Mission Planning for Field Robots using Symbolic Planning and Topology

By

Jonathan Wheare, BEng (E.&Elec.), MEng (Electronic)

Thesis

Submitted to Flinders University

for the degree of

Doctor of Philosophy

College of Science and Engineering

December 2018

Abstract

Field robotics is an area of research that takes the discipline of robotics from the confines of the laboratory into the unstructured and complex environment of the real world. Planning and guidance systems have been developed to allow field robotic platforms to operate in unstructured environments, but the limited amount of computing resources has constrained the ability of field platforms to dynamically replan their missions. Domain specific planning systems for path planning provide the efficiency that is required to handle large and complex environments, but deliberative higher level mission planning systems typically use a domain independent planner to find a solution to the vehicle's task. As such, mission planners lack understanding of their spatial environment. This thesis chronicles the development of a belief compression method using topological thinning to simplify the spatial environment sufficiently for it to be solved by a domain independent planner allowing a vehicle's mission to be planned using information about its spatial environment. Algorithms are evaluated using both simulated and real-world data showing that topological thinning can produce compact domains while maintaining a high level of routing efficiency, enabling the solution of the high-level mission planning problem. This thesis also examines the properties of topological belief compression and the effectiveness of path planning with non-uniform action costs using domain independent planners. To demonstrate the effectiveness of these algorithms, a planning and guidance system is tested on an Autonomous Surface Vessel (ASV) built around a five-metre Wave Adaptive Modular Vehicle platform (WAM-V). When performing simulated rescue tasks for 20 survivors before returning to a dock, the Symbolic With Refinement planner demonstrated plan generation resulting in a mean reduction in path length of approximately 15% when compared to a Greedy planning system.

Declaration

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Acknowledgements

I would like to thank the following people who assisted in this thesis;

Everyone who worked on AGVC 2013, AGVC 2014, RobotX 2014, RobotX 2016 and the WAM-V projects.

Don and Heath Eickhoff for the provision of images.

Robert Keane for his provision of the 10m Digital Elevation Model, and the rest of the Geographers.

Dr. Graziela Miot da Silva for the loan of the CeeScope sensor.

My Fellow PhD students.

Rowan Pivetta and James Armitage for explaining the Travelling Salesperson Problem.

My Aunt, Dr. Diane Russell for assistance with proofreading.

Mr. Richard Bowyer for providing another perspective.

The Australian Maritime College students, Supun Randeni Pathiranachchilage, James Keane and Reuben Kent. Supun also assisted in the refinement of the initial hydrodynamic model used by the WAM-V simulator.

The contributors to Wikimedia Commons and Open ClipArt which provided resources that were used to provide illustration of the concepts in this thesis.

Dr. Andrew Lammas for assistance with maritime control, in particular modelling of the TopCat vehicle, and the development of line-of-sight guidance.

My supervisors Professor Karl Sammut and Dr. Andrew Lammas. For their guidance and feedback on my writing.

I acknowledge that I am a recipient of the Australian Government Research Training Program Scholarship which supported this thesis.

List of Acronyms

ASV	Autonomous Surface Vessel
AUV	Autonomous Underwater Vehicle
COLREGS	International Regulations for Prevention of Collisions at Sea
GPS	Global Positioning System. A satellite based navigation system.
lidar	a sensor that uses laser light to measure range. Portmanteau of light and RADAR.
MOOS	Mission Oriented Operating Suite
NMEA	National Marine Electronics Association
NMEA-0183	A standard for communication between maritime devices. Commonly used for encoding GPS data.
PDDL	Problem Domain Description Language
RADAR	RAdio Direction And Ranging
ROS	Robotics Operating System
RRT	Rapidly exploring Random Tree
SONAR	SOuNd direction And Ranging
STRIPS	STanford Research Institute Planning System
T-REX	TeleoReactive EXecutive
TSP	Travelling Salesperson Problem
UAV	Unmanned Aerial Vehicle
WAM-V	Wave Adaptive Modular Vessel

Nomenclature

$\mathbb{R}$	The set of all real numbers
$\mathbb{R}^2$	The set of all two-dimensional real numbers
$\mathbb{R}^3$	The set of all three-dimensional real numbers
Sum Of Products	A Boolean statement consisting of terms containing only AND and NOT logical operators combined by an OR logical operator

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Advances in robotics have resulted in the development of machines with increasing capabilities, not only to perform actions, but also to be able to develop their own courses of action. A system that is capable of independent self-guided action can be said to be *autonomous*.

Early field robotic systems required either direct control by a human operator, or detailed plans to be provided. The increasing capability of mobile platforms means that future autonomous systems will be able to perform increasingly complex missions based on their own internally generated plans. If such systems are to be effectively utilised, then these plans must not only be both efficient to execute, but goals, possible actions, and plans must also be simple for a human to specify and review. This would enable a robotic system to be effectively supervised without requiring micro-management.

This thesis will cover a proposed approach for finding a system that can both incorporate spatial information allowing more effective plans to be built, while also allowing working with clearly defined actions and goals.

1.1 Development of Autonomy

The concept of autonomous machines has intrigued humanity since the first clockwork automata were exhibited. These machines used arrangements of springs and gears to simulate life and animation. With the exception of the Mechanical Turk, now considered to be

an elaborate trick [Campbell, 2004], they operated on a very limited basis, performing a few rote actions encoded by their mechanism. More complex behaviours were not possible until the development of information systems and the miniaturisation required to interface them to a mobile platform. With the development of such systems, the possibility emerged for a robot to demonstrate *planning*.

At its most fundamental, planning is the formulation of a set of actions to achieve desired objectives while satisfying constraints [Fox, 1994]. The application of planning to robotic systems has been attempted since the development of Shakey, described by Russell and Norvig as the first mobile general purpose robot [Russell and Norvig, 2010]. Shown in Figure 1.1, Shakey was developed at the Stanford Research Institute between 1966 and 1972. The high-level deliberative planning system developed for Shakey used an internal model based on predicate logic to store and reason with the state of its environment. This system, the first symbolic planner, was referred to as the Stanford Research Institute Planning System (STRIPS). STRIPS was able to formulate complex plans consisting of multiple actions by starting at a goal state and searching for actions that transformed the system from the goal to the initial state [Nilsson, 1984]. Table 1.1 shows the actions available to Shakey, when combined in the sequence created by the planner these actions constituted a plan that the robot could execute to achieve its assigned goal.

By the use of predicates to represent the state of the robot and it's environment, shown in Figure 1.2, STRIPS allowed the efficient searching for plans that fulfilled the robot's assigned goals. According to Brooks however, these single valued descriptions lacked the complexity to represent the interrelationships, including spatial relationships, that occur in the real world [Brooks, 1991b]. Shakey was capable of performing some spatial tasks, such as moving around it's carefully controlled environment and pushing objects around. However, the internal spatial model used by Shakey and other similar systems was only an approximation provided by its programmers. This was acceptable when that model was accurate, but Shakey's ability to maintain its internal model was limited, able to localise objects but unable to map its environment. This limitation could potentially cause the robot to generate and execute incorrect plans [Brooks, 1991a].

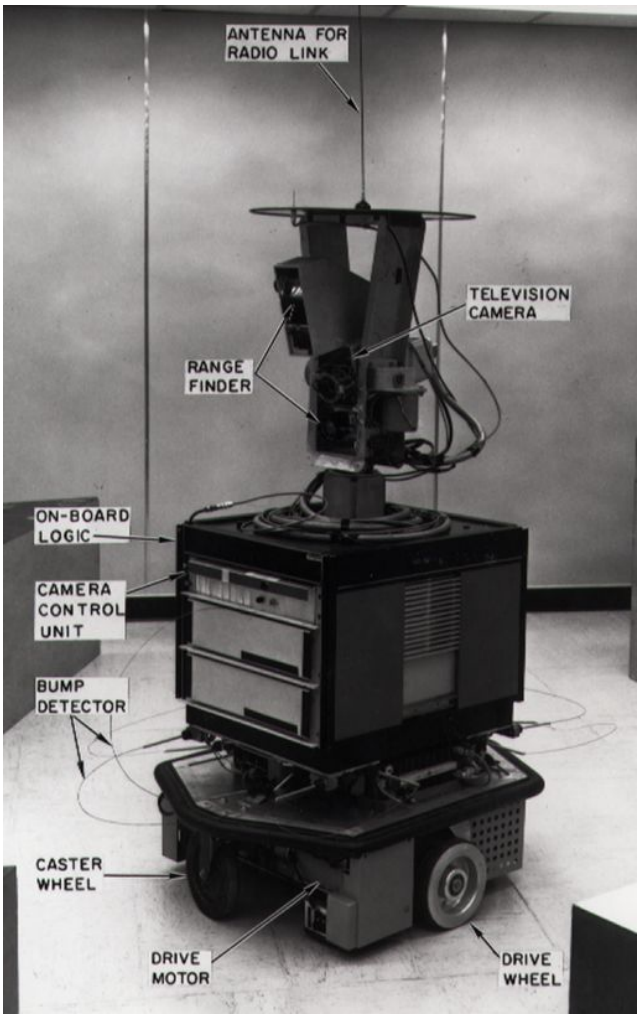


Figure 1.1: Shakey in operation surrounded by obstacles. [SRI International, 1972]. License: CC BY-SA

Table 1.1: Actions available to Shakey [Hart et al., 1972]

Action	Type	Parameter Type
GOTOB		object
GOTOD	go to	door
GOTOL		location
PUSHB		object
PUSHD	push an object to	door
PUSHL		location
GOTHRUDR	go through a door	door
PUSHTHRUDR	push through	door
OPEN	open a door	door
CLOSE	close a door	door

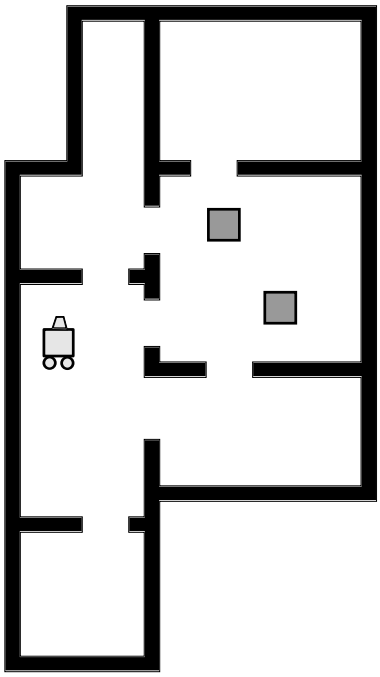


Figure 1.2: Drawing of Shakey's workspace as represented in [Hart et al., 1972]. Grey boxes are push-able objects.

Brooks proposed an alternative approach, known as the subsumption architecture, where decisions are made directly from sensor data, making the world the robot's model and allowing the robot to react directly to changes in its environment [Brooks, 1991a]. Brooks' concept simplifies the planner; by avoiding the necessity of internal state storage, the robot requires a correspondingly simpler control and computing system with a resulting decrease in required mass.

Closely related to the concept of planning is that of *scheduling* - the ordering of a number of pre-selected actions, assigning resources and setting start and finish times for actions to optimise or meet timing constraints. Like planning, this requires the ability to predict or model the result of actions to allow the cost of plans to be evaluated. The use of Brooks' subsumption architecture thus comes at a cost. By using the world as the model it cannot predict potential future states, thus preventing it from evaluating potential courses of action. For a scheduling based system to operate, some form of deliberation is required. Despite this, Brooks' criticism remains valid - a deliberative planning system can only operate within the domain that its belief can support. In particular, the manipulation of spatial data is normally performed by domain specific planning systems - path or trajectory planners - that generate detailed solutions based on the requirements of the top level planner.

In the four decades since the development of Shakey, the range of sensing methods and measurement accuracy available to a robotic system has increased greatly. Current generation research robots such as the Turtlebot [Open Source Robotics Foundation, 2014c] or PR2 [Oyama et al., 2009] can mount sensors such as the Microsoft Kinect, a depth camera capable of generating 9×10^6 depth points per second [Microsoft, 2014]. This improvement in depth sensing capability allows the use of point cloud based perception [Rusu and Cousins, 2011] and the construction of volumetric maps using systems such as Octomap [Wurm et al., 2010]. By allowing volumetric perception capabilities, the scope of what can be perceived by a mobile robotic platform has greatly increased. Shakey possessed bump sensors and a single colour camera interfaced wirelessly to a PDP-10 minicomputer. The use of Shakey's camera system was further limited to short periods due to the limited amount of battery available. In contrast, a typical current day smartphone contains not only multiple colour cameras, but a Global Positioning System (GPS) receiver, gyroscope, compass and more

than three orders of magnitude more RAM than was available to Shakey [SRI, 2015].

In parallel with the increase in locally generated data available to mobile platforms, there has been a similar increase in the amount of data available as on-line databases. In particular, the increasing number of Geographical Information Systems (GIS) has meant that any platform with network connectivity potentially has access to terabytes of spatially registered data. While it is impractical to expect a vehicle to store all possible relevant datasets, once the area of operation is known relevant subsets can be downloaded to the vehicle where it can be fused with data captured directly by the vehicle. Combined with increases in the accuracy of navigation aids such as GPS receivers, the challenge of planning is not the ability to build maps of the environment, but to process the available data into a form that can be used by the robot's on-board systems to efficiently work with the available information.

For a field robot engaged in the survey or measurement of its environment, the limits on its mission time can be both the endurance of the robot and the available time window for it to perform its task. To maximise the utilisation of the robotic platform in this window, the amount of time spent performing its function should be maximised with comparison to the travel time between observations. The return provided by this increased mission efficiency must be traded against the increase in resources required to maintain the more complex internal model. Data from sensors needs to be processed, stored, fused with other data sources, and compiled into a format that is suitable for use with the planning system. These tasks require computing resources: disk space for data storage, memory for data manipulation and CPU time for the execution of algorithms. For a practical planning system, it should also be capable of operating without the benefit of a carefully structured environment. When operating in the field, it is likely that the terrain it encounters will be shaped by weather and geographical processes into irregular forms, rather than ideal geometric shapes. The belief compression system should also be robust to changes in pose, since remapping of the input data may be required to align it with the internal model.

Despite the potential for autonomy, field robotic systems are commonly designed with a "human in the loop" architecture allowing control by a human operator. This model has been highly successful, being used on systems ranging from ROVs such as the BlueROV [Blue Robotics Inc, 2017] to military platforms such as the Packbot [Yamauchi, 2004]. However,

the workload of operating such a vehicle can be quite high. A human operator must be able to model the environment that a robot inhabits based on it's limited senses, and using this model construct plans to achieve the required goals. Murphy states that control of robots in disaster scenarios puts intense cognitive load on operators [Murphy, 2014], noting that in two cases where robots were stuck, simply resting the operator allowed the situation to be successfully resolved.

While the tele-operation model has worked well in the past, it is of limited use in situations where oversight is impractical due to signal limitations. This highlights an important quality in an autonomous system; human intuition and insight is still required in complex situations, but in a hostile environment, such guidance cannot be guaranteed. By combining human oversight with automation, a system with *supervised autonomy* can be developed. In this model the vehicle is capable of performing operations under the direction of a human operator. If the vehicle can also act autonomously, for example detecting that it has lost contact and attempt to either reacquire communication or recover itself, the result is a *shared autonomy* system. An interest in the development of supervised autonomy systems may have shaped the recent Defence Advanced Research Agency (DARPA) humanoid challenge [DARPA, 2012]. This robotics competition used a paradigm of shared autonomy under limited communication for its control. Teams were provided with a constant low-bandwidth communication channel interspersed with short bursts of high-bandwidth communication. Notably the team from Worcester Polytechnic Institute reported having full communication losses on their provided network connection for more than six minutes during a competition task [Atkeson et al., 2015]. A further example of this is the clean-up of the Fukushima Daiichi nuclear accident. The Quince robot from the Chiba Institute of Technology had to be modified with a wired cable connection because the reactor containment vessels in some buildings blocked wireless signals. The vehicle was later lost when this cable failed [Kawatsuma et al., 2012]. If such communication issues are present in a carefully controlled competition environment, the uncontrolled field environment is likely to be even more prone to such failures. As such, shared autonomy systems become even more attractive.

1.2 Existing Approaches

1.2.1 Applications for Autonomy in Marine Field Robots

Before a planning system can be developed, the potential applications need to be examined. A number of applications for maritime vehicles exist in the area of environmental monitoring. These can involve the preparation of bathymetry, sampling of water and marine life or the visual examination of structures. The implementation of these tasks may involve the execution of a pre-planned mission, the creation of dynamic plans based on immediate sensor data, or a hybrid approach combining pre-defined and dynamic plans.

A further area of potential interest is the mapping of sea grass beds. Imaging techniques such as satellite and aircraft based imagery have been used, but when compared to data gathered by direct observation, Phinn et al. found that the result of classification rarely exceed 50% accuracy [Phinn et al., 2008]. Space and airborne optical systems also have the limitation that visibility may be altered by absorption and turbidity of the water. While sea grasses are photosynthetic, and are thus limited at the depth they can exist, Duarte's survey paper estimates that many species have a mean colonisation depth of more than 20m [Duarte, 1991] while Dekker et al. observed Secchi Depths, a measure of visibility from the surface, of 1.2 to 2m during sea grass fieldwork at Wallis Lake [Dekker et al., 2005]. A potential solution to the problem of visual inspection of sea grass beds, is the utilisation of Autonomous Underwater Vehicles (AUV). Roelfsema et al. noted that an AUV can provide data from depths greater than 2.5m [Roelfsema et al., 2015]. By bringing the sensor closer to the subject, an AUV can thus give a better estimate of the sea grass composition and extent than observation from above the surface.

