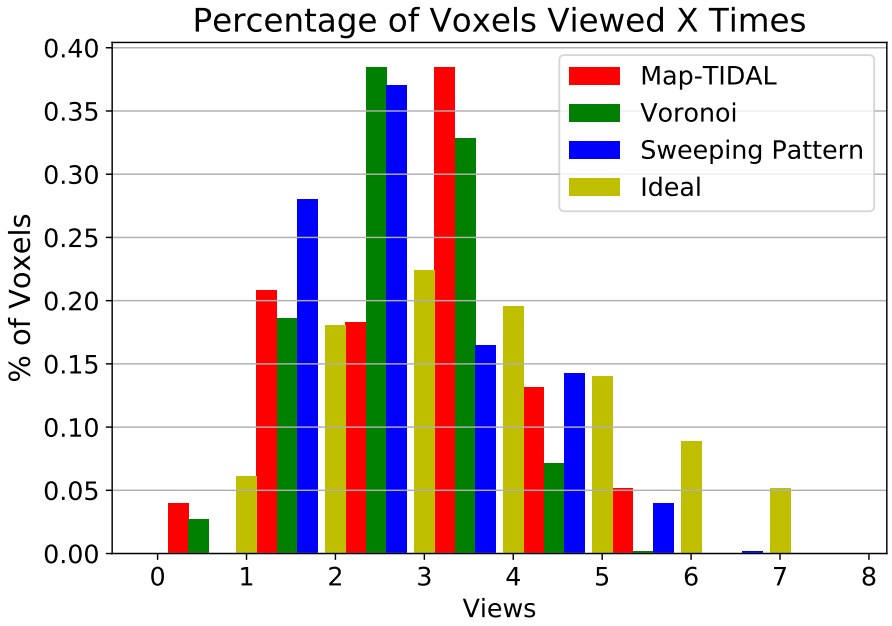


Figure 4.10: Histogram showing normalized views per voxel for different task generation methods compared with the discretized ideal gamma distribution on a pool section.

Similar analysis was done to that in section 4.2.4, the results of which are shown in figures 4.9 and 4.10. In this example, a larger section of the pool is modeled than was analyzed, to enable each method to show the ranges of the selected task positions that will force additional exploration on the robotic system. The Map-TIDAL results were generated using the same parameters as used previously in the test tank. The sweeping patterns were generated on a section of the pool wall up to 2 m deep and the bulkhead, with the space corresponding to the open section under the bulkhead and the sloped region of the bottom set as no-fly zones. To simulate the same shape using the Voronoi generator, any points generated in the same blocked region were discarded. Similar to with the test tank, Map-TIDAL shows more of a clustering of tasks near potential points of interest, in this case the light fixture (on the wall about a meter off of the bulkhead) and the corner where the bulkhead and the wall meet. Map-TIDAL also shows better extension into the areas with gray voxels that represent the

pool space outside of the area already covered in the inspection. When looking at the results shown in figure 4.10, this space is much larger than that shown in the test tank trials, and the ideal distribution has been adjusted to have the same shape value with a higher peak value to allow for this larger coverage region. Map-TIDAL has an average absolute difference from ideal of 0.081, while the Voronoi (0.108) and boustrophedon (0.095) display larger differences once again. Though the benefit is not as high in this case as it was in the test tank case, it is still present, even with half the number of tasks used in the other methods.

4.3 Discussion

Qualitative analysis of the areas with the brightest reds in the task discovery method from Fig. 4.6B, shows clear concentration of views on the center and upper right side of the tank. This corresponds with the position of the safety ladder during the testing, so it is expected that that region should be viewed from more angles due to the concentration of keypoints in that area. Both the sweeping and Voronoi patterns lack this concentration in the provided examples, but with modifications to weighting parameters both methods have the potential to incorporate weighted distributions to task generation.

The main limitation of the Voronoi and sweeping patterns for the underwater inspection case is that they require a model before they can generate their points. The sweeping pattern tool is able to recalculate quickly on the fly when a user adjusts the model of the space, but still requires user input, preventing the system from being fully autonomous. The Voronoi patterns also calculate quickly, but require the geometric limits of the space before starting. Changing the environment for either set would overwrite the existing tasks with a new set, which would be challenging, since underwater communication is slow and unreliable. This limits the effectiveness of both methods in dynamic environments with no prior knowledge of the geometry.

The proposed Map-TIDAL method is not without limitations. Since task discovery is based on the meshes generated during SLAM, it does not require prior knowledge of the geometry to be inspected. However, the quality of the generated tasks will depend heavily on the quality of the SLAM meshes. In murky water, additional sensors, such as the DVL, are necessary to maintain localization when no objects are visible, since any errors in localization result in corresponding errors in task positions. In addition, it is possible that the generated tasks may be placed in areas the robots cannot access due to the fact that only the map viewed by the individual robot is available during task discovery.

4.4 Conclusions

In this paper, we present a new algorithm, Map-TIDAL, for online task discovery from a SLAM generated environment mesh through candidate inspection points generation and pruning based on their keypoint scores and proximity to other tasks. These tasks can then be efficiently distributed to a multi-robot system through a decentralized auctioning and conflict resolution process. As new geometry is discovered (explored), the corresponding tasks are distributed, avoiding the need to maintain models of the inspected structures and focusing the inspection on areas with large numbers of visually distinct features. With suitable parameter adjustment and accurate SLAM, Map-TIDAL produces task locations that provide excellent coverage with appropriately low levels of overinspection.

Future works will include investigating collision avoidance during SLAM-based task discovery and allocation. Another potential direction is multi-modal sensor fusion, where fusion of camera images with side scan sonar data could augment localization and map generation abilities. Side scan sonar data would also provide more accurate meshes for Map-TIDAL's task discovery process. Although Map-TIDAL currently focuses on visual inspection data, the algorithm could be extended for other inspection sensors, such as ultrasonic thickness testers, or for the use of additional tools, such as brushes, scrapers, and probes. Map-TIDAL is designed for underwater use, but has potential for use in challenging aerial environments with limited communications and visual range if adapted for a system of flying drones.

Chapter 5

ANALYSIS OF MOTION PLANNING IN UNDERWATER CONTEXTS

This chapter focuses on the understanding and analysis of selected motion planning tools from the suite of tools provided through the OMPL [44]. A total of 6 methods from both the control-based and geometric-based tools are explained, tested, and analyzed in two different underwater environments: the tank and the pool discussed in 4. When considering how these methods would be able to integrate within the Map-TIDAL process, factors such as memory use, computation time, and accuracy of final location are all important metrics. Additional factors such as the path complexity and the ability to work with dynamic environments and information from other robots will also come into play with the choice of methods and their overall effectiveness in underwater planning. From the controls-based set of tools, RRT and KPIECE1 are chosen for analysis, likewise the geometric version of KPIECE1 is also analyzed for comparison of controls and geometric versions of the methods, with RRTConnect, RRT*, and SPARS as the remaining geometric methods to be reviewed.

5.1 Summary of Methods

5.1.1 RRT

Rapidly-Exploring Random Trees (or RRT) and derivations of it have become a backbone of robotic motion planning techniques. As introduced briefly in chapter 2, RRT is based on building a tree by randomly selecting configurations, finding nearest neighbors within the configuration space, and adding them to the tree if they meet appropriate conditions. In the most basic form of the algorithm from [46], given an initial condition, x_{init} and a state transition equation that provides a new state based on dynamics and constraints, a tree $\mathcal{T}$

with $\mathcal{K}$ vertices can be generated iteratively until the desired location is reached. In each iteration of the tree, a random state within the free configuration space is generated and compared to the existing vertices in the tree to find the nearest neighbor according to a distance metric corresponding to the state space in use. Then inputs are generated to move the tree toward the random state based on the timestep, Δt, being used to generate the tree. The new states are checked for collision to ensure that they remain in the free space, and the new state that is closest to the randomly chosen state is added to the tree as a vertex, and the corresponding control input is added as the edge between those two states.