Another area of interest is the sampling of water for measurement of water quality and biological content. Ryan et al. attempted to sample water from the Intermediate Nepheloid layers, which are described as layers of turbid water that develop episodically from the bottom layer [Ryan et al., 2010]. Since the sampling ability of their AUV was limited, and the emergence of suitable layers could not be reliably predicted, their vehicle had to be able to develop its own plan of water sampling events based on the sensor data that it immediately

perceived from the environment. As noted by Ryan et al., an area of interest may not only be difficult to identify before the start of a mission, but due to changes in the ocean caused by currents and diffusion it may in fact change during the mission itself [Ryan et al., 2010]. As such, the ability of a vehicle to adapt its mission plan to perceived changes in its environment may be vital to the correct performance of its mission.

Camilli et al. similarly covered adaptive path generation for AUVs in survey missions. This particular application involved localisation of methane in the water column [Camilli et al., 2004]. This paper describes a simple AUV guidance system that would automatically execute a grid pattern if the detected propane level exceeded a threshold. This group would later go on to survey the extent of hydrocarbon plumes after the Deepwater Horizon accident [Camilli et al., 2010].

In addition to the applications where oversight is limited, autonomy can be useful for its precision and repeatability. A control system can potentially allow an autonomous vehicle to more accurately perform a mission plan than a human operator. In their comparison of snorkelers versus AUVs for performing visual survey transects for sea grass maps, Roelfsema et al. found that their AUVs provided a more consistent speed of traverse, frequency of sampling, and more precisely repeated their previous transects [Roelfsema et al., 2015].

In summary, autonomous vehicles can be applicable in maritime environmental monitoring when;

- visual observation of objects and areas is required beneath the depth visible from the air or space
- direct sampling of water or marine life is required

Autonomy can provide advantages over remotely operated vehicles when;

- communication with the vehicle is limited
- repeatability of a plan needs to be maximised

Additionally compared to an ROV, an AUV has the following operational advantages;

- elimination of a dedicated support vessel

- elimination of the highly-skilled ROV operator role

The on-board autonomy does not need to be complex. Undersea gliders such as the Slocum and Seaspray can have mission durations in the order of weeks while navigating between continents, but they require little in the way of sensing and control since they travel through open water. As such, ocean gliders can perform missions with only simple intertial guidance and a GPS receiver [Rudnick et al., 2004]. For the ability to plan spatially to be useful, the spatial environment must be non-trivial. This is more likely to occur in inshore and littoral areas where there may be shoals islands and shallow water that may cause damage or prevent a vehicle from performing its mission.

The next two sections will provide a brief overview of some sample vehicles that can perform environmental monitoring, along with a brief overview of their suitability for operations in shallow and confined waters.

1.2.2 Autonomous Underwater Vessels for Environmental Monitoring

An AUV is a robotic vessel capable of travelling underwater without a tether and with some amount of internal guidance. Amongst the earliest such vessels were the Self-Propelled Underwater Research Vehicles (SPURVs) that were controlled directly by a surface vessel using acoustic communications [Widditsch, 1973]. While being little more than Remotely Operated Vehicles (ROVs) that lacked a tether, nevertheless these vehicles lead to later vessels such as the Remote Environmental Measuring UnitS (REMUS) [Von Alt et al., 1994] and AUTOSUB [Collar and McPhail, 1995] that had on-board computer systems capable of executing plans.

These later vehicles were still relatively limited in computing power; they could execute pre-planned missions, but lacked the ability to perform major re-planning of their missions during operation. The ability to re-plan would not be a feature until the development of vehicles such as the Monterey Bay Aquarium Research Institute's (MBARI) Dorado AUV which uses the Teleo-Reactive Executive (T-REX) system for autonomy. T-REX will be discussed further in Section 1.5.3.2.

1.2.2.1 Remote Environmental Measuring UnitS (REMUS)

The original prototype Remote Environmental Measuring UnitS (REMUS) AUV was developed in the mid 1990s [Allen et al., 1997] has lead to a family of vehicles ranging from the 40kg REMUS 100 to the 900kg REMUS 6000. The smaller vehicle is of most interest for littoral operations, due to its more compact nature.

With a 1.5kWh battery unit and a mission time of up to twelve hours, the power budget of the current REMUS 100 is limited to approximately 125 Watts for propulsion, sensors and data processing [HYDROID, 2016]. This leaves relatively little scope for performing the calculations that would be required for a complex mission planning system. As such, the REMUS is likely to be operated using only pre-generated mission plans, or those dynamically updated from a base station.

An image of a REMUS vehicle on static display can be seen in Figure 1.3.

1.2.2.2 Dorado

The Dorado vehicle is a modular AUV developed by MBARI for a variety of environmental monitoring tasks. It has been used for direct water sampling using spring loaded plungers to draw ocean water into sampling containers when criteria are met [Rajan et al., 2009]. Other variants of Dorado have been used for ocean mapping [Caress et al., 2008]. Dorado is a larger vehicle than the REMUS 100, with a mass of approximately 680kg in its mapping configuration.

1.2.2.3 New Small Autonomous Underwater Vehicles

Increasing miniaturisation has allowed the further reduction in size of AUVs. In particular vehicles such as Bluefin Robotics SandShark [Blu, 2017] and the Iver 3 EcoMapper. The Iver series is notable for an emphasis on low-cost using Commercial Off-The-Shelf (COTS) parts [Anderson and Crowell, 2005]. A derivative of the Iver 2 has been modified to carry a second backseat computer module which could increase the scope for detailed on-board plan generation [Universidad Politechnica de Cartagena, 2014].

Figure 1.3: Remote Environmental Measuring UnitS (REMUS) 100 vehicle on display. Image by [MKFI, 2014]. License: Public domain.

1.2.3 Autonomous Surface Vessels for Environmental Monitoring

As mentioned in the previous section, Autonomous Underwater Vehicles have shown application in environmental monitoring tasks. Autonomous Surface Vessels (ASVs) can potentially perform similar tasks while also providing more accurate localisation sensor data using Global Position System (GPS) and visual data. ASVs are also less limited in volume; unlike an AUV, they are not required to be ballasted to reach neutral buoyancy. This allows more of their mass to be allocated to power and sensing systems, potentially allowing a longer range than an equivalent mass AUV. The primary disadvantages of an ASV compared to an AUV are the limitations on the depth of water that their sensors can perceive, and a susceptibility to wind and wave effects that underwater vehicles can pass beneath.

This section will provide a brief overview of four ASV platforms that have been developed for inshore and riverine environmental monitoring.

1.2.3.1 Riverwatch Autonomous Surface Vessel

Built by researchers from the Instituto de Desenvolvimento de Novas Tecnologias, the Universidade Nova de Lisboa, and Instituto Universitário de Lisboa, Riverwatch is an autonomous

surface vessel comprised of an electricly powered catamaran hull. The concept of Riverwatch is to provide a platform for environmental monitoring of riverine environments that combines the capabilities of both airborne and surface platforms to allow long range aerial mapping in addition to lidar[1] and SOuNd direction And Ranging (SONAR) mapping of the environment. Carrying cameras and a launching platform for an Unmanned Aerial Vehicle (UAV) in addition to its lidar and SONAR sensors, the platform is capable of sensing both above and beneath the waterline while also being able to examine nearby land areas [Pinto et al., 2014a]. The image in Figure 1.4 shows the vehicle with its UAV hovering in front of the landing area [Pinto et al., 2014b].

1.2.3.2 TopCat Autonomous Surface Vessel

Originally developed for the 2014 Maritime RobotX Challenge [Maritime RobotX Challenge, 2014] by a combined team from Flinders University and the Australian Maritime College, TopCat is a 5m Autonomous Surface Vessel based on a twin hull Wave Adaptive Modular Vessel (WAM-V) [Marine Advanced Research Inc., 2015]. The vehicle was later modified by moving the battery carriage, updating the electronics, and the addition of a cover to the electronics. This was the version that competed in RobotX 2016, as seen in Figure 1.5.

The vessel's environment is perceived by a combination of a surface search RADAR, lidar and a camera system. Localisation information is provided by a dual antenna GPS with a Real-Time Kinematic (RTK) connection to a reference station, and an Attitude Heading Reference System (AHRS). A wind sensor is used to estimate windage effects on the surface vehicle.

TopCat is driven by a pair of Torqeedo Cruise 2R electric outboards powered by a pair of Kokam 3.9kWh lithium polymer (LiPo) batteries. Power supplied by these batteries to the on-board systems allows an endurance of up to five hours at a four knot cruise speed.

Despite its origins as a competition vehicle, TopCat was developed for an eventual research role. This has involved using TopCat's deployment mechanism to carry sensors such as

[1]This thesis will use the convention used by the Macquarie Dictionary that lidar is a noun created as a portmanteau of laser and RADAR, rather than an acronym [Macmillan Publishers Australia, 2018].

Figure 1.4: Riverwatch Autonomous Surface Vessel with hovering Unmanned Aerial Vehicle [Pinto et al., 2014a]. Image ©2014 IEEE .

underwater cameras and depth sounding equipment. To demonstrate that the ASV is suitable for such tasks, details of a mission to measure bathymetry with TopCat can be found in Appendix A.

Future developments of TopCat may include modification to carry a marsupial vehicle for the deployment of liferafts [Machado et al., 2014] similar to the ROAZII USV [Martins et al., 2007] in the ICARUS project [De Cubber et al., 2013]. In this configuration, TopCat could visit survivors of a boating disaster and provide assistance, deploying survival equipment to those in distress.

1.2.3.3 Z-Boat Autonomous Surface Vessel

Much smaller in scale than the Riverwatch and TopCat platforms, the Z-Boat was designed for the survey of small water bodies [Teledyne Oceanscience, 2015b]. This platform has found application in tasks such as depth measurement in areas that are too small or too hazardous for a manned platform such as mine tailing dams. Initially developed with a tele-operated control system, the vessel was found to be capable of operation far beyond the visual field of the operator. To allow the full capability of the boat to be used when surveying, the manufacturer of the Z-Boat has announced the availability of a full auto-pilot allowing it

Figure 1.5: TopCat Autonomous Surface Vessel at Maritime RobotX 2016. Cropped from original image by Andrew Webb.

to follow a pre-planned path during survey operations [Teledyne Oceanscience, 2015a].

1.2.3.4 Lizhbeth Autonomous Surface Vessel

Lizhbeth is an ASV developed at ETH-Zurich for the task of measuring water quality at a variety of depths. Initially used to measure algal growth in Lake Zurich, this system should be applicable to other similar tasks [Hitz et al., 2012]. This vehicle was later used to demonstrate a system for informed planning and re-planning of paths to reduce uncertainty in the property being measured [Hitz et al., 2014].

1.3 Applications for Autonomy in Inspection

Platforms such as Riverwatch, Lizhbeth and TopCat are capable of carrying sensors on a smaller platform than would be required for a comparable manned platform, however for a mission to be executed, a plan that will achieve the mission goals is required. For an inspection task, this may require carrying the sensor to multiple locations where the object(s) of interest can be clearly viewed. This problem is referred to as *coverage planning*.

Coverage planning systems has applications ranging from cleaning floors [Kleiner et al., 2017] to preparing structures for painting [Ren et al., 2008]. From a maritime field robotic

Figure 1.6: Lizhbeth Autonomous Surface Vessel [Hitz et al., 2012]. Image ©2012 IEEE .

context, one of the more interesting approaches is the lazy tour.

Hover et.al. describes a complete system for the planning and execution of hull inspection missions [Hover et al., 2012] using an intervention type AUV, the Hovering Autonomous Underwater Vehicle (HAUV) to perform a search of the hulls of two ships, the SS Curtis, and the USCGC Seneca. While much of ships hull is a smooth curve, parts such as the propeller and associated hardware are complex shapes that cannot be trivially solved. The method used for inspection path generation was the Travelling Salesperson Problem (TSP) based system originally described in Saha et al. [Saha et al., 2006].

Englot and Hover used a sampling based technique to provide coverage of complex shapes. A sample set was chosen that provided the required coverage, with the TSP tour generated using Christofides algorithm [Christofides, 1976] and refined using the Lin-Kernighan algorithm [Lin and Kernighan, 1973]. To allow the estimation of collision-free distances to be found, the Rapidly exploring Random Tree (RRT) algorithm was used however the expense of calculating RRT paths for all possible paths was considered impractical. The initial paths were thus calculated using only Euclidean distances, with RRTs being used to calculate the distances between vertices that selected to be on the tour. This process was repeated with new tours being generated, and their distances being updated until the overall tour length was stable. Since the distances were only calculated for the connections on the tour, this algorithm is referred to as the *lazy tour*.

1.4 Applications for Autonomy in Search and Rescue

Another application for autonomous vehicles is maritime Search and Rescue (SAR). Unlike an environmental monitoring mission where external data sources can provide the ability to produce plans at leisure, SAR tasks can combine both limited information and a rapidly developing environment where human lives may rely on rapid and efficient response.

A project to develop assistive robotic tools for search and rescue operations is ICARUS[2]. Two real-life disasters were used as a reference, an earthquake in Haiti, and a shipwreck involving a passenger liner. A variety of robotic platforms were integrated to map the environment, search for survivors and deploy life-saving equipment [De Cubber et al., 2013].

ICARUS's second scenario culminated in a field exercise that simulated an event similar to the Costa Concordia shipwreck. A variety of sea and air platforms were deployed under a unified control system, including Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs) and Unmanned Surface Vehicles (USVs) to map and search an area. Data were fused from multiple sources to produce tasking for the multiple autonomous platforms in operation, and vehicles could then provide direct support to survivors by the deployment of lifesaving equipment [Machado et al., 2014].

The majority of platforms used in project ICARUS were general purpose, but two specific autonomous surface vehicles deserve closer consideration.

1.4.1 EMergency Integrated Lifesaving LanYard (EMILY)

The EMergency Integrated Lifesaving Lanyard (EMILY) is a small-scale autonomous surface vessel that has applications in monitoring as well as SAR. In its rescue configuration, EMILY operates as a self-deploying emergency float that can be sent out through surf to assist a swimmer in difficulty [Patterson et al., 2013]. An image of EMILY on display can be seen in Figure 1.7.

[2]Despite the capitalisation, this does not appear to be an acronym.

Figure 1.7: EMergency Integrated Lifesaving LanYard (EMILY) at the 2016 Naval STEM Exposition. Image cropped from original by [Office of Naval Research, 2016]. License: CC BY 2.0

1.4.2 The Unmanned Capsule (UCAP)

Developed as part of Project ICARUS, the Unmanned Capsule (UCAP) is a small unmanned surface vessel developed specifically for rescue. UCAP was designed to carry a 4-person life-raft at least 1.2 kilometres at two knots. Once reaching the location of a survivor, UCAP would deploy its life raft [Matos et al., 2013].

1.5 Software for Autonomous Vehicles

The implementation of robotic planning and control algorithms presents its own challenges. Early robots such as Shakey used standard mainframe programming environments - Shakey's STRIPS planner was implemented in the LISP language on a PDP-10/15 [Hart et al., 1972]. The first subsumption architecture robot, Allen, similarly used an offboard LISP machine to emulate its hardware, while its successors Tom and Jerry, used Programmable Array Logic (PAL) chips to implement their control system [Brooks, 1990]. In the decades since the creation of these robots, a number of systems have been created to simplify the development of robot software. This section will provide a brief overview of the systems commonly in use for maritime field robots.

1.5.1 Mission Orientated Operating Suite (MOOS)

The Mission Oriented Operating Suite (MOOS) is a software system for mobile robotics that uses a central database to pass messages between independent processes. Each individual task within the robot's control system is a separate program running as an independent process in its own memory area. Since none of these programs share memory, communication between processes is performed using message passing. Messages are limited to either a string or a single double-precision number and are sent to named *topics* that can be subscribed to by multiple other nodes [Newman, 2008]. Since all messages travel through the central node, the communications of this system form a star architecture.

The reduction of a complex task into several individual simple tasks resembles the Subsumption architecture. This also potentially allows these individual systems to be re-used. For example, a system for interpreting fix data from a GPS unit can be developed once and then used across may different platforms.

MOOS was used with both AUVs such as the Yellowfin [West et al., 2010] and ASVs such as MIT's Scout [Curcio et al., 2005]. The flexibility of MOOS is sufficient that MIT later used it to develop an automated control system for following maritime navigation rules [Benjamin et al., 2004].

1.5.2 Robotics Operating System (ROS)

Quigley et al. identified the origins of the Robotics Operating System (ROS) as the Switchyard system that was first implemented as part of the STanford Artificial Intelligence Robot (STAIR) [Quigley et al., 2007] and Personal Robot 1 (PR1) programs [Wyrobek et al., 2008] [Quigley et al., 2009].

In ROS, communication between programs - referred to as *nodes* - is also done using topics, identified by a text string. In the ROS system, a topic is created when one or more programs advertise their ability to *publish* information as *messages* on this topic name. A topic can have zero or more *subscribers* that will receive these messages upon publication. A centralised system for connecting nodes exists, but is not involved in communication once the

connection is established. Thus, the communication in ROS is peer-to-peer.