When considering applications in the underwater field, the notable benefits of RRT are the computational simplicity, the applicability to partially controllable systems, and the bias toward exploration of unseen areas. Paths that tend towards unseen areas are ideal for the task detection in Map-TIDAL. Computational speed and low memory use are both of high importance when path planning is running alongside complex SLAM and task detection with limited onboard computational power. In addition, RRT has been shown to function for partially controllable agents, making it a natural choice for lower DOF underwater robots. Limitations on the original method include that there is no optimization of the path, and convergence rates can be low. Reliance on an accurate distance measure is also a challenge, as will be discussed further when selecting appropriate state spaces.

RRT-Connect

RRT-Connect [47] has two main differences from the original algorithm. First, rather than only extending the tree incrementally at each step, it attempts to extend the tree further until either the goal position or an obstacle or boundary is reached. Second, it features the growth of two trees, one from the initial position, the other from the goal position, and attempts to connect these two trees to find the solution.

The big limitations on this method are that it not intended to work with differential constraints, and needs an inverse kinematic solver for the second tree to get full control commands, or to be solved as a purely geometric problem without the inverse kinematic

considerations (as is used in the OMPL geometric implemenation). However, due to the longer sections and multiple trees, paths are often more direct and efficient, even without an explicit optimization method. RRT-Connect is also applicable to largely open spaces, such as on the exterior of hulls, because the extended edge lengths allow more space to be covered in fewer steps. Not only is that good for reducing the need to increment many times in such open spaces, but it also leads to a path with fewer arbitrary sharp changes in direction. Underwater robots often struggle with sharp turns, higher DOF UUVs due to the wobbling generated in abrupt changes, while the torpedo shaped AUVs are often strictly limited in their maximum turn radius.

Optimal RRT

Optimal RRT [48], or RRT*, is a modification of the original RRT to include optimization as the tree is built. When a new vertex, x_{new} is added to the tree in RRT*, a cost function (typically distance) is used to compare the path length along the tree to the path length if x_{new} is connected directly to previous nearby vertices. If this new connection is shorter than the previous connections and collision free, the two vertices are connected with a new edge, and the edges and vertices that were a longer route are removed. This ensures that when the solution is found, it is asymptotically optimal.

The biggest benefit of RRT* is the optimality condition, which ensures that the paths generated will not be significantly larger than they need to be. Like the RRT-Connect algorithm, RRT* also provides paths with fewer unnecessary turns than the RRT method, in this case due to the tree simplification step rather than the extension step.

5.1.2 KPIECE

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) [49] is a single-query motion planner designed for systems with complex dynamics. It works by discretizing the environment on several levels, so that each level becomes more fine, and this discretization can be used to find areas that have been explored less. These discretized cells

are considered exterior if they have less than 2n instantiated non-diagonal neighbors (where n is the dimension of the discretized space), and interior if they have 2n instantiated neighbor cells. A tree of motions is built by sampling motions that expand into different cells with a bias toward exterior cells. Once a cell is chosen, the controls to get to that cell are sampled, and the progress is computed based on increased coverage of the cells. The motions used in KPIECE are generated with only forward time motions considered, meaning no inverse kinematics are needed, and they can be done based on simulated robot physics.

KPIECE offers a flexible framework with the ability to run in multiple levels of discretization, multiple parallel threads in cases with multi-core machines, and the ability to change the progress metric and add a goal bias to suit the use case. It has speed-up benefits shown against several other planners, including the basic RRT algorithm. It frequently provides paths that are not optimal in terms of distance, but offer extensive exploration in the environment which could be a benefit to inspection cases and task generation. The OMPL implementation used in this evaluation is KPIECE1, which uses one level of discretization, a grid.

5.1.3 SPARS

The SPARS [45] method builds two graphs in parallel, the sparse graph S and the dense graph D. S is built of an adaptively selected subset of nodes from D, while D is built of samples from the free configuration space. Sparse edge lengths are limited by $2 * \Delta$ (the visibility range is Δ), and by using a maximum neighborhood radius of δ in the dense graph. A node is the representative of a sample if they can be connected with an obstacle free path shorter than Δ and the node is the closest node that can do this. Similarly, the visibility region for a node is the set of all collision-free configurations that share the node as their representative. "The interface between two nodes ... is the shared boundary of their visibility regions" [45]. A sample from the dense graph supports the interface of its representative spanner node if there is another dense graph sample with a different representative so that the local path between the samples is an edge of the dense graph. From these definitions, the midpoint

between two spanner nodes is on the interface if the local path is obstacle-free. A sample q is added to the spanner (S) if: (in order)

1. If no existing spanner nodes connect to q it is added as a "guard" for coverage purposes.

2. If it connects two nodes that aren't connected, an edge is added between those nodes. If adding an edge is not possible, q is added as a "bridge" node and connected to both nodes.

3. If q is on an interface, and q and each other sample (q') do not share an edge, they are candidates for addition, then SPARS attempts to add them as "pair":

   (a) If possible, an edge is added between the representatives of q and q' (v and v' respectively). If not:

   (b) If the local path between v and v' is not collision free, SPARS checks if they can be connected through the midpoint between q and q', the midpoint is added and interfaces are reduced if possible. If not:

   (c) Both q and q' are added to S and connected appropriately, then interfaces are reduced if possible.

   If q is still not added, considering all neighbors v'' of v that are not linked to v', it attempts to as as a "path":

   (a) Find the longest path on S between the midpoint $m(v, v')$ and the midpoint of each $m(v, v'')$.

   (b) Find the shortest path between q and all samples that support the interface between v and v'' that have v as their representative in D.

   (c) If the length of that shortest path is shorter than t (the spanner stretch factor) times the length of the longest path, consider all samples along the shortest path for addition and reduce interfaces if possible.

4. If q is not added after all of this, increment the failures count.

Because of the communications constraints between robots underwater, it is hard to communicate large detailed maps between underwater robots. In [45], the authors present trials showing many orders of magnitude smaller roadmaps than other roadmap based methods. While the maps are still too dense to be transferred fully between robots given the communication constraints (their largest map for the many holes trial would be about 27.8kB, which would take just over an hour to transfer), however it seems feasible that the task positions themselves may be able to act as additional sparse nodes for areas, and with the estimate they give of 12 bytes per edge (about a second and a half to transfer at 60 bits/s), it could be possible to update maps using the existing shared task list with new connections as they are discovered by the robots, especially if additional data compression is possible.

The second feature of SPARS that appears applicable to underwater inspections is the idea of visibility embedded into the graphs used. With one of the hardest parts of underwater inspections being the limited visibility, having a dense graph built only for areas that are currently visible, then saving only the minimum of what is needed for the sparse graph outside of that seems like an efficient way to manage the challenges of dynamic environments, while accounting for the very low visibility underwater. This also feeds the idea of using tasks discovered by other robots as potential nodes in the sparse graph, since tasks from one robot are likely to be outside of the visual range of the other robots in the system, but still represent knowledge of the overall environment.

5.2 Adaptations for the Underwater Environment

5.2.1 State Space Setup

OMPL greatly simplifies the implementation and evaluation of the chosen path planning algorithms, but some changes are still required to work specifically for the underwater environment. OMPL defines several state spaces that work with their implementations of the algorithms, and for the most part, only small adjustments are needed. Of the spaces avail-

able, the most applicable for the BlueROV 2 is the SE3 state space, with three variables for position and a quaternion for orientation. The distance metric in this space is a simple Euclidian distance for the position and an arc length for the orientation. For the geometric planners this is almost enough, but when using meters as the units, the arc length calculation quickly overbalances the position based distance, so the weighting of the arc length was lowered.

For the control based planners, velocity is also a part of the state, and an additional compound state was used. The compound state was made of a standard SE3 space with the adjusted weights for the distance metric, alongside a 6-element vector for the velocity. While it is possible to consider the desired velocity as part of the goal, to keep things equivalent between the methods, the weighting for the velocity vector was chosen to be zero, so that the velocity was only used as a part of the motion calculations for the controls. The control space in this case has 4-DOF, x, y, z and the yaw angle, in keeping with the standard BlueROV 2 configuration.

Testing for the motion planning models' applicability is done for the BlueROV 2 here, but other state spaces of interest included in the OMPL library include the Owen [63] and Vana [64] state spaces. These spaces produce gently looping flight paths for aerial vehicles and AUVs, which require a minimum forward velocity and have limited turning ability. These spaces would be ideal for torpedo shaped vehicles, and would be used in future implementation involving that kind of vehicle.

5.2.2 Motion Model and Visualizations

While the geometric based planners require only the geometry of the space and the robot to complete their motion planning, the controls based planners also require a motion model to generate the correct control responses for the robot. While the robot itself is only controllable in 4-DOF, it still moves in a 6-DOF manner. While the OMPL planning algorithms do not take into considerations such as waves, tides, and currents, the same motion model used in the simulations from chapter 3 can be applied here, with the external forces (waves, tides,

and currents) set to zero. Equation 5.1 shows the motion model from [52], where η is the generalized position and orientation in the NED coordinate system, ν is the generalized velocity, τ is the external forces (set to zeros here), M is the inertia matrix with both rigid-body and added mass, $C(\nu)$ is the Coriolis and centripetal matrix, $D(\nu)$ is the hydrodynamic drag matrix, and $g(\eta)$ is the hydrostatic forces and moments, all in the standard.

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau \tag{5.1}$$

$$M = \begin{pmatrix} 18.814 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 30.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.113 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1726 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.2429 \end{pmatrix} \tag{5.2}$$

$$D(\nu) = \begin{pmatrix} -8.814 - 21.42\dot{\nu_0} & 0 & 0 & 0 & 0 & 0 \\ 0 & -20.1 - 28.41\dot{\nu_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & -20.1 - 52.13\dot{\nu_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -19.24\dot{\nu_3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0604 - 22.17\dot{\nu_4} & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.0279 - 14.98\dot{\nu_5} \end{pmatrix} \tag{5.3}$$

$$C(\nu) = \begin{pmatrix} 0 & 0 & 0 & 0 & 30.1\dot{\nu_2} & -30.1\dot{\nu_1} \\ 0 & 0 & 0 & -30.1\dot{\nu_2} & 0 & 18.81\dot{\nu_0} \\ 0 & 0 & 0 & 30.1\dot{\nu_1} & -18.81\dot{\nu_0} & 0 \\ 0 & 30.1\dot{\nu_2} & -30.1\dot{\nu_1} & 0 & 0.2707\dot{\nu_5} & -0.1121\dot{\nu_4} \\ -30.1\dot{\nu_2} & 0 & 18.81\dot{\nu_0} & -0.2707\dot{\nu_5} & 0 & 0.113\dot{\nu_3} \\ 30.1\dot{\nu_1} & -18.81\dot{\nu_0} & 0 & 0.1121\dot{\nu_4} & -0.133\dot{\nu_3} & 0 \end{pmatrix} \tag{5.4}$$

$$g(\nu) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 29.43\cos\eta_4 \sin\eta_3 \\ 29.43\sin\eta_4 \\ 0 \end{pmatrix} \tag{5.5}$$

The final step of adapting the motion model to the OMPL tools is to perform a coordinate transform to match ENU convention used in OMPL. This is necessary for the visualization within the application provided within OMPL, and to work with the provided state space models, which are also in ENU convention. As OMPL uses a world fixed coordinate system for the overall motion planning, the CAD model for the pool was transformed so that the initial position of the robot from the previous analysis of the Map-TIDAL algorithms is the origin of the model. The tank model has its origin on the floor in the center of the tank.

5.3 Analysis and Results

5.3.1 Quantitative

The path planning methods are compared in several different sets of trials. Six trials are done in the pool environment, and three in the tank environment. In the pool, two different starting points, the origin start 1: (0.0, 0.0, 0.0) and randomly chosen start 2: (1.30, -0.73, 1.98), and three different end points, simple goal 1: (1, -1, -1), behind the bulkhead goal 2: (1.5, 0, 2), and randomly chosen goal 3: (-0.46, -2.38, -1.02), are used in each combination. In the tank environment, the robot starts at (2, 3.5, 0) and has three goals: randomly chosen goal 1: (2.25, 2.6, 0.51), the randomly chosen point with an quaternion change to goal 2: (0, 0, 0.1041662, 0.9945599), and the first task location generated by Map-TIDAL goal 3: (1.64, 1.62, -0.049) with rotation (0, 0, 0.1041662, 0.9945599).

Each trial was run 10 times with each planner. Each planner was given a time limit of 100 seconds, a memory limit of 10000 MB, and a goal threshold of 0.01 m. Interestingly enough, every planner used about the same amount of memory, between 350 and 357 MB per run, regardless of the planner chosen. Computation times fell into three categories for the purposes of application to underwater robotics, planners that used the maximum amount of time given: RRT*, control RRT, and control KPIECE1; planners that were complete in under 0.02 seconds: RRT-Connect and geometric KPIECE1; and SPARS, which processed in a wide variety of times in the 10-100 second range, averaging around 28 seconds over all the trials. It was also noted that in visualization trials, the planners that use the full time given were able to produce similar quality of results given only 10 seconds for the geometric based planners, and 30 seconds for the control based planners.

The resulting average distances from the goal are presented in table 5.1, for the control based planners. All the geometric planners were able to find an exact solution each time, indicating that they produce more exact results. The lack of consideration of the actual control efforts may mean that the paths generated by geometric planners may not be able to be followed accurately enough. The control version of KPIECE1 got slightly closer to the

Table 5.1: Average Distance Between Final Location and Goal (m)

	Control RRT	Control KPIECE1
Pool: Start 1, Goal 1	0.1040429	0.09628323
Pool: Start 1, Goal 2	0.2353666	0.16370444
Pool: Start 1, Goal 3	0.10562171	0.1199825
Pool: Start 2, Goal 1	0.2958421	0.2253596
Pool: Start 2, Goal 2	0.338674	0.29338891
Pool: Start 2, Goal 3	0.0961419	0.08786319
Tank: Goal 1	0.4701992	0.41762703
Tank: Goal 2	0.4963848	0.42859
Tank: Goal 3	0.53696065	0.3639301

final goal location on average than the RRT in most trials, but the results of both methods were in an acceptable range.

The next point of comparison is the overall path length for each planner. Of the paths found by the control planners, RRT almost always showed a shorter path than KPIECE1, a trend also seen in the geometric versions of the planners. As expected, RRT showed much longer paths than the more optimized RRT-Connect and RRT*, with RRT* usually having the shortest, or tied for shortest, path lengths.

Each of the geometric planners also featured a simplification to optimize for near minimal path length performed on the final paths chosen in each instance. While the paths found before simplification in each run were different, it is interesting to note that regardless of planner, the simplified solution for every run of every trial except the pool trial converged to the same value, the Euclidean distance. This is likely because the environments of both the pool and the tank are very open, reflecting the openness of large interior tanks and on the exterior of hulls. The exception is the goal location that was set behind the pool bulkhead,