Interpretation of the communicated messages would not be possible if there was not a standardised method for the serialisation and de-serialisation of data. In addition to a system for the generation of custom message types, ROS provides a set of generic message types for commonly used sensor and applications types allowing communication between nodes. As an example of such a message type, Figure 1.8 shows the LaserScan message type for messages from lidar sensors. Since each message header contains information on the parameters of the scan, this type can be applied to a variety of different lidar scanners without modification. Similarly, it is possible to develop higher level software that works with lidar data and that is independent of the exact design of the sensor that is in use. Thus, a tool-based system consisting of small independent programs can be developed. These nodes can be combined together to build more complex solutions.

An example of the homogeneity of ROS based tools is the wide support for lidar devices. Drivers exist for sensors ranging from the XV-11 [Perko et al., 2016] found on robotic vacuum cleaners, up to the 3D sensing Velodyne units [O'Quinn, 2016]. Notably, the planar lidar units all produce laserscan messages on their data topics, while the volumetric models produce a message type that encodes point cloud data that can be used with the Point Cloud Library (PCL) [Rusu and Cousins, 2011].

In addition to communication between systems, a standardised method for the recording of messages exists. Rosbag allows the recording of multiple topics in a time-stamped file. This allows both post processing to analyse the behaviour of a system, and replay to allow a program to be executed with input data as if it were being executed on a running robot [Field et al., 2014].

In the past ten years, ROS has been used on a large array of systems. The re-usability of ROS has made it particularly attractive for development due to the large range of software and device drivers now available. This has led to the current time of writing when one hundred and thirteen robots and related devices now have portal pages on the ROS wiki [Open Source Robotics Foundation, 2018]. While aimed at ground robots such as the PR2 [Cousins, 2010], the flexibility of ROS has allowed it to be used on maritime robots such as

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Figure 1.8: sensor_msgs/LaserScan message type

the Riverwatch ASV [Pinto et al., 2014b], TopCat ASV and [Sammut et al., 2014] Yellowfin AUV [DeMarco et al., 2011]. The paper describing support for ROS on the Yellowfin is notable in that the authors developed a gateway between ROS and the existing MOOS based control system. They note that while MOOS is capable and has access to systems such as IvP-Helm that support maritime navigation, ROS has a more flexible implementation and access to a larger software ecosystem [DeMarco et al., 2011].

The topic based communication system in ROS is useful for streaming based systems such as lidar and vision data, but planning architectures require the ability to dispatch actions and receive confirmation of their completion. Synchronisation of such systems can be difficult, with particular importance on knowing if a planner is active, what goal each system is currently attempting to achieve, and the progress of a planner to achieving this goal. In ROS, the actionlib library provides an interface that supports feedback and pre-emption using the underlying topic-based communication system. Actionlib supports the ROS system for custom message creation, thus allowing domain specific goal, feedback and result messages to be generated. This has the further effect of allowing these control messages to be recorded along with sensor and status information using the rosbag system.

Actionlib has become ubiquitous enough that interfaces are provided on systems such as the move_base 2D navigation package and the moveit arm planner [Open Source Robotics Foundation, 2014a]. State machine based executive systems such as SMACH [Bohren and Cousins, 2010] and FlexBE [Schillinger, 2015] are able to support the execution of actions

using Actionlib, and can use the response of the Actionlib server to update their state.

A limitation of Actionlib is that since it is based on the ROS topic system, communication is not guaranteed. ROS does cache data at the publisher, so re-transmission is possible, but for full reliability monitoring of a client's execution state will be required.

1.5.3 Implementation of Autonomy in Maritime Autonomous Vehicles

Previous sections have examined the platforms, software and applications for maritime autonomy. This section provides more detail on the architectures and systems used for automated planning in autonomous marine vehicles. A brief summary of significant approaches is found in Table 1.2.

1.5.3.1 Reactive Planners

Various systems for autonomy have been developed such as the petri-net based scripting system by Palomeras et al. [Palomeras et al., 2012], the multi-AUV system developed by Sotzing [Sotzing et al., 2008] and the Mission Oriented Operating Suite (MOOS) developed by Newman [Newman, 2008] and already covered in Section 1.5.1. Of particular interest is the scripting system used by the AUTOSUB AUV. In a 2010 paper the top level control system of the vehicle was not only described, but state machine representations and interpreted code were presented. The paper provides a particularly detailed look at a modern reactive AUV control system [Molnar et al., 2010].

1.5.3.2 Teleo-Reactive EXecutive (T-REX)

The Teleo-Reactive EXecutive (T-REX) was developed by the Monterey Bay Aquarium Research Institute based on the EUROPA planner. Plans in EUROPA are expressed as timelines showing when actions should occur, while the state variables are stored as domains representing the set of all possible values that a variable may take [Barreiro et al., 2012].

T-REX uses EUROPA's timeline representation to schedule access to the AUVs resources [McGann et al., 2007]. The ability of T-REX to update its plan based on changes in the environment has been used to execute actions based on immediate stimuli. This has allowed experiments to be performed that require local decision making capability such as mapping algal blooms [Rajan et al., 2009]. This system is capable of performing spatial tasks - travelling to waypoints, and moving relative to objects - but when operating in environments such as coastal areas where the vehicle may run aground, human oversight of the mission plan is still required to ensure that the vehicle remains within a safe operating area [Das et al., 2011].

1.5.4 Symbolic Planning

The STRIPS planner was the first in a category of systems referred to as symbolic planners. By using a domain model defining the relationship between possible actions and results, and a task model describing the starting condition and the desired goal, plans can be generated consisting of a sequence of actions resulting in the goal being reached [Knoblock et al., 1998]. This abstraction allows planners to be applied to applications beyond the concept envisioned for Shakey, extending to areas such as combining symbolic action plans with geometric planning for arm manipulation tasks [Dornhege et al., 2009], and the generation of temporal search plans [Bernardini et al., 2013].

A visualisation of Shakey's environment is shown in Figure 1.2. This limited environment had six rooms and a number of objects that could be pushed. Despite this simple environment, Shakey's set of possible actions were surprisingly numerous with separate movement commands for moving to objects, locations and doors. The importance of the spatial connectivity in its environment can be seen in this set of actions. As shown in Table 1.1, more than half of the actions used a door as their parameter [Hart et al., 1972].

1.5.4.1 Description Languages

Symbolic planning systems have been adopted widely enough that a number of planning languages have been developed to encode symbolic information. The most commonly used of these has been the Problem Domain Description Language (PDDL), a language developed for the Artificial Intelligence Planning Systems Conference to allow planning systems to compete [Knoblock et al., 1998]. Updates to this language have included the addition of time and numerical quantities [Fox and Long, 2003], while support for probabilistic planning with rewards was added to the derived Probabilistic Planning Description Language (PPDL) [Younes and Littman, 2004]. Belta et al. provides a useful overview of symbolic planning, discussing the general architecture of a planning based robotic system including the use of motion primitives to build a vehicle path [Belta et al., 2007].

PDDL based systems using symbolic manipulation are not the only method used for planning in robotic systems. The EUROPA planner used by T-REX originally used the New Domain Definition Language (NDDL) to describe its domains, before later adopting the Action Notation Modelling Language (ANML) [Smith et al., 2008].

What symbolic systems typically lack is an ability to work directly with spatial values. NDDL and later revisions of PDDL have support for single dimensional variables, but no direct support for spatial concepts such as areas and volumes. Shakey's STRIPS based planner supported coordinates, but these were not manipulated by the plannner and could be replaced by named waypoints without altering the behaviour of the planning system. Notably, none of Shakey's example plans in Hart et al. used the support for numerical coordinates [Hart et al., 1972].

1.5.4.2 Symbolic Planning with Real-World Systems

To develop a planning domain capable of supporting spatial concepts, some method of decomposing the spatial environment into areas or volumes is needed, where the decomposed components are then labelled and spatial relationships are generated. Each area could then be tagged with information providing a semantic context for the domain model. Thus, some

model of identifying and characterising the space is required. Belouaer, Bouzid and Mouaddib developed an ontology for representing fuzzy human spatial concepts with symbolic values [Belouaer et al., 2010]. That was later developed into a spatial extension for PDDL [Belouaer et al., 2012]. This could be used to make an interface allowing naturalistic interactions with spatial properties, however a method for generating the domain model is still required.

Real world robotic problems can be much more complex, including multiple actions that need to be performed at locations connected by areas with many possible patterns of traversal. If the domain is extended to include obstacles that prevent a robot from travelling directly between objects, then there may be multiple paths that can be taken. Such a task is not unprecedented: even a task as simple as fetching a beverage may require the traversal of multiple doorways and elevators [Cousins, 2010] requiring the evaluation of many potential paths to efficiently perform its assigned task. Since determining the optimal path to an object is a form of search, this could be integrated into the same search process that is used to produce the robot's plan. However, the expansion of complexity in the domain and task must be balanced by the increase in size of the search space and the corresponding time to reach a solution. The time required to search may be irrelevant for off-line systems but, if the plan is being generated on the vehicle, then production of an optimal plan is inefficient if the time saved compared to a less optimal plan is less than required to do the search. Thus, if the vehicle is to be capable of creating mission plans based on the changing environment that it perceives, then a trade-off may be necessary. A planner that can rapidly produce plans that approach the optimal based on a changing environment can be desirable.

Bylander examined the computational complexity of STRIPS planning and found that except for the simplest cases, the algorithmic complexity of planning is PSPACE-complete [Bylander, 1994]. As such, expanding the states to be searched to allow all possible paths to be examined would rapidly result in a system that could not produce a result within a meaningful time period. This means that path planning is typically restricted to domain specific planners, while domain independent planning is typically restricted to the deliberative layer. Bylander's analysis of planning complexity in STRIPS type planners showed that the complexity scaling of a symbolic planning domain is based on the number of pre- and post-

conditions that are associated with the actions. Strict restrictions on these conditions when searching for actions allows the task to be performed in polynomial time rather than being a non-deterministic polynomial complete (NP-COMPLETE) or polynomial space complete (PSPACE-COMPLETE) task [Bylander, 1994].

A possible solution to the problem of integrating symbolic and spatial planning is the use of *semantic attachment* - the integration of an external domain specific planning system to a domain independent planner. Dornhege et al. used this approach to integrate a probabilistic roadmap based arm motion planner with a symbolic planner to perform a simulated urban search and rescue task [Dornhege et al., 2012]. Muñoz et al. similarly integrated symbolic and path planning by building a path planner into their symbolic planner [Muñoz et al., 2016]

The ubiquity of ROS has resulted in a number of executives being developed. Arguably the system with the most complete support for symbolic planning is the ROSPLAN system developed at King's College. ROSPLAN uses a database to store the robot's belief, and can use this information with a PDDL based symbolic planner to generate plans. Actions can then be dispatched to a robot using Actionlib.

The software distribution available on the GitHub website includes a demonstration of using a turtlebot to visit a set of waypoints whose cost is generated using an RRT-Tree [KCL-Planning, 2016], while Cashmore et al. describes the use of the planner with the Girona 500 AUV to manipulate switch panels in a test tank environment [Cashmore et al., 2015].

Due to the importance of symbolic planning to this thesis, a more detailed example is included in Appendix B. This appendix introduces the concepts behind a domain-independent symbolic planning system using some of the same PDDL code used in the planning systems later in this thesis.

Table 1.2: Summary of Maritime Autonomy Approaches

Name	Mission/Domain Encoding	Action Selection
Component Oriented Layer-Based Architecture for Autonomy (COLA2)	Petri-net	Petri-net player [Palomeras et al., 2012]
Teleo-REactive EXecutive (T-REX)	New Domain Description Language (NDDL)	EUROPA solver [McGann et al., 2007]
ROSPLAN	Problem Domain Description Language (PDDL)	Symbolic planner [Cashmore et al., 2015]

1.5.5 Path Planning for Maritime Vehicles

In a robotics application, some form of path generation is required. Since the space to be explored generally can include the space of all possible vehicle positions, a planning system specifically designed for the problem is required. Domain specific spatial planning algorithms for AUVs include the Fast Marching system by Pêtrés et al. [Pêtrès et al., 2007], and the system by Kruger et al. that allow for the effect of estuarine currents [Kruger et al., 2006].

Other popular algorithms include the use of Rapidly exploring Random Trees (RRTs), a sampling based algorithm that explores the space between obstacles [LaValle, 1998]. RRTs can use vehicle models to generate feasible trajectories, but are commonly designed for simple straight paths. RRTs have been applied to maritime vehicles including the NaviGator ASV entered in the 2016 Maritime RobotX challenge [Frank et al., 2016].

Generation of minimum cost paths is only part of the maritime planning problem, vehicles are also required to be compliant with the International Regulations for Prevention of Collisions at Sea, 1972 (COLREGS). Benjamin and Curcio investigated COLREGS navigation for ASVs [Benjamin et al., 2004] which was later used in the development of the vehicle described in Benjamin et al. [Benjamin et al., 2006]. This system used the interval planning model developed by Benjamin [Benjamin, 2004].

Another approach is the use of lattice planning [Pivtoraiko and Kelly, 2005], searching through the spatial environment using sets of motion primitives to find a kinematically acheivable solution to the problem. Shah et al. uses such a search based approach, along with a flaw finding based solution for repairing plans that violate COLREGS constraints [Shah et al., 2016].

Agrawal and Dolan used A* search through a four-dimensional configuration space to identify feasible paths that avoided obstacles while meeting COLREGS constraints [Agrawal and Dolan, 2015].

1.5.6 Sources of Belief

Planning systems can attempt to provide solutions to tasks, but the ability to plan is dependent upon the accuracy of the robot's internal model of the world. This model can be built and updated from both internal and external sources. Additionally, if the planning system is to demonstrate on-line re-planning based on changes in its environment, a system of interpreting sensor data would be required. This system would allow the vehicle to update its knowledge of the environment in real time during the mission.

1.5.6.1 On-Board Data Sources

For a wide ranging robot to effectively plan using its belief, data need to be stored in an efficient manner. Robotic systems have typically used grid based systems for map creation. ROS based systems can use the costmap2D package for storing map cells as free, occupied and unknown [Marder-Eppstein et al., 2018]. Data sources for such costmaps can include a number of sources include Simultaneous Localisation and Mapping (SLAM) algorithms such as Gmapping [Kümmerle et al., 2011].

Storage of map data for three dimensions could be performed in a similar manner, but for a regular grid the space requirements would scale with a power of three, rather than a power of two. A potential solution to this problem is the use of hierarchical systems for data storage such as Octomap - an octree based system for storing probabilistic models [Wurm et al., 2010]. Since measurements contain uncertainty and noise, the use of a probabilistic storage system combined with measured and estimated values for sensor performance could allow this uncertainty to be estimated. These error and uncertainty estimates could then be utilised by the planning algorithm, for example constructing search patterns to maximise the likelihood of finding an object. Octomap has already been used for notable systems including a stereo camera based exploration system [Shade and Newman, 2011] and the spatial relation system used by Sjöö et al. [Sjöö et al., 2010]. Octomap is a capable system, but its method of object orientation can complicate its use. The base tree class implements the methods required to create a tree, but the methods required to set a value in a leaf vary by subclass. This lack of commonality can complicate the creation of a general method for the

handling of octrees.

In addition to Octomap, the Point Cloud Library [Rusu and Cousins, 2011] also implements an octree that supports the indexing of point locations which allows for the rapid searching for nearest neighbours. This differs from Octomap's implementation where the tree structure stores the data.

Interpretation of visual and SONAR data to maintain the internal belief state is also required. An example of a visual interpretation system for maritime use is the Monterey Bay Aquarium Research Institutes (MBARI) Automated Visual Event Detection and Classification (AVEDac) system, but these may have to be adapted for operation on a mobile vehicle [Cline et al., 2009].

Terrain and object reconstruction from SONAR data were demonstrated by Papadopoulos et al. This technique constructed voxel objects by merging SONAR scans [Papadopoulos et al., 2011]. This type of system would build models over time that could then be combined with pointcloud recognition algorithms such as the Viewpoint Feature Histogram [Rusu et al., 2010] to identify objects. Another example of a SONAR interpretation system is AUTOTRACKER - a pipeline tracking system for sidescan SONAR [Evans et al., 2003].

1.5.6.2 Off-Board Data Sources

A number of methods exist for computerised storage of spatial data. Geographical Information Systems (GIS) including ESRI ArcGIS [Environmental Systems Research Institute, 2015] and the open source GRASS [GRASS Development Team, 2015] system allow a human operator to manipulate large spatial datasets. GIS data is typically stored in either raster (array like) or vector (primitive based) forms, although support for point cloud data is now becoming available.

A more specialised robotics system that has promise is RoboEarth, a knowledge processing system for mobile robots that includes such features as data mining and visual feature classification [Waibel et al., 2011].

Specialised marine data is available from sources such as BLUElink, a joint venture between

the Bureau of Meteorology, Commonwealth Scientific and Industrial Research Organisation (CSIRO) and the Royal Australian Navy (RAN) to create a system for sharing current and predicted ocean data [Brassington et al., 2007]. Using this data, seven day predictions of ocean currents, temperature, salinity and sea level anomaly are available from the Bureau of Meteorology [Commonwealth of Australia, 2018].

Information on water depth and coastal extents can be more difficult to find. Spatial data for navigational purposes in Australian waters is only available from the Australian Hydrographic Office (AHO) as encrypted navigation charts. An alternative may be the use of charts by the National Oceanic and Atmospheric Administration (NOAA) which cover the United States and much of the Pacific Ocean. These charts are provided under a license that allows usage with minimal restriction.