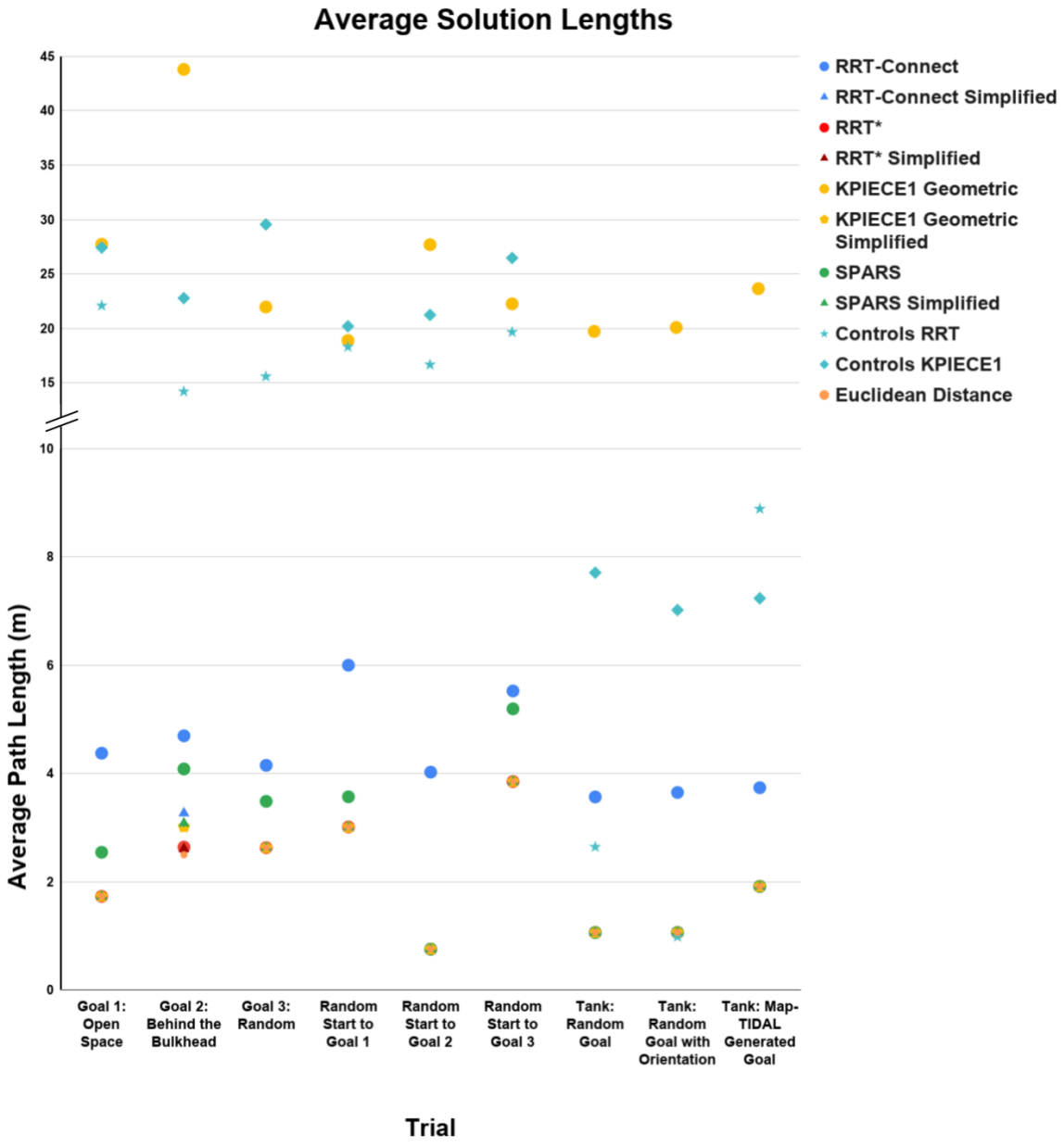


Figure 5.1: Graph comparing the average path lengths of the various planners for each set of trials.

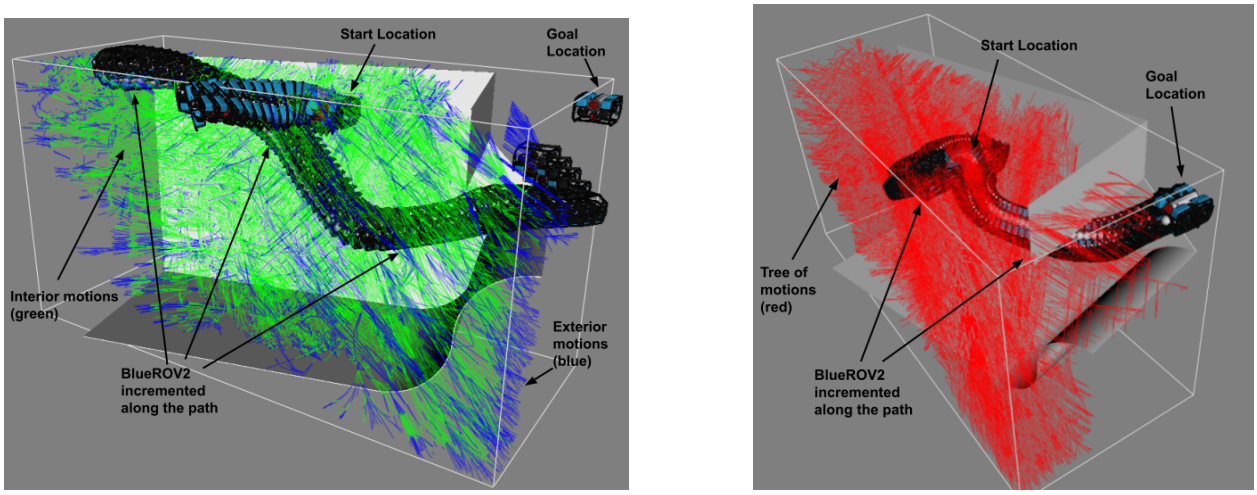
for which following the path along the shortest distance would result in collision. In that trial, the planner with the shortest path length was the simplified RRT* (the simplified version was only 0.03 m shorter, so it is hard to see on the graph).

Another interesting observation from the path lengths before simplification is that the KPIECE1 methods had significantly longer path lengths than any other method, and the variation in the path lengths within the same trials were much higher, often 20-60 meters different between runs, while the other geometric planners saw paths within 0-3 m of each other. Even though the path length averages for RRT are in the same average length to the KPIECE1 methods, RRT generally had only about 10 m of variation in path length. This again shows the tendency of KPIECE1 to wander and explore the region, while the other planners provide more direct paths.

5.3.2 Qualitative

In addition to the performance statistics presented in the previous section, several aspects of the motion planners are better observed as more abstract qualities of the results. Here, the planners are evaluated in terms of smoothness, exploration, excess angular displacement, and mapping ability. Smoothness in this case refers to how sharp the turns in the path are, since sharp turns are challenging for underwater robots. Paths that make turns more gradually are preferred. The balance of exploration to discover more tasks at the edges of the explored regions and the efficiency of direct paths where possible is important, and moderate amounts of exploration can be good, but huge, unnecessary variations in path will lead to excessive time and energy spent when it is not needed. While the system will eventually feature robots without tethers, it is possible there could be tethered robots still in the system, leading to the third quality being evaluated- excessive angular displacement. For tethered robots, spinning will quickly result in twisted, tangled and possibly damaged tethers, while for untethered robots, spinning is more of a localization problem as large angular changes will likely result in a loss of visual contact with the vessel being inspected, forcing relocalization in the visual SLAM module when the vessel is in view again.

Figures 5.2 and 5.3 show samples from each planner, visualized using the OMPL app. The second pool trial, featuring the move under the bulkhead, was chosen to visualize as it was the one with the most numeric differences between the methods, as well as the most interesting motions. As expected from the quantitative review, the control based planners feature significantly longer and loopier paths, and are not exact in reaching the goal location. Both control based methods have expansive trees with many motions in them, unlike the geometric methods with much more sparse trees, largely due to the finer timestep needed for forward propagation within the motion model. For the geometric planners, the trees shown are the full results, while the BlueROV 2 is visualized on the simplified path.

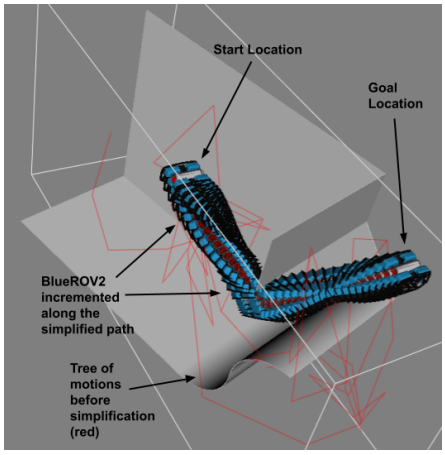


(a) Control based KPIECE1

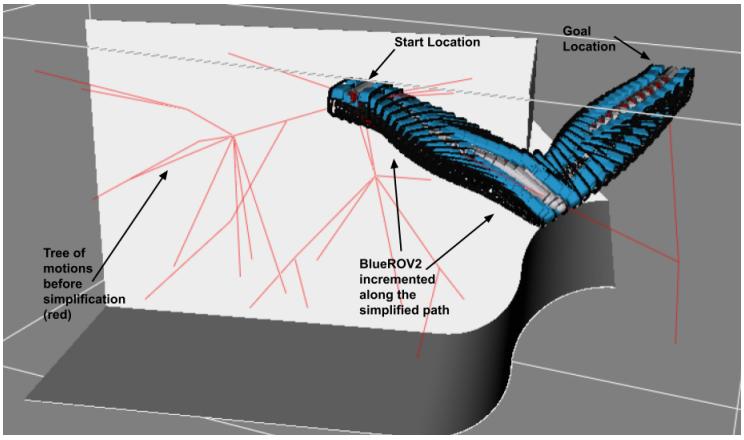
(b) Control based RRT

Figure 5.2: Sample of paths generated by each planner.

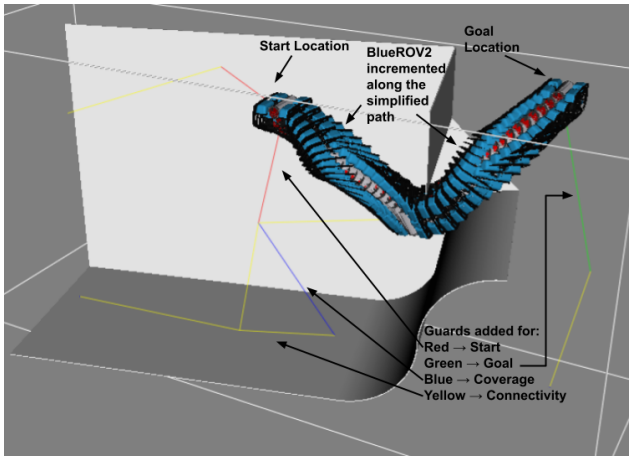
The smoothness of the control paths is generally much better than that of the geometric based planners, especially looking at the part of the path that goes under the bulkhead. For each of the geometric paths, it drops down, then sharply back up in a V shape. The control based planners make this transition more smoothly, in a curve rather than a sharp V, though they do have some sharper turns in the parts of their paths that loop, and the control based



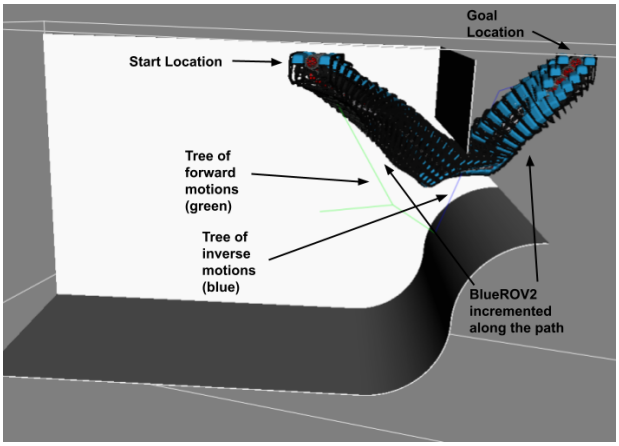
(a) Geometric KPIECE1



(b) RRT*



(c) SPARS



(d) RRT-Connect

Figure 5.3: Sample of paths generated by each planner.

KPIECE1 is generally more abrupt in direction changes than the control based RRT.

When looking at how each planner explores the region, it is clear that the simplified paths have significantly less exploration than the paths before simplification, or the paths of the control based methods. Looking at how the trees are formed, it is clear the RRT-Connect has the most direct tree, with only one offshoot, while the control based methods have much

denser trees. The high density in the control based trees comes from needing to have small propagation step sizes for control considerations, and the lack of tree thinning involved in the control based methods. Comparing the trees from geometric KPIECE1 and the RRT* show that the KPIECE1 tree forms more long zig-zag paths, while the RRT* retains a more typical branching tree behavior. All the methods except the RRT-Connect have parts of their tree or graph extending to fill the limits of the space allowed in the simulation. There are two extremes here: in general the control based planners do too much exploration with their extra loops, while the simplified solutions found in the geometric planner are all very direct with little to no exploration.

In terms of excessive angular displacement, or spinning, in each planner, it is important to note that despite only having 4DOF control, the requirement of keeping the full 6DOF motion model means that the paths still include pitch and roll elements, even if the controls would not be capable of producing them. In the case of the geometric models, the controls are not considered, and thus these models also feature pitch and roll aspects that would need to be accounted for in the collision checking when generating the controls for those paths. Considering only yaw spinning, none of the geometric based models complete a full spin, though the KPIECE1 planner comes close, then reverses the spin before arriving at the goal location, and despite the additional looping, the same is true of the control based KPIECE1 and RRT.

The final quality, mapping ability is only present in one of the planners- SPARS. While all the other methods tested here are one-shot methods, SPARS maintains a graph based map of the environment. Because of this, it is easier to compare between the other methods, since they function very similarly. However, the mapping ability of SPARS puts it in the unique position of being the method most able to integrate with the long term list of planning objectives generated by Map-TIDAL, and make use of them for more direct cooperative planning. While the other planners are capable of planning faster than SPARS on single shot trials, SPARS had to build the map fresh each trial, which added significant time- but note that the extra time is still less than the expected time burden of underwater communications

for the Map-TIDAL conflict resolution phases. It is expected that mapping will take less time as time goes on, and the map is made more complete, and that the map will then be limited in size, the other concern with final map. The quality of the graph based maps generated by SPARS shows good coverage of both pool and tank environments.

5.4 Conclusions

From this introductory survey of how motion planners may be applied in the underwater environment, there are three main questions with how the planners performed: to use control or geometric based planners, which method performed best overall, and what method or methods are worth investigating and adapting to work with Map-TIDAL.

The question of whether to use a control based planner or a geometric planner comes down largely to optimization. The control based methods available through OMPL did not feature any path or tree optimizations, while all of the geometric planners were optimized. While some meandering and smooth paths can be a good thing, both controls based planners had significantly longer paths than necessary, to the point where the euclidean distance estimate used for cost in the auction phase of Map-TIDAL would be completely inaccurate. While the final paths are not expected to match the Euclidean distance, in order to account for obstacle avoidance, they shouldn't be as far off as the paths generated by the control based planners are. However, as the main feature responsible for this difference is the optimization, it would be prudent to either try to apply controls considerations to the planners that already have optimization, or to apply optimization strategies to the control based planners. With the planning time as the next biggest concern for the planners, it can be noted that both control based planners took the full allowed time, while several geometric based planners (SPARS, RRT-Connect, and KPIECE1) did not. This, as well as potential options to consider controls constraints without a full motion model, make it likely that a geometric based planner with additional constraints will be best moving forward.

To determine the best planner, consider evaluations of time and solution quality. Of the geometric planners, RRT-Connect, the geometric based KPIECE1, and SPARS all produced

results in less time than allowed. While RRT* used the full time in the multi-run trials, when run individually during the visualization tests, it also performed complete runs with valid solutions in under 0.02 seconds, showing that it is also capable of producing quick results. Of these, RRT* and RRT-connect displayed more branching and less wandering paths. RRT-Connect and KPIECE1 perform path optimizations only after a connected path is found. However SPARS and RRT* perform optimizations as they run, RRT* in checking for a shorter direct connection to the start and SPARS in pruning the map to remove unnecessary nodes. While both SPARS and RRT* also perform optimizations after a path has been found, these optimizations result in smaller changes than RRT-Connect and KPIECE1 required. The time advantage of RRT* over SPARS leads to the conclusion that RRT* is the best planner of those evaluated, with SPARS being worth further investigation for both optimality and the potential benefits that come from the graph based map.