Geosciences Australia was created in 2000 as a merger of two former Australian government geoscience and survey organisations. Tasked with providing geographic information to promote safety and the effective use of natural resources, this organisation provides spatial data, including bathymetry, under a number of licenses including Creative Commons [Geoscience Australia, 2017]. Unlike the charts created by the AHO, this data is intended for analysis rather than navigation, however it could be used for the generation of mission plans based on spatial criteria.

1.6 Summary and Proposed Approach

The previous section has provided an overview of existing approaches and applications for field robotic systems designed to travel on or underneath water, and the concepts required to guide and plan for them. These sections have introduced a number of specialised planning systems for tasks such as mission, coverage and path planning, as summarised in Table 1.3.

One of the more interesting concepts however is the symbolic planner which is capable of adapting to multiple different tasks simply by altering the domain and task descriptions. However, such general purpose planners cannot efficiently operate with spatial data. But

what if the spatial environment could be reduced to its fundamental underlying topology? Could such a system provide both the flexibility of generalised symbolic planning while still allowing plans to be informed by spatial information?

1.7 Aims

This thesis has the following aims;

- Examine the feasability of developing a planning system combining both spatial and symbolic data.
- Implement and demonstrate that this planner can effectively schedule missions.
- Evaluate the performance of the mission planner and scheduler for field robotic tasks.

Table 1.3: Summary of Planning Approaches

Type of Planning	Approach	Significant Implementations
Path	Rapidly exploring Random Tree (RRT)	Frank et al. [Frank et al., 2016]
	Lattice	Shah et al. [Shah et al., 2016]
	Interval planning	Benjamin et al. [Benjamin et al., 2006]
Coverage	TSP based lazy	Englot and Hover [Englot and Hover, 2013]
	Genetic algorithm	Ren at al. [Ren et al., 2008]
	Watershed Segmentation	Kleiner at al. [Kleiner et al., 2017]
Mission	Constraint Satisfication	McGann et al. [McGann et al., 2007]
	Symbolic Planning	Cashmore et al. [Cashmore et al., 2015].
	Symbolic planning with lifting	Munõz et al. [Muñoz et al., 2016]

1.8 Key research questions

- Can the effectiveness of deliberative symbolic planning be improved by the incorporation of spatial data?
- Can symbolic spatial planning be applied to the real world?
- Will this allow more effective utilisation of field robotic assets?

1.9 Research Methodology

This thesis examines the integration of spatial data with symbolic planning to leverage the available data resources for improved operation of field robotic systems. This is used to develop a planning system capable of both supervised and full autonomy.

The process of this development occurs in a number of stages, first is a review of existing approaches to the problem of planning in field robotics, including an examination of hardware and software platforms. This section identifies that the combination of the flexibility of symbolic planning combined with spatial data about the robots environment could improve the quality of robotic planning.

The next chapter examines how spatial data can be represented, and how this data can be compressed to identify the essential underlying information that can be used for informing planning. This section includes an overview and examination of the properties of several different algorithms for handling geometric data.

The third chapter uses the identified geometric algorithms and evaluates their effectiveness at producing high-level path plans. This is done be experiments on a combination of synthetic and real-world problems.

The fourth chapter evaluates the ability of high-level symbolic planning algorithms to combine both path and mission planning in a single operation. This chapter will also test experimentally the scalability of these systems.

Chapter five combines both the topological compression of spatial data with the previously

tested symbolic planners in a planning system based on the mission planning stack created for the TopCat Autonomous Surface Vessel. This system is tested in a simulated environment based on maritime rescue.

1.10 Research Significance

High level control systems for maritime mission scheduling have been developed, but they are typically either reactive in nature - preventing long term planning - or lack spatial awareness - making them unable to operate autonomously in a complex environment.

This thesis describes the development of a system capable of performing planning with both symbolic and spatial data, scheduling software for controlling the robot and classification systems allowing data to be updated in real time. This will improve mission capabilities by allowing spatial data to be symbolically evaluated during mission planning, execution and evaluation phases.

While the example implementation was originally intended for an AUV, its planning can be applied to a variety of field robotic types. The field demonstration uses a maritime ASV to demonstrate its operation.

By combining knowledge of the environment with information on the vehicle's performance encoded in the domain model and mission goals in the task model, a solution can be reached that will maximise the return in data for a mission.

1.11 Contributions

This thesis proposes and investigates an altered model of planning for field robotics that changes the normal separation between the mission and path planning layers. This provides a number of contributions to knowledge. These have been organised into two groups, a list of the contributions to the theory of mission planning, and a list showing the specific novel contributions;

1.11.1 Research

- An exploration of methods for geometry representation and belief compression, and the selection of methods for skeletonisation, and segmentation that could be applied to the belief compression problem in external spaces. High level planning systems are limited in the complexity of the search space that they can explore. To allow complex spatial relationships to be incorporated into their search domain, these spatial relationships need to be reduced to a form that contains the essential information, a process called belief compression. The commonly used approaches for performing this compression are either only generalisable to $\mathbb{R}^2$, or limited in the type of environment that they are applicable to. Chapter 2 explores these possible systems and selects systems from biomedical and computer vision for application to the robotic mission planning domain.

- A direct comparison of both the robustness and effectiveness of the selected belief compression methods. If the new planning model is to generate effective plans then this belief compression system must be able to inform the task of path planning. Since the selected belief compression methods are not used in the robotics space, no overall quantitative comparison of their effectiveness is available. Chapter 3 includes experiments testing the compression methods robustness to affine transformations, and their effectiveness at producing a simplified model that can be used to perform the high-level planning task on both synthetic and real-world derived datasets.

- An evaluation of the PDDL based symbolic planners that are currently used in ROS-based robots to find their applicability to planning in the modified mission planning model. Symbolic planning systems are extremely flexible at producing plans based on the known state of the system. However, while some spatial based test domains exist, they typically use unity cost to represent the movement between such spaces. Chapter 4 in particular tests these planning systems for the following properties that would be required for the modified mission planning system to be effective;

  - Scalability under different sizes of spatial domain, number of goals, and complexity of preconditions

- Planning time
- Effectiveness of generated plans with both symmetric and asymmetric action costs
- Applicability of search and heuristic types
- The development and implementation of a planning system for an ASV that demonstrates the effectiveness of the proposed planning model.

1.11.2 Engineering

- Chapter 5 uses an ASV simulation that incorporates planning and control systems developed for the TopCat ASV running the ROS system to test the effectiveness of the planning model at performing planning in spatial environments derived from real-world data. These experiments show that the planning system outperforms a greedy system at mission planning when the overall order of actions is important.

1.12 Publications

Jonathan Wheare, Andrew Lammas, Karl Sammut, 2018, *Towards the Generation of Mission Plans for Operation of Autonomous Marine Vehicles in Confined Areas* (Accepted for Publication)

1.12.1 Conference Presentations

Jonathan Wheare, A/Prof Karl Sammut, Dr. Andrew Lammas, Tenzin Crouch, Dr. Graziela Miot Da Silva (2015) *An Autonomous Surface Vessel for coastal environmental monitoring or: Getting a robot to do your field work*.

1.12.2 Competition Journal Papers

Jonathan Wheare, Bradley Donnely, Russell Peake, Thomas Arbon, Matthew Anderson, Dr. Sherry Randhawa, Assoc. Professor Karl Sammut (2014) *Autonomus Ground Vehicle Competition 2014 - Flinders University Team Redback*

Assoc. Professor Karl Sammut, Jonathan Wheare, Dr. Andrew Lammas, Mr Richard Bowyer, Matthew Anderson, Thomas Arbon, Bradley Donnelly, Russell Peake, Tenzin Crouch, Rowan Pivetta, Joshua Renfrey, Tobias Wooldridge, Scott Stevens, Andrew Webb, Dr. Alexander Forrest, James Keane, Harry Hubbert, Reuben Kent, Supun Randeni Pathiranachchilage (2014) *Maritime RobotX journal paper - Flinders University / Australian Maritime College - Team Topcat* `http://robotx.org/files/Flinders%20University-%20Australian%20Maritime%20College%20Maritime%20RobotX%20journal.pdf`

Jonathan Wheare, Bradley Donnelly, Russell Peake, Thomas Arbon, Matthew Anderson, Rowan Pivetta, Dr. Sherry Randhawa, Assoc. Professor Karl Sammut, Dr. Andrew Lammas (2014) *Autonomus Ground Vehicle Competition 2014 - Flinders University Team Redback*

Tenzin Crouch, Allan Mankavil, Andrew Webb, Bradley Donnelly, Derrick Kickel, Patrick Kloasen, James Armitage, Rowan Pivetta, Joshua Renfrey, Jonathan Wheare, Lee-Ying Wu, Keelan Burns, Michael Cadzow, Dr. Nasser Asgari, Mr. Richard Bowyer, Dr. Andrew Lammas, Dr. Jimmy Li, Dr. Sherry Randhawa, Assoc. Professor Karl Sammut (2013) *Autonomus Ground Vehicle Competition 2013 - Flinders University Team Redback*

1.12.3 Competition Presentations

Autonomous Ground Vehicle Challenge (AGVC) 2013 - lead presenter

Maritime RobotX 2014 - lead and final presenter

Autonomous Ground Vehicle Challenge (AGVC) 2014 - lead presenter

Maritime RobotX 2016 - presenter

1.13 Outline

- Chapter 1 explores the application of planning to field robots. This includes a general introduction to field robots, brief coverage of field robotic platforms, and information on software and approaches used for robotic planning.
- Chapter 2 covers the existing techniques for the processing of spatial, geometric and topological data for use with symbolic planning. Particular detail will be given to the generation of thin structures called *skeletons* derived from geometric shapes.
- Chapter 3 investigates the effectiveness of skeletisation algorithms at producing structures that can be used for belief compression. These algorithms will be evaluated for their effectiveness in producing compressed belief spaces that can be used to inform spatial planning.
- Chapter 4 examines the effectiveness of domain independent planners at performing high level scheduling tasks using spatial information. These systems will be evaluated in a variety of environments.
- Chapter 5 covers the operation of the TopCat ASV as a field robotic platform. This will demonstrate that the planning systems covered in this thesis can be used to plan missions for a field robotic platform.
- Chapter 6 will summarise the findings of this thesis and propose future work.

In addition, appendices give an overview of the operation of symbolic planning, the available simulation platforms and information on the handling of spatial data.

Chapter 2

Belief Compression for Symbolic Planning with Topology

2.1 Introduction

As covered in the introductory chapter, robots are increasing in capability to the point where they are capable of storing and manipulating significant amounts of information. This increasing capability will result in them being able to perform increasingly complex missions, potentially including the achievement of multiple sub-goals. If such systems are to be used successfully, not only will they need to effectively perform their assigned tasks, but the human operators will also need an efficient way to specify what tasks are to be achieved. The use of general purpose planners allows atomic, well defined actions to be combined to produce plans that are both effective and understandable.

For planning with complex actions, symbolic planning shows promise. This technique uses values and predicates to represent *belief* - a robot's representation of its environment, combined with a set of predefined actions allowing these values to be transformed. A symbolic planner uses a search algorithm [Russell and Norvig, 2010, p66] to find a sequence of actions that will transform its current belief into a desired goal state [Fikes and Nilsson, 1972]. The use of such a planning system has the advantage that it can use its information on the robot's current state, and the predicted effect of actions, to produce complex plans that meet

its goals, rather than taking a single action at a time based on its immediate state.

The original implementation of symbolic planning used with Shakey allowed such searching to be performed by representing its work area as a series of rooms with connecting doorways. Mobile objects within these rooms were represented by corresponding objects within the robot's belief space, allowing courses of actions to be generated to transform the environment. Shakey's plan was derived from the robots' belief state at the start of the run, which could cause plan execution to fail if the environment changed during execution due to an external action. Shakey or its operators could update this belief, but would only be creating new static positions of its movable objects [Brooks, 1991a].

At the core of the problem of robotic planning in spatial environments is thus how can the environment be measured and turned into a model that can be used for planning? Existing solutions for incorporating spatial data into planning solutions typically use semantic attachment[3], either performing a path generation operation between all possible combinations of goals, or incorporating a separate path planning module alongside the mission planner. Could the spatial environment instead be modelled directly in the planning domain?

Since classical planning systems are based around the concept of searching for possible arrangements of actions in a graph representing the state space of the system, a representation of the spatial environment can be merged into such a representation. An example of such an approach is the work of Belauer and Bouzid who demonstrated the representation of spatial data by decomposing logical areas into symbolic trees [Belouaer et al., 2010], but this is a specialised solution for an environment with a regular geometry. An algorithm for a field robotic platform would need to support the irregularly shaped spaces that would be encountered. This requires a more general solution to the problem of belief compression that supports such spaces.

The first half of this chapter will examine the areas of digital geometry and topology with a view to their application in preparing efficiently compressed spatial data for symbolic planning, while the second half of the chapter will cover the implementation of a selected group of algorithms for belief compression in robotic planning.

[3]More information on semantic attachment can be found in Section 1.5.4

2.2 Representation of Geometric Objects

Any model of the physical world stored within the digital world of a computer must be a summary of the continuous and finely divisible real world that is sampled into the domain of the discrete and digital. As such, a method is required to summarise and store geometric information in a way that it can be manipulated by a computer.

Geometric objects can be stored as combinations of solids, surfaces or using a grid based metric. Each of these forms requires a different set of assumptions that can also limit how volumes can be generated and manipulated.

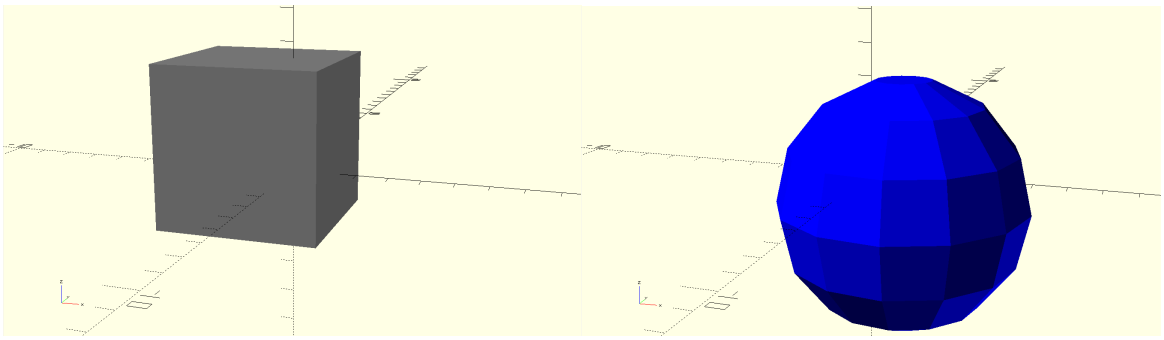
2.2.1 Representation as Solids

A possible solution to the storage problem is the use of simple and platonic[4] solids to represent geometric information. This is the form of representation that was used in the original Shakey planning system - the objects shown in photos of Shakey are primarily prisms, so a platonic solid based system would accurately reflect reality. Solids can be stored as a combination of type, pose and scale information. However, production of this data from sensor data can be more complicated. A classification system must pre-process the sensor data to find and infer the required shape before the internal representation can be updated.

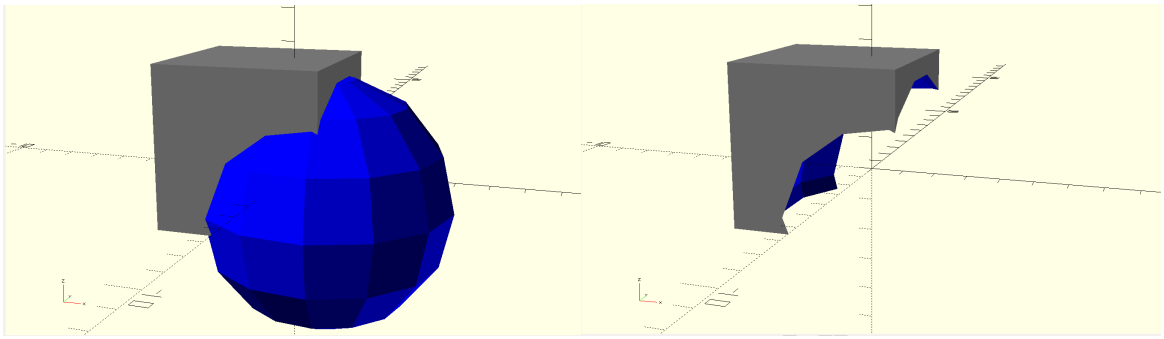
More complex shapes can be constructed by creating sets of solids by the union, difference and intersection of solid shapes, a technique referred to as *constructive solid geometry*. In this representation the final shape is stored as a recipe of the operations required to produce the desired result [Requicha and Rossignac, 1992]. An example of Constructive Solid Geometry can be seen in Figure 2.1

Besides use in Computer Aided Design (CAD), this form of representation has found application in the form of *tetrahedral meshes* - complex volumes formed of the union of tetrahedrons, each in turn described as sets of four vertices. Software exists to generate such meshes using Delaunay triangulation [Si, 2015, Boissonnat et al., 2000], and they can be

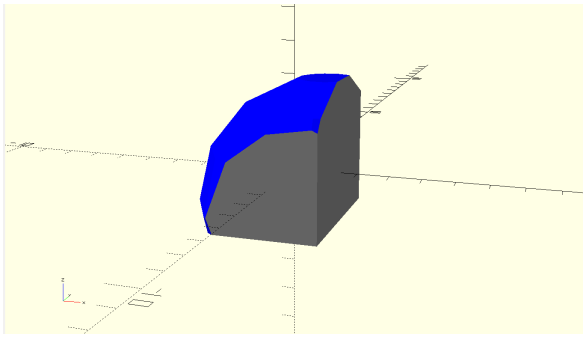
[4] A platonic solid is a regular polyhedron, the faces of which are of equal size and shape [Helicon Publishing, 2005]. The cube and tetrahedron are included in the set of platonic solids.