Chapter 6

CONCLUSIONS

6.1 Contributions

This dissertation has presented the contributions of several years of research into underwater multi-robot systems for inspection and maintenance of naval platforms. By adapting simulations to suit the challenges of slow communication, unique underwater sensors, and appropriate motion models, task allocation algorithms could be tested on multiple platforms. Using the simulation to verify functionality was critical when adapting previous d-CBBA methods to be able to account for tasks added dynamically as the robots discovered them in their environments. Dynamic task allocation is critical in the underwater environment, where the exact geometry of the structures being inspected isn't available to the robots. Due to the harsh environment, the system must be able to adapt to the loss or addition of members, as well as being able to handle a variety of robots with different capabilities. The task allocation algorithm presented here is key to the functionality of cooperative underwater robot inspections.

The Map-TIDAL system is the second major contribution in this work. While developing the task allocation algorithm, one of the biggest challenges was to find a way to determine where the robots in the system should go to perform the best underwater inspections possible. Existing methods were found wanting, as they required access to CAD models of the inspected areas, large amounts of computational power, and/or relied on randomized or equally distributed point generation. Underwater environments are a dual challenge of huge amounts of largely featureless spaces, and small but important areas of interest with complex geometry. Evenly distributed points ensure coverage, but do not offer enough detail to the complex and interesting features. Randomized points within certain distributions can offer

Gaps in Current Technology	Solutions from this Research
Datasets lack customizable sensor options, simulations lack full motion models or appropriate visual rendering to sufficiently model underwater robotics for cooperation algorithm testing.	Combine options from the UUV simulator and Project Dave, and use the Fossen motion model to find appropriate values to model the BlueROV2 in simulation. From 3.1.
Decentralized auctioning mechanisms rely on having a known set of tasks available at the beginning of operation, which is rarely the case in underwater inspections where tasks must be determined as the robots explore.	Adapt the d-CBBA algorithm with a simple cost function and separate bidding mechanism to enable additional tasks to be added during operation. From 3.2.
Task generation for inspections relies on knowledge of the inspection target and environment, both of which are lacking in the underwater inspection use case.	Leverage SLAM and task allocation to enable Map-TIDAL: online task generation based on geometry found during individual SLAM, pruned with knowledge provided from all the robots through the task allocation process. From 4.
Many motion planning algorithms exist for air and ground based robots. Which are applicable to the dynamic and challenging underwater environment?	From planner analysis, RRT* and SPARS show the best results based on data collected from two different underwater environments, making these methods good options to expand the Map-TIDAL process to include motion planning. From 5.
Overall, existing solutions work in isolation or in the air, but cannot handle the complexity and challenges of underwater environments alone.	I have taken these partial solutions and used their strengths to overcome the weaknesses in other areas, combining them in ways that fill the gaps in the abilities of underwater multi-robot systems.

similar levels of coverage, but without prior knowledge of the environment geometry, they will also fail to take into account the areas of interest.

Map-TIDAL leverages the map created by the SLAM module to overcome these issues, as more SLAM features are found, the mesh naturally becomes denser, while the mesh is larger and more sparse in areas with fewer features. By using the geometry of the mesh to determine the inspection points, areas of interest are automatically given more attention, while the completeness of the mesh still maintains the necessary coverage. The combined SLAM, task discovery and task allocation loop also allows the robots to maintain a shared representation of the environment through the map of tasks they share, without burdening limited communication bandwidths with direct sharing of SLAM maps.

With addition of motion planning to the suite of tools developed for underwater robotic inspections, the system will be able to plot collision free courses to safely inspect interior and exterior underwater structures. By evaluating different motion planning tools, their strengths and weaknesses within the underwater inspection tasks can be balanced to find the optimal methods. Based on the analysis of available motion planning tools, RRT* and SPARS are the most applicable to adapt to work with Map-TIDAL.

6.2 Impacts

Underwater inspections of interior and exterior spaces is a huge challenge for naval vessels, and robotic solutions are one of the best options to increase efficiency and overall capacity. The contributions presented here all work towards that end. While multi-robot systems are more challenging than individual robots, the benefits to robust operation and overall capacity that come from multi-robot systems is worth the effort required to make them functional. While work towards parts of the solution can be found in a wide variety of research and industrial tools, most of these tools are made to work in isolation, without considering the ways they could benefit a whole system or the end needs of the user.

Shipyards' demands on robots lead to frequent breakdowns that render single-robot systems inoperable, a problem that multi-robot systems conquer by their numbers- as long as

some of the system still works, the job gets done. Relieving the strain on the human divers and tank inspectors is another huge benefit of the system this research works towards, saving both lives and time by freeing up the human divers to focus on more specialized maintenance needs. By knowing what needs fixed before bringing a vessel into drydock, repair time can be reduced and more accurately predicted. If the time savings of a single robot can turn a 2 week inspection into a single shift, the savings from using a multi-robot system can be expected to be even higher.

6.3 Future Work

The future of this work has several promising directions to continue the push towards a fully autonomous system. As noted in chapter 5, the analysis of the motion planning tools available shows that implementing RRT* and SPARS as motion planning could benefit the Map-TIDAL system, and fully integrating them is definitely one of the next steps. With motion planning and collision checking, the system will be ready to begin larger scale trials and demonstrations.

In addition to motion planning, there are also plans to integrate data from side scan sonars, and a wider array of test vehicles. The addition of sonar and other sensors will allow more accurate SLAM, better meshes for Map-TIDAL to work with, and an expanded range of inspection capabilities. Using a variety of vehicles will also demonstrate the effectiveness of the algorithms' ability to handle heterogeneous systems. By enhancing the capability of the system to work with whichever robots are available, the system will grow more robust.

One of the largest upcoming challenges for this system to reach the goal of being a fully functional and usable tool will be demonstrating to leadership that the system is safe for use and accurate enough to be relied on. Work towards not just demonstrations of functionality, but also safety and accuracy will be an important next step. Development of good visualizations and user interfaces will will also work towards proving the system to leadership, operators, and inspectors alike. Even in the more technical sides of the research, keeping the goals and needs of the end users in mind is important. No matter how effective

the system, if people do not believe in its usefulness and trust its results, it will not be used. Human factors are just as important as technical advancements.

This research brings together theoretical and experimental tools, integrating and adapting them in a way that makes the whole bigger than the sum of its parts.

BIBLIOGRAPHY

[1] B. Lin and X. Dong, "Ship hull inspection: A survey," *Ocean Eng.*, vol. 289, p. 116281, 2023.

[2] R. Eckholdt Andersen, R. Yding Brogaard, and E. Boukas, "Remote inspection techniques: A review of autonomous robotic inspection for marine vessels," *IEEE Trans. Field Robot.*, vol. 2, pp. 1–20, 2025.

[3] SS521-AG-PRO-010, *US. Navy Diving Manual*, vol. Rev. 6 with Change A. 2011.

[4] "Confined spaces injuries in the maritime industry 2024," Sep 2024.

[5] Field-Robotics-Lab, "Project DAVE." `https://github.com/Field-Robotics-Lab/dave/wiki`.

[6] "BlueROV2." `https://bluerobotics.com/store/rov/bluerov2/`, Feb 2022.

[7] P. Mihelich, J. Leibs, and K. Konolige, "stereo-image-proc."

[8] T. Moore and D. Stouch, "A generalized extended Kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[9] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *J. Field Robot.*, vol. 36, no. 2, pp. 416–446, 2019.

[10] M. Ruediger, T. M. Paine, and A. G. Banerjee, "Simulation of underwater environments to investigate multi-robot systems for marine hull inspection," in *IEEE OCEANS, Hampton Roads*, pp. 1–7, 2022.

[11] P. Georg Olofsson Zwilgmeyer, M. Yip, A. Langeland Teigen, R. Mester, and A. Stahl, "The varos synthetic underwater data set: Towards realistic multi-sensor underwater data with ground truth," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 3715–3723, 2021.

[12] M. Ferrera, V. Creuze, J. Moras, and P. Trouvé-Peloux, "AQUALOC: an underwater dataset for visual-inertial-pressure localization," *CoRR*, vol. abs/1910.14532, 2019.