(a) (b)



(c) (d)



(e)

Figure 2.1: Construction of a solid shape using Constructive Solid Geometry and set operations. (a) a cube C (b) a sphere S (c) union of the sphere and cube $S \cup C$ (d) subtraction of the sphere from the cube $C - S$ (e) intersection of the cube and sphere $S \cap C$.

used for the construction of Reeb graphs [Doraiswamy and Natarajan, 2009].

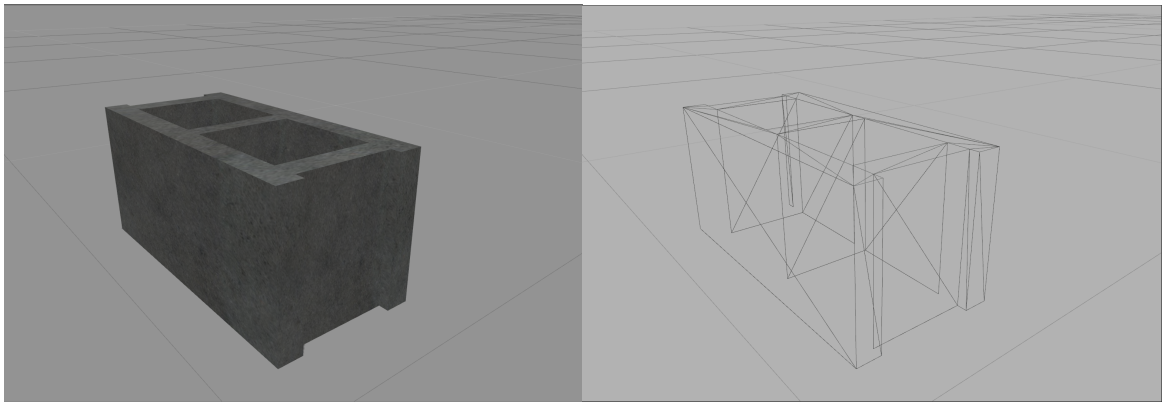
2.2.2 Representation as a Surface

An area or volume can also be described by a fully enclosing surface. Whatever is inside the surface is part of the shape. This form of shape is commonly used in Geographical Informations Systems (GIS) where a shape in $\mathbb{R}^2$ can be represented as a *polygon* - a collection of line segments in a closed loop.

In $\mathbb{R}^3$, a surface can be described as a triangular mesh. This representation is commonly used in computer graphics, but a mesh that is completely enclosing, referred to as ”water-tight”, can be used to describe a volume. The StereoLithography (STL) file format commonly used in rapid fabrication uses this format for encoding volumetric shapes [Ennex Corporation, 2015]. An example of representing a solid as a surface can be seen in Figure 2.2

2.2.3 Representation as a Grid

A regular grid can be used to store information about the shape of objects by marking the grid cells that correspond to the locations that are occupied by an object. This method is popular in autonomous robotics since it can be easily updated from sensor data, adding and



(a) (b)

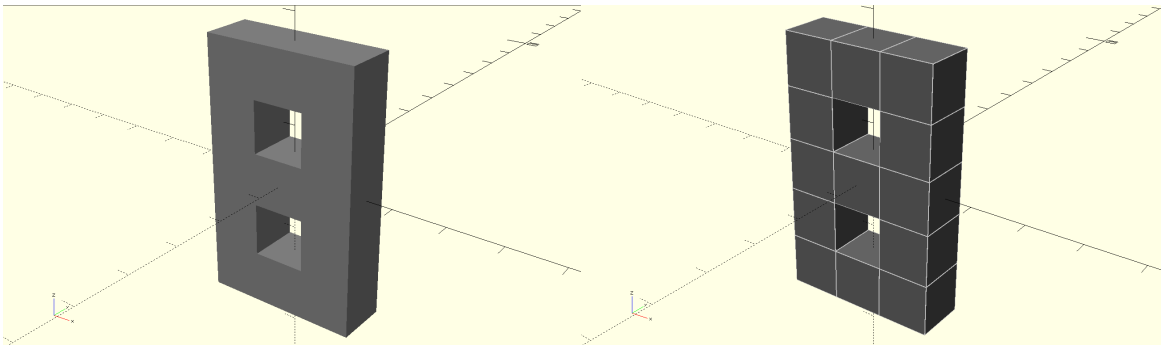
Figure 2.2: Representation of a volume by describing it’s surface. (a) Model of a cinderblock (b) Underlying triangular mesh. The mesh describes the surface of the volume. Model sourced from the gazebo model repository [Koenig, 2018]. Model license: CC-By

clearing obstacle cells as sensor data is received. The grid may use a variety of methods for storing the occupancy of a cell as either a simple boolean value, some measure of the number of returns seen from a cell, or a probability of occupancy. The storage efficiency of this method is dependent upon the homogeneity and amount of information encoded, since the memory allocated is dependent on the extent of the area rather than the data that is encoded.

Storage efficiency can be improved by attempting to remove unused areas. A popular approach is the use of k-dimensional trees, recursive tree structures that only create leaves for areas that contain data, to store the occupancy information. Such data structures have found application in robotic motion planning [Yahja et al., 1998].

2.3 Reduction of Volumes

While solids and surfaces have found application in robotics, the grid representation is the most popular. The regular nature of a grid simplifies tasks such as mapping and motion planning. However, there is likely still too much information to allow efficient manipulation of the data. A possible solution to the problem of complexity is to use techniques that reduce the dimensionality of the data while still maintaining properties about relationships. Dimensionality reduction can be performed by multiple techniques based on the type of data available and the property that is to be maintained.



(a) (b)

Figure 2.3: Representation of a volume using a regular grid. (a) Volume to be represented (b) Set of grid cubes. The volume is the union of the individual cubes.

Eight methods for performing such a reduction were examined. These methods can be broken down into four general categories;

- Methods that are based on finding a geometric median.
- Methods based on topology.
- Methods that use clustering approaches.
- Methods that use landmarks.

These approaches are summarised in Table 2.1.

Table 2.1: Methods for reduction of volumes

Category	Name	Description
Geometric	Voronoi Diagram	The locus of points midway between pairs of points
	Medial Axis Transform	Grid based method that iteratively removes the outermost set of elements until the remainder is thin.
	Delaunay Triangulation	Decomposition into triangles where no triangle contains another vertex
	Reeb Graph	The evolution of the level sets in a solid.
Topological	Lee et al. [Lee et al., 1994] thinning	Grid based removal of simple points from the outermost elements. Simple points are identified using the euler number of the 3x3x3 neighbourhood and connectivity tests.
	Palágyi and Kuba. [Palágyi and Kuba, 1999] thinning	Grid based removal of simple points from the outermost elements. Simple points are identified using a set of templates.
Clustering	Spectral Clustering	Grouping of elements into clusters based on similarity.
Landmark	Topological Landmarks	Construction of a graph based on features of the environment found while exploring.

2.3.1 Reeb Graph

The Reeb Graph of an object is created by converting the volume containing the object into a sequence of two-dimensional sections, segmenting the object within each section, and comparing the connectivity of the object's segments between these sections. The result is a graph describing how the object changes over the sequence of segments, the simplest case of which is to take regular slices across a single axis. However, this configuration is highly susceptible to changes in rotation of the volume. An example of constructing a Reeb graph can be seen in Figure 2.4.

Reeb graphs were used by Aleotti and Caselli for topological segmentation of an object for grasp planning [Aleotti and Caselli, 2011], unlike axis based Reeb graphs they used the evolution of the object from a central point to characterise the object, thus avoiding the rotational dependence of axis based graphs.

Doraiswamy and Natarajan provided an efficient method for calculating such graphs from tetrahedral meshes [Doraiswamy and Natarajan, 2009]. Finally, Garcia and Gonzalez de Santos used a graph representation of topology from a Reeb graph to explore spaces [Garcia and Gonzalez de Santos, 2004].

2.3.2 Voronoi Graph and the Medial Axis Transform

The Voronoi diagram is constructed from a set of points by finding the locus that is equidistant between neighbouring points. In $\mathbb{R}^2$ this locus is a thin line, while in $\mathbb{R}^3$ it forms a surface. In the case where a grid based metric rather than a set of points exists, the Voronoi locus can be approximated using the medial axis transform, starting at the outside of a shape and removing a layer at time until only a single thickness of points is left. These diagrams can be used to decompose an area or volume into cells based on the distance from the seed or generation points. If the seeds correspond to the exterior shell of obstacles to be avoided then the generated cell borders can be used as paths [Rao, 1993]. But, navigation on the Voronoi graph can be as simple as a control system that keep the robot equidistant from obstacles while moving in the direction of its goal.

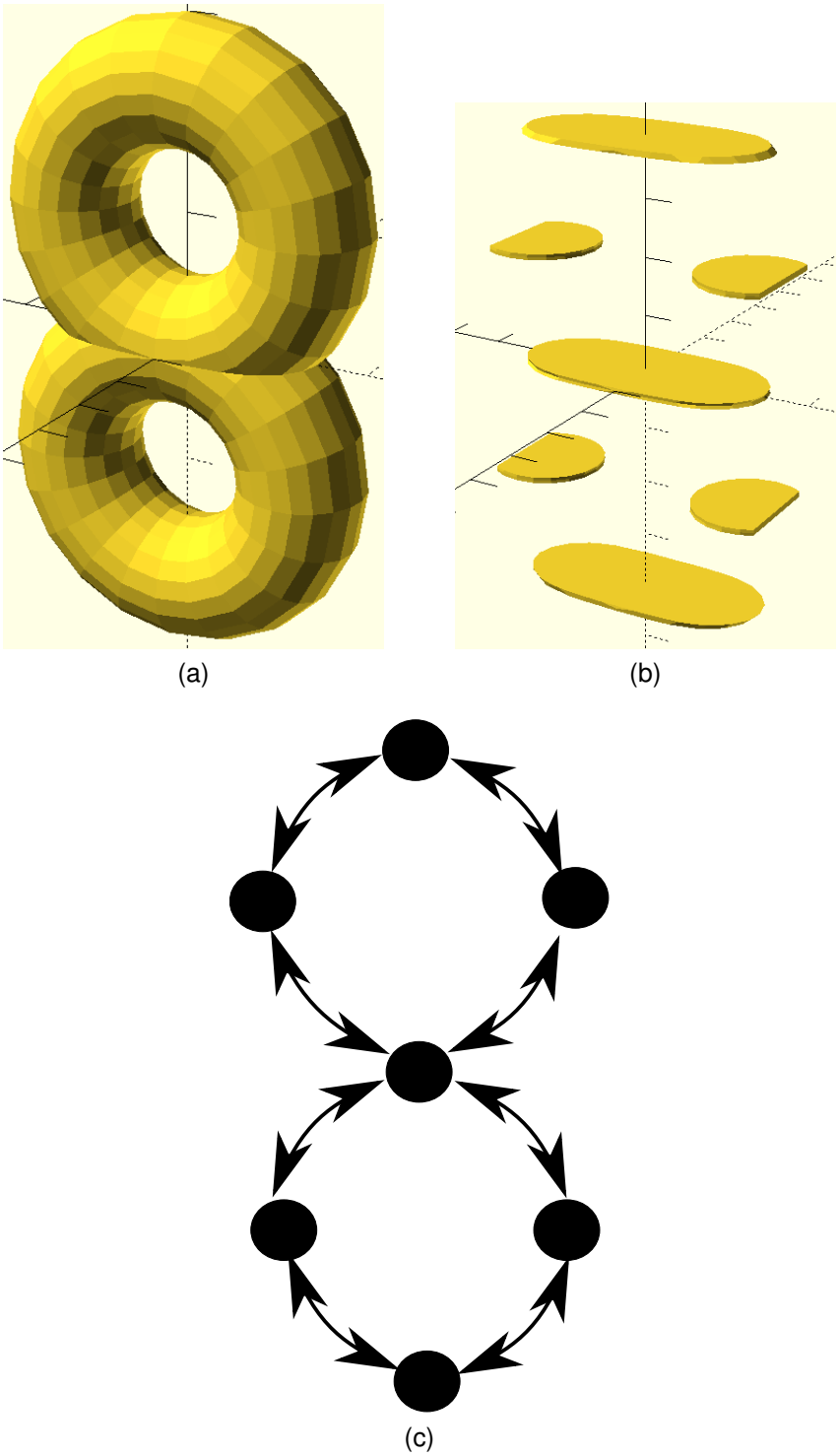


Figure 2.4: Construction of a Reeb graph, a graph representing the evolution of the level sets, from a volume. (a) Input volume consisting of a pair of toruses (b) level sets sliced along the z-axis (c) resultant graph.

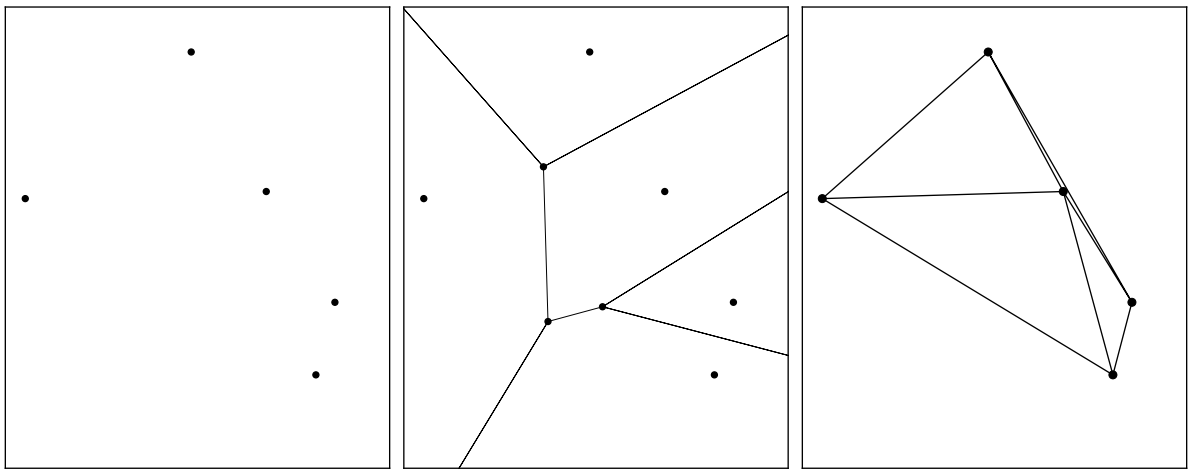


Figure 2.5: Construction of Voronoi graph and Delaunay triangulation from vertices. (a) input vertices (b) corresponding Voronoi graph (c) corresponding Delaunay triangulation. Images generated using the Python bindings to the Triangle library [Rufat, 2017] and Matplotlib [Hunter, 2007].

Thrun used the Voronoi graph to plan robotic motion by generating a graph structure of the robot's immediate surroundings. These local segments were then merged together to form a global graph that described the robot's observed environment. In addition to the graph generation, Thrun also segmented the traversable areas by the construction of critical lines in the areas where obstacles approached most closely [Thrun, 1998].

Sud et al. used a different recursive approach, generating first and second order Voronoi diagrams to create a Multi-agent Navigation Graph (MaNG) [Sud et al., 2007]. This algorithm uses both obstacles and agents as seed points to create paths for multiple agents cooperating in an area. Sud et al. reported that large numbers of agents could be routed in real time through spaces despite the existence of dynamic obstacles.

In R^3, Voronoi based algorithms have been used for identifying and characterising the branching of blood vessels in medical imaging systems. The Vascular Modelling Toolkit (VMTK) is a software package designed to enable the performance of such characterisation tasks [Antiga et al., 2008]. This library can characterise the topology, branching points, and the angle of separation of blood vessels [Antiga and Steinman, 2004]. As mentioned earlier in the section, the Voronoi diagram of a shape in $\mathbb{R}^3$ is not thin, but VMTK uses the Voronoi diagram sheets as a first step to find the centreline.

In addition to its applications in robotic planning, the Voronoi diagram has found application

in other areas where coverage is required to be found. Bruck et al. used the medial axis transform for designing the deployment of hierarchical sensor networks [Bruck et al., 2007]. By identifying the medial axis of a deployment area, a backbone can be designed that will allow efficient communications with individual sensor nodes. Choset describes the use of first and second order generalised Voronoi diagrams for sensor network design [Choset and Burdick, 1995]. Notably this algorithm solves the issue of thinness in $\mathbb{R}^3$ by first generating the Voronoi surface and then finding the Voronoi diagram of that surface.

The skeleton created by the medial axis transform is dependent on the geometry of the object, and if the distance of each point to the exterior is known for the iteration number, a spherical volume of that radius can be created at each element in the locus of the skeleton. The sum of all these volumes is a reconstruction of the original volume. This technique, referred to as the 'power crust' algorithm, is used to repair and reconstruct shapes from noisy surface data [Amenta et al., 2001].

2.3.3 Other Approaches

In addition to to the previously covered approaches, two more are of interest, the Delaunay Triangulation and Machine Learning.

Given a set of points, the Delaunay triangulation of those points is a set of triangles constructed such that each point is at least one vertex, and a circle constructed from the vertices of any triangle does not contain any other vertex [Frey and George, 2000]. Notably, the Delaunay triangulation is the dual of the Voronoi diagram - as such, once one has been constructed it can be used to produce the other. Such triangulations have been proposed as a source for vehicle motion planning, including the work by Pêtrés et al. who applied their fast marching algorithm, a form of breadth-first search, to both regular grids and Delaunay meshes [Pêtrès et al., 2007]. Unlike methods based on regular grids, triangulations allow plans to be generated at multiple possible resolutions. This potentially allows more efficient planning in sparse environments.

Clustering is a type of machine learning where points are grouped based on a property. Brunskill et al. used the AdaBoost machine learning algorithm to cluster and segment indoor

areas from lidar data [Brunskill et al., 2007]. This algorithm was later used as part of a navigation system that used natural language commands to direct a robot [Kollar et al., 2010].

2.3.4 Topological Thinning

Topological thinning operates in a similar method to the medial axis transform - removing a layer of points at a time until only the skeleton is left. The primary difference is that before a point is removed, it is tested to see if it is *simple* - a point which if removed will not change the topology of an object. A definition of such simple points in $\mathbb{R}^n, n = [2, 3, 4]$ is provided by Couprie and Bertrand [Couprie and Bertrand, 2009]. Unlike the medial axis transform, topological thinning algorithms always produce a single pixel wide skeleton [Lee et al., 1994]. In an ideal algorithm, this skeleton is dependent only on the topology and not the geometry of the input shape. Information on the implementation of Thinning algorithms, including figures showing their effects, will be covered in Section 2.4.2.

Topological skeletons have been used in medical applications to find the routes through the branching air passages in the lungs, and in unrolling sections of intestine. Kiraly et al. used a hybrid skeletisation technique to identify the structure of the airways in patient's lungs using data captured by a MultiDetector Computed-Tomography (MDCT) scanner [Kiraly et al., 2004]. They demonstrated that a volumetric representation of the human bronchial tubes can be decomposed to produce a tree representation that allows a path plan to be generated.

Cornea suggested that skeletons can be used for navigation [Cornea et al., 2006]. Skeletisation has also been used for producing seeds for the optimisation of flight paths for Unmanned Air Vehicles (UAV) [Sun and Tsung-Ying, 2008].

If this technique is used to generate a graph based representation of the possible paths around the obstacles, the topology of the environment could be characterised. Such a topological graph would allow paths of specified homotopy to be generated between arbitrary points in the environment. However, there are other approaches to planning while considering the topology of a space.

2.3.5 Topological Landmarks

In addition to the previously mentioned approaches for producing topological information from spaces, it is also possible to produce graphs from the changes or obstacles in the space. Wong and McDonald used a cellular decomposition based on Morse functions to create a topological graph of the environment for efficient coverage planning. They used this to demonstrate coverage of a space for a simulated vacuum cleaner robot [Wong and MacDonald, 2003].

Acar and Choset developed a system for exploring the topology of a space using Morse decompositions. They demonstrated the application of this method to practical robotics by incrementally generating a graph representing the topological changes [Acar and Choset, 2002]. An example of such a construction can be seen in Figure 2.6. In (a), a robot is using an alternating path to explore it's environment from left to right. When the edge of an obstacle is detected, a critical point is created as shown in (b). The resulting graph in (c) is the connection of these critical points.

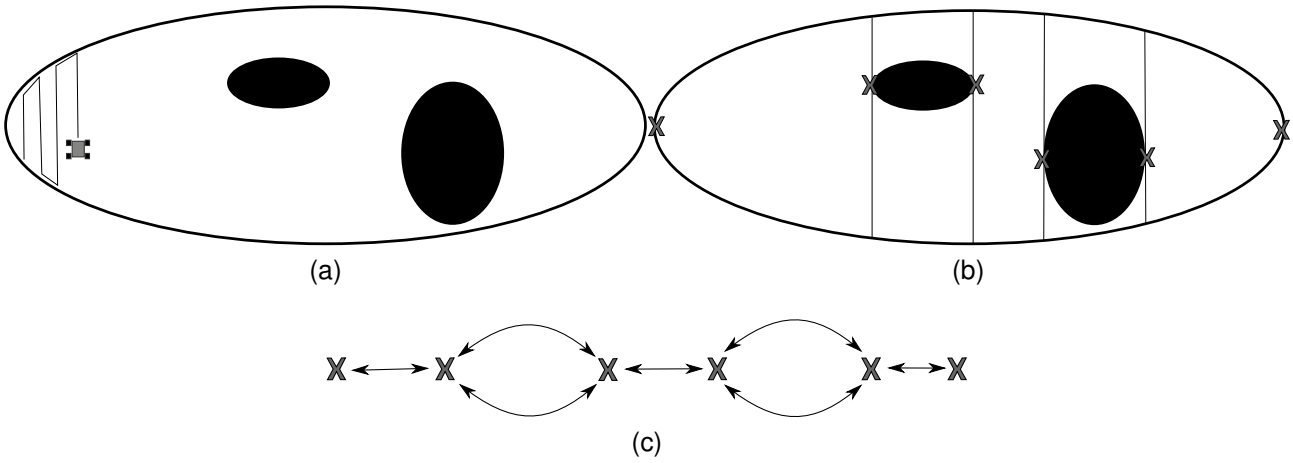


Figure 2.6: Construction of cell decomposition using Morse decomposition. (a) Robot exploring a space using an alternating path (b) Critical points found by the robot. These points are where the edges of obstacles are encountered. (c) Resulting graph of decomposed cells.

2.4 Topological Segmentation with Skeletisation for Mission Planning

Algorithms for planning and navigation such as Thrun's have used graphs derived from the local environment [Thrun, 1998], but the generated graph only reflects the topology or geometry of the volume that the algorithm is executed upon. If the planning system is to be executed globally, then like Kiraly et. al. [Kiraly et al., 2004]., the thinning algorithm must be executed on the global rather than the local volume. The Voronoi and medial axis methods are not suited to this task due to their geometric sensitivity.

2.4.1 Proposal

What is required is an algorithm that can produce a graph that informs the planning system based on the spatial information, while keeping a representation that is compact enough to allow efficient planning.

Generation of such a compact representation of the spatial information could be done by considering the topology of the space. Rather than constraining the trajectory as performed by Bhattacharya, Likhachev and Kumar [Bhattacharya et al., 2012], the robot's space of all possible motions could be decomposed into cells where all possible trajectories are homotopic. With such a decomposition all possible trajectories within a cell, could be continuously deformed from one to another [Hatcher, 2002], then, given the existence of an optimal trajectory within a cell, finding that exact optimal trajectory can be neglected for the high level planning tasks. Instead, the homotopic volume can be treated as an abstract object within the domain independent planner, with the creation of an optimal trajectory left to a lower level algorithm. This lower level algorithm can also be more efficient since it will only be required to search the cells identified by the high level algorithm, rather than searching all possible cells.

Since the branching of the skeleton is dependent on the topology of the space, the number of segments will be dependent on the number of obstacles to be considered, rather than the

geometric size of the volume or obstacles. By analysis of the skeletal locus, the topology of the volume can be inferred: a volume with unchanging topology will produce a skeleton that is either a single voxel or a curve containing two endpoints. The information from this analysis can be encoded into a suitable symbolic representation, allowing scale independent planning can be performed using a symbolic planning engine. The envisioned planning system would thus support the creation of domains capable of supporting temporal planning using spatial data while minimising the effect of the combinatorial explosion in complexity that would occur with directly encoding the available spatial data into the domain.

To produce a graph that reflects the topology of the volume, the proposal is to segment the volume in a manner similar to the planning graph. The graph is composed of nodes that are used to represent the available working areas connected by edges that represent the connectivity. To allow the planning graph to be used with the input volume, the volume must be segmented into a similar group of volumes. The proposed method is;

- reduce the volume to its skeleton,
- analyse the topology of the resulting structure,
- reconstruct the volume in segmented form.

The following sections will cover the properties of these method in more detail.

2.4.2 Thinning

Thinning is a process where parts of a volume are removed until a minimal result is left that still maintains a property of the input volume. While there are similarities in operation between Lee et al. thinning [Lee et al., 1994], Palágyi and Kuba [Palágyi and Kuba, 1999] thinning, and the medial axis transform - all these algorithms remove voxels from the exterior of a volume until the result is a single voxel wide - there are significant differences in the result and robustness of the algortithms.

2.4.2.1 Medial Axis Transform

In the case of the medial axis transform, the property that is maintained is the distance from the edges of the volume. The algorithm works on each cardinal direction in turn, removing a layer of voxels from the surface of the volume in that direction. Since the process is incremental, the result is those voxels that are equidistant along the Chebyshev distance from the surface will be retained. The transform differs from the Voronoi Diagram in that the loci of the Voronoi graph lie along the Euclidian distance from the seed points. Figure 2.7 shows the effect of applying the algorithm on two basic shapes. The square is reduced to a set of lines which lie equidistant to the edges, while removing a single voxel at the centre of the square results in a shape that contains a single loop, this shows that despite its geometric nature, some topological information is still encoded in the graph. Figure 2.8 shows the process of generating the result over twelve iterations.

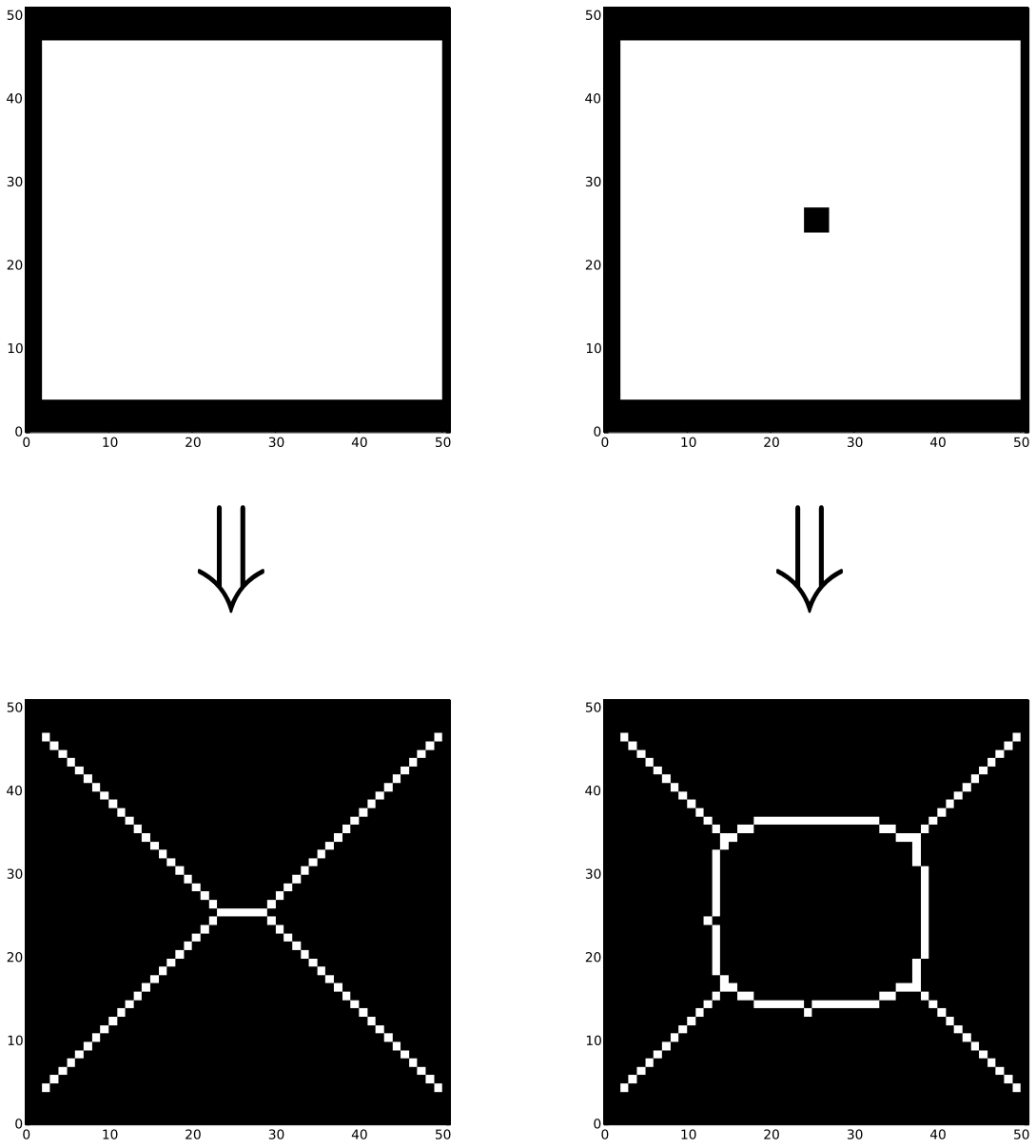
An algorithm for generating the medial axis can be seen in Algorithm 1. This algorithm uses the convention from image processing that *foreground* elements are those that are set, while *background* elements are those that are clear.

2.4.2.2 Thinning using the Lee, Kashyap and Chu Algorithm

The Lee et al. algorithm ("Lee skeleton") can be considered a derivative of the medial axis transform in that it uses the same iterative method to select foreground voxels for removal. Before the voxels are removed they are tested to see if their removal will alter the topology of the 3x3x3 neighbourhood that surrounds them. These tests are;

- counting the number of neighbours of the central voxel,
- calculating the Euler characteristic of the neighbourhood,
- testing the point for simplicity.

The neighbour count test simply totals the number of foreground voxels in the neighbourhood. If the number is two or less, the voxel is either standalone, the middle of a branch, or an endpoint. Thus if the voxel is removed, a change in the connectivity of the volume will



(a) (b)

Figure 2.7: The medial axis transform of an input with (a) a square of pixels (b) the pixels with the removal of a 3x3 section.

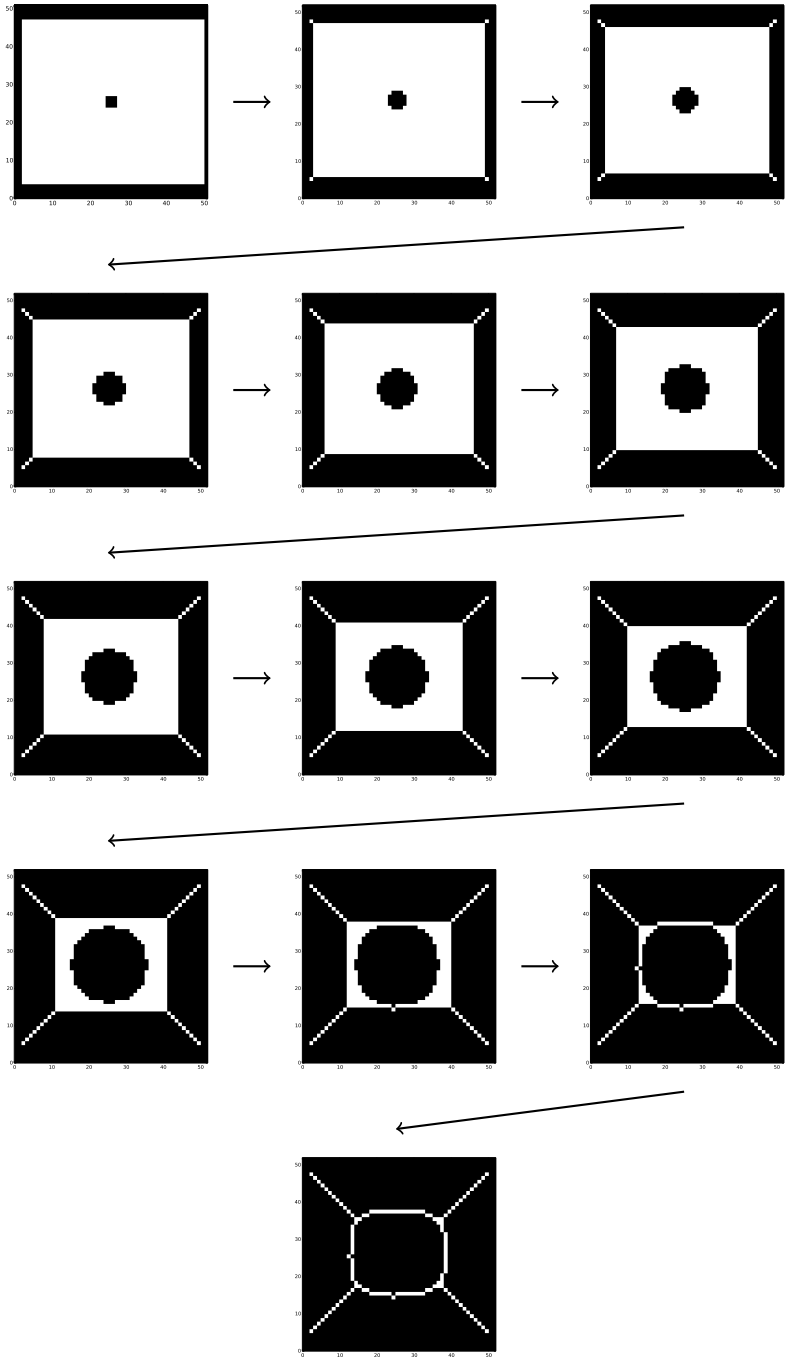


Figure 2.8: Evolution of the medial axis algorithm on the shape from Figure 2.7 (b). Each sub-image represents a single iteration of the algorithm.