[13] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A Gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, Sep 2016.

[14] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2577–2582, 2012.

[15] O. Kermorgant, "A dynamic simulator for underwater vehicle-manipulators," vol. 8810, 10 2014.

[16] J. Faigl, P. Váňa, and J. Deckerová, "Fast heuristics for the 3-d multi-goal path planning based on the generalized traveling salesman problem with neighborhoods," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2439–2446, 2019.

[17] J. Clausen, "Branch and bound algorithms-principles and examples," 2003.

[18] K. E. C. Booth, T. T. Tran, G. Nejat, and J. C. Beck, "Mixed-integer and constraint programming techniques for mobile robot task planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 500–507, 2016.

[19] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast scheduling of robot teams performing tasks with temporospatial constraints," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 220–239, 2018.

[20] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 4863–4869, AAAI Press, 2017.

[21] L. Brunet, H.-L. Choi, and J. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, 2009.

[22] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3844–3851, 2019.

[23] V. Tereshchuk, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "A scheduling method for multi-robot assembly of aircraft structures with soft task precedence constraints," *Robot. Comput. Int. Manuf.*, vol. 71, p. 102154, 2021.

[24] J. Li, B. Liu, C. Liu, and C. Lin, "Dynamic task allocation for heterogeneous multi-autonomous underwater vehicle collaboration under mine countermeasures missions," *Journal of Marine Science and Engineering*, vol. 13, no. 3, 2025.

[25] X. Wang, X. Fan, P. Shi, J. Ni, and Z. Zhou, "An overview of key SLAM technologies for underwater scenes," *Remote Sensing*, vol. 15, no. 10, p. 2496, 2023.

[26] M. Heshmat, L. Saad Saoud, M. Abujabal, A. Sultan, M. Elmezain, L. Seneviratne, and I. Hussain, "Underwater slam meets deep learning: Challenges, multi-sensor integration, and future directions," *Sensors*, vol. 25, no. 11, p. 3258, 2025.

[27] F. Hidalgo, C. Kahlefendt, and T. Bräunl, "Monocular ORB-SLAM application in underwater scenarios," in *OCEANS - MTS/IEEE*, pp. 1–4, 2018.

[28] L. M. Hodne, E. Leikvoll, M. Yip, A. L. Teigen, A. Stahl, and R. Mester, "Detecting and suppressing marine snow for underwater visual SLAM," in *IEEE/CVF Conf. Comp. Vis. Pattern Recognit. Workshops*, pp. 5097–5105, 2022.

[29] T. Łuczyński, M. Pfingsthorn, and A. Birk, "The pinax-model for accurate and efficient refraction correction of underwater cameras in flat-pane housings," *Ocean Eng.*, vol. 133, pp. 9–22, 2017.

[30] E. Westman, A. Hinduja, and M. Kaess, "Feature-based SLAM for imaging sonar with under-constrained landmarks," in *IEEE Int. Conf. Robot. Autom.*, pp. 3629–3636, 2018.

[31] J. Wang, F. Chen, Y. Huang, J. McConnell, T. Shan, and B. Englot, "Virtual maps for autonomous exploration of cluttered underwater environments," *IEEE J. Oceanic Eng.*, vol. 47, no. 4, pp. 916–935, 2022.

[32] N. Loi, Y. Z. Tan, E. W. Goh, and M. H. Ang, "Sonar SLAM in structured underwater environments," in *OCEANS - Singapore*, pp. 1–10, 2024.

[33] D. C. Estrada, F. R. Dalgleish, C. J. D. Ouden, B. Ramos, Y. Li, and B. Ouyang, "Underwater LiDAR image enhancement using a GAN based machine learning technique," *IEEE Sensors J.*, vol. 22, no. 5, pp. 4438–4451, 2022.

[34] J. Zhang, F. Han, D. Han, J. Yang, W. Zhao, and H. Li, "Integration of sonar and visual–inertial systems for SLAM in underwater environments," *IEEE Sensors J.*, vol. 24, no. 10, pp. 16792–16804, 2024.

[35] S. Pan, Z. Hong, Z. Hu, X. Xu, W. Lu, and L. Hu, "RUSSO: Robust underwater SLAM with sonar optimization against visual degradation," *IEEE/ASME Trans. Mechatron.*, pp. 1–12, 2025.

[36] S. Ding, T. Zhang, Y. Li, S. Xu, and M. Lei, "Underwater multi-sensor fusion localization with visual-inertial-depth using hybrid residuals and efficient loop closing," *Measurement*, vol. 238, p. 115245, 2024.

[37] S. Xu, K. Zhang, and S. Wang, "AQUA-SLAM: Tightly coupled underwater acoustic-visual-inertial SLAM with sensor calibration," *IEEE Trans. Robot.*, vol. 41, pp. 2785–2803, 2025.

[38] S. Rahman, A. Quattrini Li, and I. Rekleitis, "SVIn2: A multi-sensor fusion-based underwater SLAM system," *Int. J. Robot. Res.*, vol. 41, no. 11-12, pp. 1022–1042, 2022.

[39] S. Xu, J. Scharff Willners, J. Roe, S. Katagiri, T. Luczynski, Y. Petillot, and S. Wang, "Tank dataset: An underwater multi-sensor dataset for SLAM evaluation," *Int. J. Robot. Res.*, p. 02783649251364904, 2025.

[40] Y. Liu, W. Zhao, H. Liu, Y. Wang, and X. Yue, "Coverage path planning for robotic quality inspection with control on measurement uncertainty," *IEEE/ASME Trans. Mechatron.*, vol. 27, no. 5, pp. 3482–3493, 2022.

[41] A. Mavrinac, X. Chen, and J. L. Alarcon-Herrera, "Semiautomatic model-based view planning for active triangulation 3-D inspection systems," *IEEE/ASME Trans. Mechatron.*, vol. 20, no. 2, pp. 799–811, 2015.

[42] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: An algorithmic approach," *Ann. Math. Artif. Intell.*, vol. 52, pp. 109–142, 2008.

[43] A. Yazici, G. Kirlik, O. Parlaktuna, and A. Sipahioglu, "A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints," *IEEE Trans. Cybernet.*, vol. 44, no. 3, pp. 305–314, 2014.

[44] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. `https://ompl.kavrakilab.org`.

[45] A. Dobson, A. Krontiris, and K. E. Bekris, "Sparse roadmap spanners," in *Algorithmic Foundations of Robotics X* (E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, eds.), (Berlin, Heidelberg), pp. 279–296, Springer Berlin Heidelberg, 2013.

[46] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[47] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, pp. 473–479 vol.1, 1999.

[48] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," in *International Journal of Robotics Research*, vol. 30, 2011.

[49] I. Şucan and L. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[50] "ROS Noetic Ninjemys." `http://wiki.ros.org/noetic`, May 2020.

[51] F. Vaz, "Scripts to help BlueROV2 integration with ROS and UUV simulator." `https://github.com/fredvaz/bluerov2`, Jul 2018.

[52] T. I. Fossen, *Guidance and Control of Ocean Vehicles*. John Wiley, 1994.

[53] T. Paine, "Robust model identification methods for nonlinear second-order plant models for underwater vehicles," Master's thesis, Johns Hopkins University, 2018.

[54] D. T. Sen and T. C. Vinh, "Determination of added mass and inertia moment of marine ships moving in 6 degrees of freedom," *International Journal of Transportation Engineering and Technology*, vol. 2, p. 8–14, Mar 2016.

[55] J.-C. Trujillo, R. Munguia, E. Guerra, and A. Grau, "Cooperative monocular-based SLAM for multi-UAV systems in GPS-denied environments," *Sensors*, vol. 18, no. 5, 2018.

[56] A. G. Banerjee, S. Chowdhury, W. Losert, and S. K. Gupta, "Real-time path planning for coordinated transport of multiple particles using optical tweezers," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 4, pp. 669–678, 2012.

[57] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. Robot. Autom.*, pp. 1689–1696, 2020.