Algorithm 1 Medial Axis Transform

$iteration \leftarrow 0$
$i \leftarrow 0$
$directions \leftarrow [north, south, east, west, up, down]$
while $number\_of\_changes > 0$ **do**
  $removable \leftarrow \emptyset$ ▷ The set of all removable voxels
  for $direction\ in\ directions$ **do** ▷ once for each cardinal direction
    for $voxel \in forground$ **do**
      if $(voxel[+direction] \in foreground) \cap (voxel[-direction]\ \in background)$ **then**
        $removable \leftarrow voxel \cup removable$
        $number\_of\_changes = number\_of\_changes + 1$
      end if
    end for
  end for
  $voxels[removable] \leftarrow 0$ ▷ All removable voxels are set to background simultaneously
  $i \leftarrow i + 1$
  $iteration[removable] \leftarrow i$
end while
return $voxels, iterations$

occur. Voxels with three or more neighbours continue to the next test.

The second test estimates if the removal of the voxel will cause a change to the volume's Euler characteristic. Since the characteristic of a shape is equal to the sum of the characteristics of its components, if the local neighbourhood has a characteristic of zero, removing the voxel will not cause the characteristic of the overall shape to change its value. In the Lee et al. algorithm this value is calculated for the 3x3x3 neighbourhood by summing the characteristics of the 8 2x2x2 neighbourhoods that compose it. These individual characteristics are calculated using a lookup table.

The final test is performed by finding the number of connected components in the neighbourhood if the voxel is removed. Since the voxel is adjacent to all its neighbours, if more than one component is found, the removal of the voxel will cause two parts of the volume to become disconnected.

Algorithm 2 is the algorithm for performing the skeletonisation, while Algorithms 3, 4, and 5 are the tests for preserving topology. Lee et. al. uses a subiteration method for labeling the local neighbourhood as shown in Algorithm 5. The version of this algorithm used in this thesis uses the labeling function from the Scikit Image toolbox [scikit-image development team, 2015a]. Figure 2.9 shows the effect of applying the algorithm to a simple box shape,

and the same with a single voxel removed. Figure 2.10 shows the process of the algorithm on the second shape as the iterations are performed. Both of these results notably differ from the medial axis in that it lacks the four branches at the corners of the images.

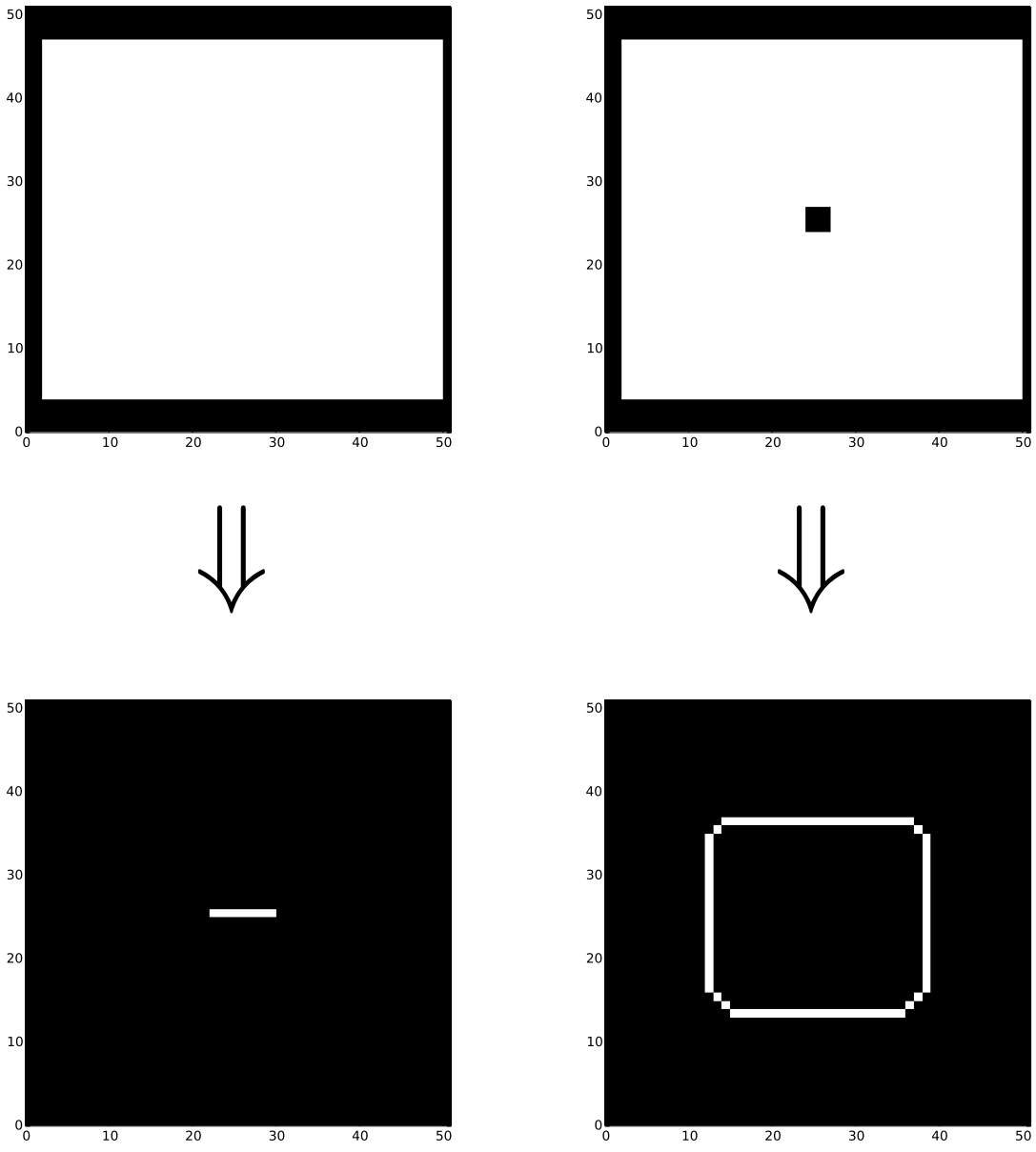


Figure 2.9: Skeletisation with Lee et al. of an input with (a) a square of pixels (b) the pixels with the removal of a 3x3 section.

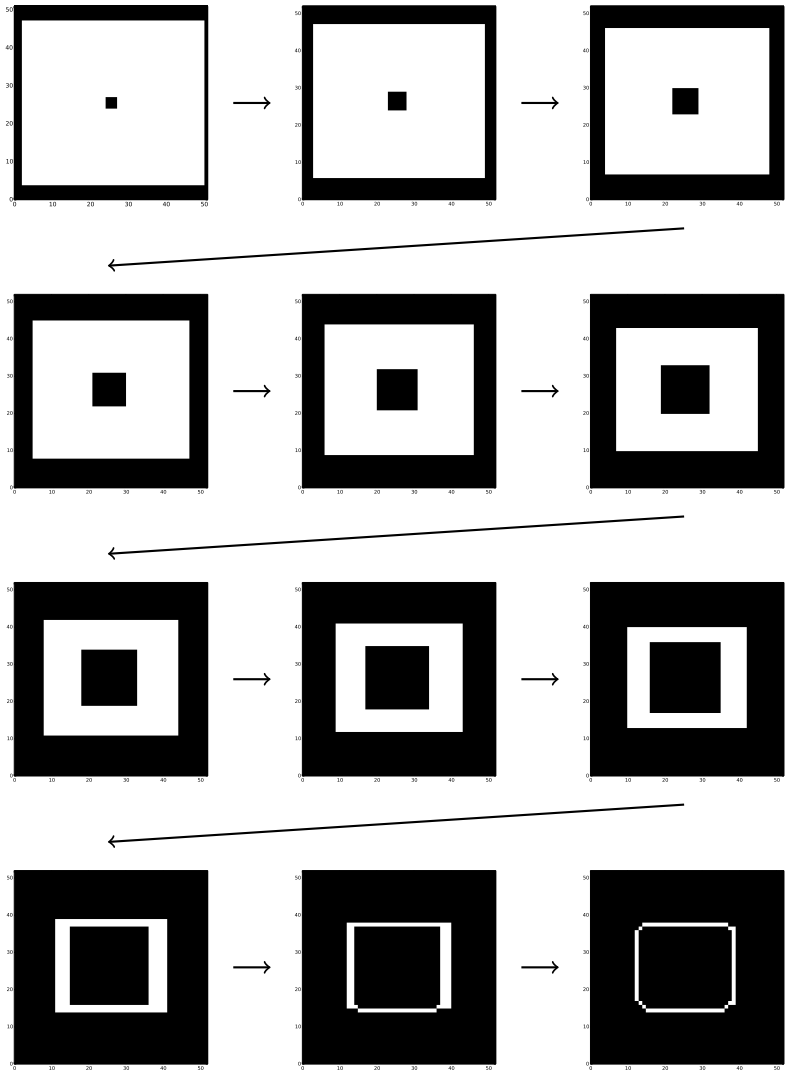


Figure 2.10: Evolution of the Lee et al. algorithm on the shape from Figure 2.9 (b). Each sub-image represents a single pass of the algorithm.

Algorithm 2 Lee et al. Skeletonisation

while $number\_of\_changes > 0$ **do**
  $removable \leftarrow \emptyset$
  $i \leftarrow 0$ ▷ The set of all removable voxels
  $directions \leftarrow [north, south, east, west, up, down]$ ▷ once for each cardinal direction
  for $direction \in directions$ **do**
    for $voxel \in forground$ **do**
      if $(voxel[+direction] \in foreground) \cap (voxel[-direction] \in background)$ **then**
        if $countNeighbours(voxel) > 2$ **then**
          if $calcEuler(voxel) == 0$ **then**
            if $isSimple(voxel)$ **then**
              $removable \leftarrow voxel$
              $number\_of\_changes = number\_of\_changes + 1$
            end if
          end if
        end if
      end if
    end for
  end for
  $i \leftarrow i + 1$
  $voxels[removable] \leftarrow 0$ ▷ All removable voxels are set to background simultaneously
  $iteration[removable] \leftarrow i$
end while
return $voxels, iterations$

Algorithm 3 countNeighbours - Count number of foreground voxels in neighbourhood

$count \leftarrow 0$
for $neighbour \in voxel\_neighbourhood$ **do**
  if $neighbour\ is\ foreground$ **then**
    $count \leftarrow count + 1$
  end if
end for
return $count$

Algorithm 4 calcEuler - Calculate the Euler number of a 3x3x3 neighbourhood using a lookup table.

$euler \leftarrow 0$
$lut \leftarrow euler\ values\ for\ 2x2x2neighbourhood$
for $i\ in\ range\ 1\ ..8$ **do** ▷ The 3x3x3 neighbourhood is treated as 8 2x2x2 neighbourhoods
  $euler \leftarrow euler + lut[voxel\ neighbours[i]]$
end for
return $euler$

Algorithm 5 isSimple - Find the connectivity of the voxels neighbourhood with the voxel removed.

$temp \leftarrow 0$
$temp[voxel\ neighbours \in foreground] \leftarrow 1$ ▷ set all foreground to 1.
$temp[14] \leftarrow 0$ ▷ clear the central voxel
$count \leftarrow count\_labels[temp]]$
if $count < 2$ **then** ▷ If the count is greater than 1, removing the voxel will cause two parts of the skeleton to become disconnected.
    return $True$
else
    return $False$
end if

2.4.2.3 Thinning using the Palágyi and Kuba Algorithm

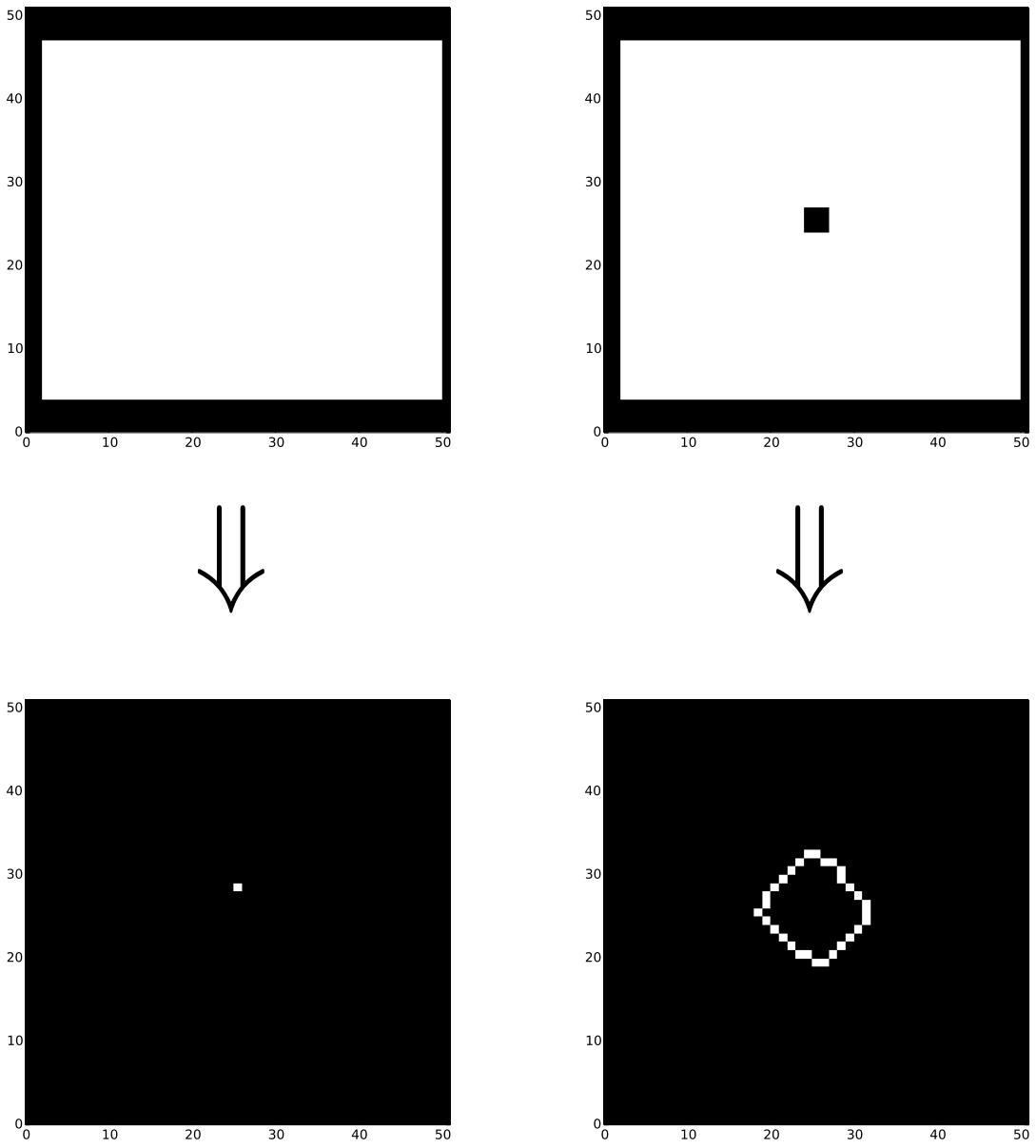
Like the Lee et al. algorithm, the Palagyi and Kuba algorithm considers a 3x3x3 neighbourhood to decide if removal of a voxel will cause the topology to change. Unlike both previous algorithms, it does not consider the cardinal directions when finding candidates for removal, but rather tests across diagonals. At each step all candidate voxels are tested for removal by the use of a set of templates, an arrangement of voxels in the neighbourhood that matches the template can be removed without affecting the topology of the volume.

Algorithm 6 shows an implementation of the thinning algorithm. Unlike the previous two algorithms, voxels are removed after every sub-iteration rather than at the end. This results in a less regular skeleton than the previous two algorithms. Figure 2.11 shows the effect of skeletonisation on two simple shapes, while Figure 2.12 shows the evolution of thinning a simple shape.

2.4.3 Graph Filtering

In their paper on path planning for virtual bronchoscopy, Kiraly et al. discussed the causes of *false branches*, skeleton elements that are caused by imaging or processing artifacts [Kiraly et al., 2004]. Removal of false branches produced a skeleton that more closely reflected the underlying state of the patients lungs.

Even with an absolutely correct model of the environment, the skeletisation algorithm may still produce branches based on geometry rather than topology. This geometric information



(a) (b)

Figure 2.11: Skeletisation with Palágyi and Kuba of an input volume with (a) a square of pixels (b) the pixels with the removal of a 3x3 section.