[58] Y. Tian, Y. Chang, F. H. Arias, C. Nieto-Granda, J. P. How, and L. Carlone, "Kimera-Multi: Robust, distributed, dense metric-semantic SLAM for multi-robot systems," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2022–2038, 2022.

[59] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone, "Incremental Visual-Inertial 3D Mesh Generation with Structural Regularities," in *IEEE Int. Conf. Robot. Autom.*, pp. 8220–8226, 2019.

[60] M. Zhou, J. Li, C. Wang, J. Wang, and L. Wang, "Applications of Voronoi diagrams in multi-robot coverage: A review," *J. Marine Sci. Eng.*, vol. 12, no. 6: 1022, 2024.

[61] WebSVG, "Voronoi editor." https://github.com/WebSVG/voronoi.

[62] R. Bähnemann, N. Lawrance, J. J. Chung, M. Pantic, R. Siegwart, and J. Nieto, "Revisiting boustrophedon coverage path planning as a generalized travelling salesman problem," *Field Service Robot.*, pp. 277–290, 2021.

[63] T. McLain, R. W. Beard, and M. Owen, "Implementing dubins airplane paths on fixed-wing uavs," *Handbook of Unmanned Aerial Vehicles*, vol. ch. 68, p. 1677–1701, 2014.

[64] P. Váňa, A. Alves Neto, J. Faigl, and D. G. Macharet, "Minimal 3d dubins path with bounded curvature and pitch angle," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8497–8503, 2020.

[65] M. McIntire, E. Nunes, and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Int. Conf. Autonomous Agents & Multiagent Syst.*, pp. 1078–1086, 2016.

[66] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *2009 IEEE International Conference on Robotics and Automation*, pp. 2061–2067, 2009.

[67] A. Shkolnik and R. Tedrake, "High-dimensional underactuated motion planning via task space control," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3762–3768, 2008.

Appendix A

PROOF OF DMG PROPERTY OF TASK ALLOCATION BID FUNCTION

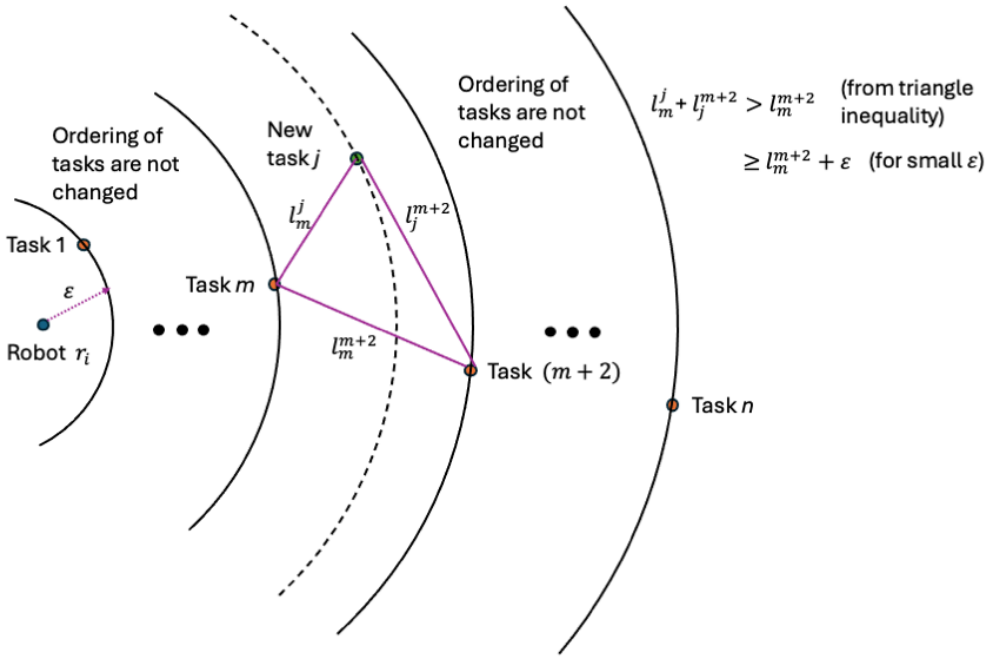


Figure A.1: Schematic illustration of task bidding by a robot.

Lemma 1. *The scoring function in Algorithm 1 satisfies the DMG property for any sufficiently large positive value of the constant task completion reward.*

Proof. We show that the lemma holds individually for any given robot participating in the multi-robot task auctioning and conflict resolution algorithm. Per standard notation, r_i maintains a bundle of tasks $\mathbf{b}_i$. The total value (or score) on adding an arbitrary task j in

the very front (beginning) of the bundle $\mathbf{b}_i$ is computed as

$$S_i^{\mathbf{p}_i \oplus \{j\}} = nR - \left(l_i^j + \sum_{k=2}^{n} l_{k-1}^k \right). \tag{A.1}$$

Here, $\mathbf{p}_i$ is the robot path to sequentially execute all the tasks in $\mathbf{b}_i$; R is the fixed reward of completing any task by any robot; l_i^j is the Euclidean distance (shortest path length) of task j from the initial location of r_i; and $l_{k-1}^k, k = 2, 3, \ldots, n$, is the Euclidean distance of the task at the kth position of $\mathbf{b}_i$ from the task at the $(k-1)$th position. Note here that $S_i^{\mathbf{p}_i}$ is the total score of completing all the other $(n-1)$ tasks in the most optimal manner based on their respective distances from r_i and each other. Accordingly, the *marginal* score improvement of adding task j at the beginning of $\mathbf{b}_i$ is given by

$$\begin{aligned} c_{ij}[\mathbf{b}_i] &= S_i^{\mathbf{p}_i \oplus \{j\}} - S_i^{\mathbf{p}_i} \\ &= R - l_i^j. \end{aligned} \tag{A.2}$$

This is true as the optimal sequence of the other, pre-allocated $(n-1)$ tasks is unchanged when task j is added at the beginning (see Figure A.1 for geometric illustration). Note that we do not use the max operator explicitly for the first term in the LHS of Equation (A.2) (unlike lines 6 and 7 in Algorithm 1), as we assume here that the best position to place j is in the front of $\mathbf{b}_i$.

Now, suppose that $m < n$ tasks are included before task j in $\mathbf{b}_i$ to realize an optimal task execution sequence. The total score then becomes (with $m+1$ as the best place to insert j)

$$\begin{aligned} S_i^{\mathbf{p}_i \oplus_m \{j\}} &= nR - \left(l_i^1 + \sum_{k=2}^{m} l_{k-1}^k + l_m^j \right) && \text{if } m = n-1, \\ &= nR - \left(l_i^1 + \sum_{k=2}^{m} l_{k-1}^k + l_m^j + \sum_{k=m+2}^{n} l_{k-1}^k \right) && \text{otherwise.} \end{aligned} \tag{A.3}$$

The marginal score gain is then given by

$$c_{ij}[\mathbf{b}_i \oplus_m \mathbf{b}] = S_i^{\mathbf{p}_i \oplus_m \{j\}} - S_i^{\mathbf{p}_i},$$

where $\mathbf{b}$ is the bundle before task j is added at the $(m+1)$th position. This always comes out to be

$$c_{ij}[\mathbf{b}_i \oplus_m \mathbf{b}] = R - \left(l_m^j + l_j^{m+2} - l_m^{m+2}\right) \quad m = 1, 2, \ldots, n-1, \tag{A.4}$$

since the optimal ordering of the other $(n-1)$ tasks remains unchanged for both the first and second sequences of m and $(n-m-1)$ tasks, respectively (see Figure A.1 for geometric explanation).

Therefore, the DMG property is true as $c_{ij}[\mathbf{b}_i] \geq c_{ij}[\mathbf{b}_i \oplus_m \mathbf{b}]$ for any admissible value of m, provided R is a sufficiently large positive number so that $c_{ij} \geq 0$. □

Note that we can easily extend this result for the dynamic task allocation scenario, where new tasks are added sequentially for a robot to bid on.