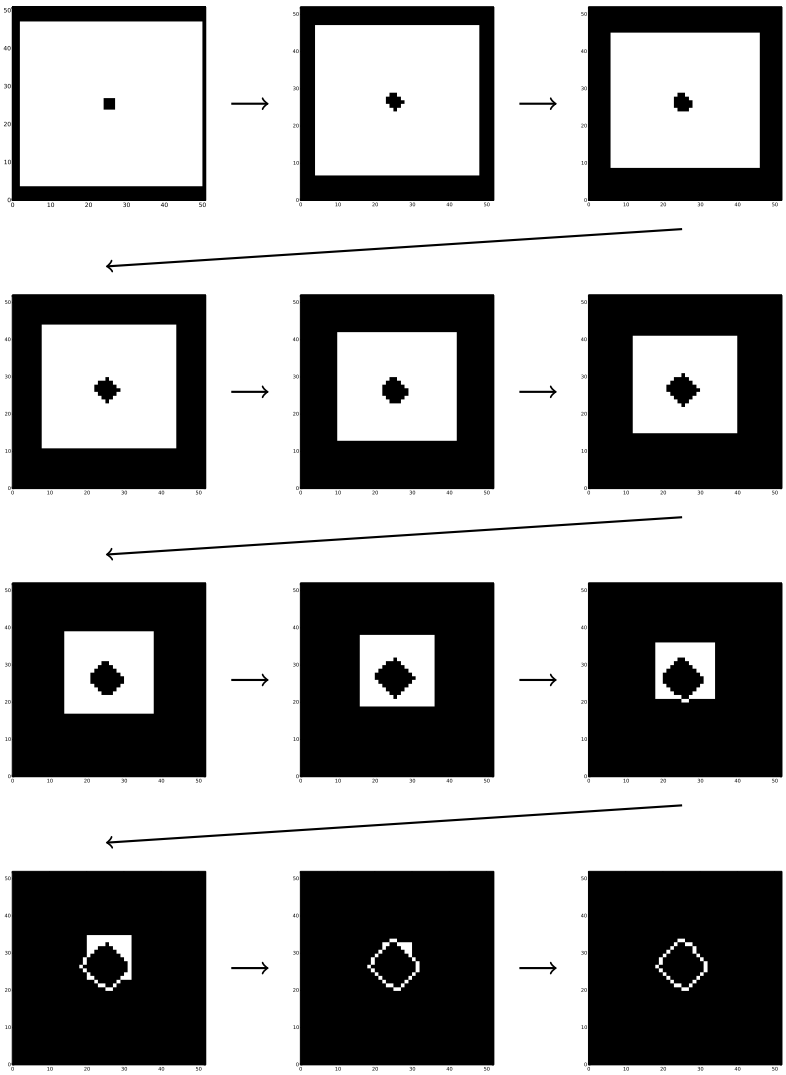


Figure 2.12: Evolution of the Palágyi and Kuba algorithm on the shape from Figure 2.11 (b). Each sub-image represents a single pass of the algorithm.

Algorithm 6 Palágyi and Kuba Skeletonisation

while $number\_of\_changes > 0$ **do** ▷ The set of all removable voxels
  for $i\ in\ range\ 1\ to\ 12$ **do** ▷ once for each combination of cardinal directions
    $removable \leftarrow \emptyset$
    for $voxel\ in\ all\ forground\ voxels$ **do**
      if $voxel\ in\ background$ **then**
        $rotate\ neighbourhood$
        if $template[neighbourhood]\ is\ true$ **then**
          $removable \leftarrow voxel$
          $number\_of\_changes = number\_of\_changes + 1$
        end if
      end if
    end for
    $voxels[removable] \leftarrow 0$ ▷ Removable voxels are set to background at the end of each subiteration
  end for
end while

results in branches that terminate in endpoints. Since these endpoints do not connect more than once to the skeleton, they can be contracted back to their junction without altering the underlying topology of the space. Given that the skeleton is homotopic with the fundamental group, it will contain the minimum number of loops while still maintaining topology. Since the fundamental group contains all the branches that represent the possible alternate paths through a space, it contains the minimal set of information required to generate a schedule. Using techniques adapted from Kiraly et al., the skeleton may be filtered to produce the fundamental group. This filtered skeleton may produce a graph that is more suited to scheduling. The techniques that will be tested include the contraction of endpoints, the removal of redundant nodes, and the merger of closely spaced nodes.

2.4.3.1 Contraction of Endpoints

Since singly connected nodes are trivial to identify, their removal is performed by finding the connected branch and setting their values to background. The node at the other endpoint has the branch removed from its list of connected components. This process is repeated until no further removable branches are found. This is equivalent to the length-based elimination technique used by Kiraly et al. with a minimum length of zero [Kiraly et al., 2004]

2.4.3.2 Removal of Redundant Nodes

With the removal of endpoints and connected branches, some nodes may remain that have two branches connected, making them topologically identical to branches. These are removed by picking one branch and setting the voxels of the redundant node and branch to its identifying value.

2.4.3.3 Merger of Closely Spaced Nodes

The skeletisation algorithm can produce nodes that are connected by a branch of trivial length. To identify and remove these nodes, a heuristic is applied that examines the thinning distance along the length of each branch. If the minimum distance is less than the distance

between nodes, the nodes can be considered topologically redundant. Merger between nodes is accomplished by marking both nodes and the connecting bridge to be a single node with the first nodes identifier.

The resulting graph is referred to as the *reduced topological graph* due to its similarity to the *reduced general Voronoi graph* proposed by Choset and Nagatani [Choset and Nagatani, 2001].

2.4.4 Segmentation

With the skeleton reduced to contractible segments, regions can be identified by reconstructing a segmented volume while maintaining topology. In the proposed algorithm a combination of both the critical point detection and geometric reconstruction approaches will be used. As mentioned, the thin curve skeleton lacks the geometric information used in both the generalised Voronoi diagrams and the power crust algorithms. However, the skeletisation algorithms used are all iterative, removing a layer of simple voxels at a time from the border of the volume, so the iteration number required to remove a voxel can be recorded. This *thinning distance* $d_{thinning}$ can be used as a proxy for the geometric information when attempting a reconstruction.

As discussed in section 2.3.2, approaches that use critical lines to generate bounded regions have been used by several groups. These lines can bound regions that are in $\mathbb{R}^2$, but because lines cannot bound regions in $\mathbb{R}^3$ and higher, this approach does not generalise into higher dimensional spaces. To find an approach that can work in such spaces, a number of alternate methods were examined;

- region growing across the Chebyshev distance to create a watershed
- a method inspired by the ”Power Crust” spherical reconstruction - growing regions to a distance set by $d_{thinning}$
- watershed segmentation using $d_{thinning}$

Detail of these algorithms will be presented next, with an evaluation of their effectiveness for planning will be presented in the next chapter.

2.4.4.1 Region Growing

The nodes form a set of seed points that show what parts of the volume have changing topology. To find what parts of the foreground should be associated with each node can be performed by finding which node is the closest to each individual graph element.

This could be performed by finding the distance for each such element to each node, but would require a distance to be calculated for each element. If instead, the nodes are used as a seed point, and regions are grown from these seeds, a general solution for the elements' connectivity can be established. These regions expand until they reach the image background or they meet at which point they form the *watershed*.

Three techniques were used to calculate these regions. The first was to simply find the watershed across they Chebyshev distance. But, since the nodes were not equidistant from the background and each other, they would sometimes completely enclose a background feature resulting in a segment where not all trajectories were homotopic. An algorithm implementing this approach is shown in Algorithm 7. This is similar to a search algorithm in that a frontier is generated representing the next set of voxels to be expanded. Since cell expansion is based on adjacency, this is a form of breadth-first search [Russell and Norvig, 2010].

2.4.4.2 Seeded Region Growing

The power crust algorithm mentioned in 2.3.2 uses the property of the medial axis being the locus of maximally inscribed spheres to reconstruct a volume. In this algorithm, a set of spheres are created along the medial axis with radii equal to the distance from the edge as found by the iteration number of the voxels of the medial axis. This algorithm was applied to the reconstruction of thinned skeletons. However, where the medial axis guarantees that the locus is equidistant from the edges of a volume, the topological skeleton is not always centered. As such, a seeding distance was calculated by finding the minimum iteration number of the removed voxels adjacent to the skeleton. This minimial thinning distance was used as a limit on the regions' growth. Once all regions had reached their maximum growth

as constrained by $d_{thinning\_min}$, a second pass was performed to grow the regions until no unallocated segment was left. This method, where a graph is searched for a certain number of iterations, is a form of iterative deepening search [Russell and Norvig, 2010].

2.4.4.3 Watershed using $d_{thinning}$

The third algorithm used the watershed implementation from the scikit image package to calculate the segmentation [scikit-image development team, 2015b]. Typically, this algorithm would use the distance transform to generate an input matrix with the gradient of the image. Since the nodes are at the minima of the thinned image, the thinning distance $d_{thinning}$ was used instead, allowing the regions to grow along the inverse of the thinning algorithm. This method, where lowest cost cells are added first, is a form of equal-cost search [Russell and Norvig, 2010].

2.5 Conclusion

This chapter has outlined the available methods for handling spatial data for planning, with an emphasis on the available algorithms for compression of belief spaces into forms suitable for planning. In particular a number of skeletisation, filtering and reconstruction methods have been identified as candidates for effective planning graph creation.

As noted in the previous chapter however, searching for a plan with a symbolic planner can

Algorithm 7 Region growing

$frontier \leftarrow seed$ ▷ Initialise with the set of seed points
$unallocated \leftarrow foreground - seed$
$new\_frontier \leftarrow \emptyset$
while $number\_of\_changes > 0$ **do**
  $number\_of\_changes \leftarrow 0$
  for $voxel$ in $frontier$ **do**
    for $neighbour$ in $(neighbourhood(voxel) \cap unallocated)$ **do**
      $new\_frontier \leftarrow neighbour$
      $number\_of\_changes = number\_of\_changes + 1$
    end for
  end for
end while return voxels

be an expensive operation. Thus, it is important to minimise the number of states that the planner must search to find a valid plan. The next chapter will cover the testing of these algorithms for suitability in planning applications, while subsequent chapters will describe integration with robotic planning systems.

Table 2.2: Summary of Selected Geometric Simplification Algorithms

Type	Name	Summary
Thinning	Lee et al. [Lee et al., 1994] thinning	Grid based removal of simple points from the outermost elements. Simple points are identified using the euler number of the 3x3x3 neighbourhood and connectivity tests.
	Palágyi and Kuba. [Palágyi and Kuba, 1999] thinning	Grid based removal of simple points from the outermost elements. Simple points are identified using a set of templates.
	Medial Axis Transform	Grid based method that iteratively removes the outermost set of elements until the remainder is thin.
Segmentation	Region Growing	Grow segments from an initial population by iteratively adding to the frontier. Similar to breadth first search
	Seeded Region Growing	Grow segments from an initial population by iteratively adding to the frontier until a preset iteration count is reached. Similar to iterative deepening search
	Watershed	Grow segments based on order of removal during thinning operation. Similar to equal-cost search.

Chapter 3

Evaluation of Topological Planning

3.1 Introduction

In the previous chapter, a number of potential algorithms for the creation of graphs and segmentations for robotic planning were examined. In particular, skeletisation with the Lee et al. [Lee et al., 1994] and Palágyi and Kuba [Palágyi and Kuba, 1999] algorithms could be useful for belief compression in robotic planning.

To provide a useful comparison with existing approaches, the medial axis transform was chosen due to it's use in systems such as Powercrust [Amenta et al., 2001], and it's similarity to the Voronoi graph used in Thrun's work on metric topological maps [Thrun, 1998]. In this chapter, the effectiveness of the skeletonisation and medial axis transform algorithms at producing data suitable for symbolic planning will be examined.

3.1.1 Implementation

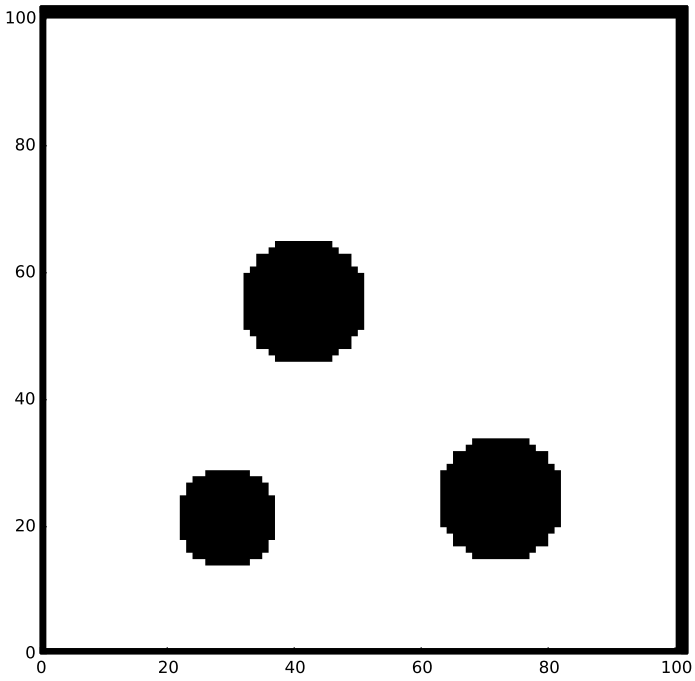
Implementations of these algorithms are available from sources including Cornea [Cornea, 2005] and Homann [Homann, 2007], however a unified codebase suitable for eventual testing with robotic hardware is required. For this reason all thinning and segmentation algorithms were implemented as a python library. This allowed direct comparison between the performance of different algorithms by simply altering the function calls that were performed.

Testing of these algorithms was performed by the use of a combination of Bourne Again SHell (BASH) [Free Software Foundation, Inc, 2018] and Python [Python Software Foundation, 2018] scripts. A set of test images were first generated, then each image was processed by a testing script and the result was saved to a text file as a set of comma separated values. Once the testing was complete, another script file was executed that performed analysis and generated figures. Statistical analysis of data was performed using algorithms from the NumPy mathematical core library of the SciPy ecosystem [Jones et al., 2001].

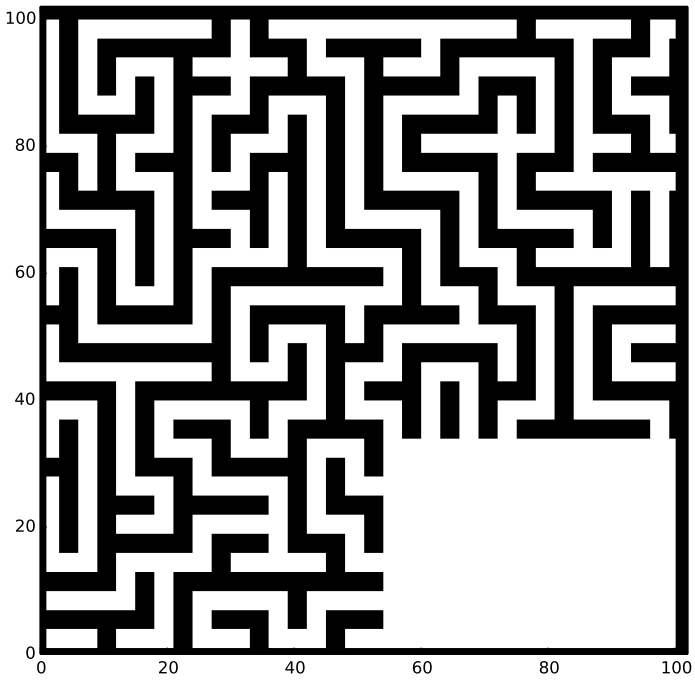
3.1.2 Evaluation of Skeletisation

While the intention is to test these algorithms on spatial data, part of the hypothesis is that the complexity of the robotic planning problem in spatial environments can be made to scale with the complexity of the environment rather than the size of the map being processed. To evaluate the effectiveness of the skeletonisation algorithm, randomly constructed input images were created in two basic configurations: an open configuration representing an outdoor environment with small obstacles, an example of which can be seen in Figure 3.1(a), and a closed configuration representing an indoor environment, consisting of open spaces connected by maze-like corridors generated from a script [Wingbermuehle, 2010], an example of which can be seen in Figure 3.1(b). To illustrate the effects of thinning, Figure 3.2 shows the skeleton generated by thinning with the Lee et al. algorithm, while Figure 3.3 shows the skeleton generated by thinning with the Palágyi and Kuba algorithm.

At the core of the proposed algorithm is the proposition that the removal of redundant information using skeletonisation allows the production of planning graphs that scale with the complexity of an input image rather than the number of elements it contains. To investigate this the number of *nodes*, places where a skeleton branches, were investigated for a range of sample images and skeletonisation types. This investigation was performed by generating a 400×400 pixel obstacle type image, with down-sampled images scaled to 320×320, 240×240, 160×160 and 80×80 pixels. The original and down-sampled images were then skeletonised and their nodes counted, and the results graphed in Figure 3.4(a). The Lee skeleton was stable through all sizes except the smallest while the Palágyi skeleton showed a small in-

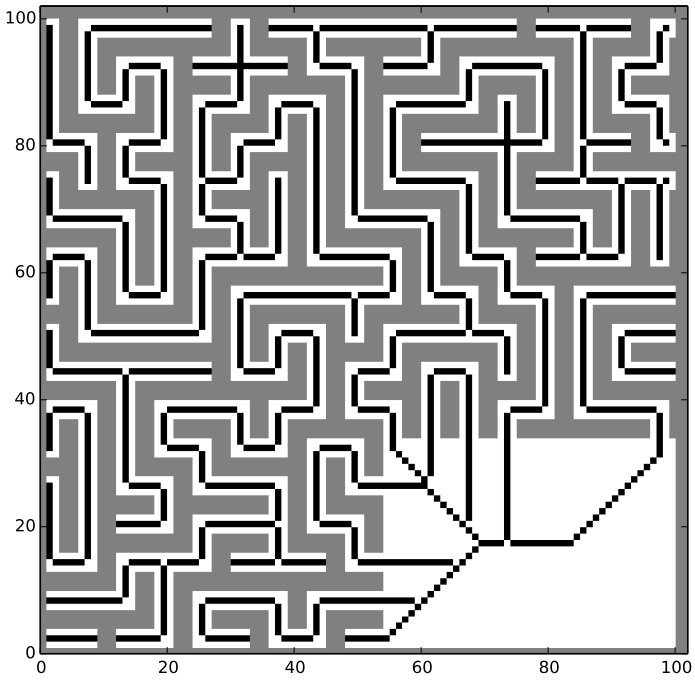


(a)

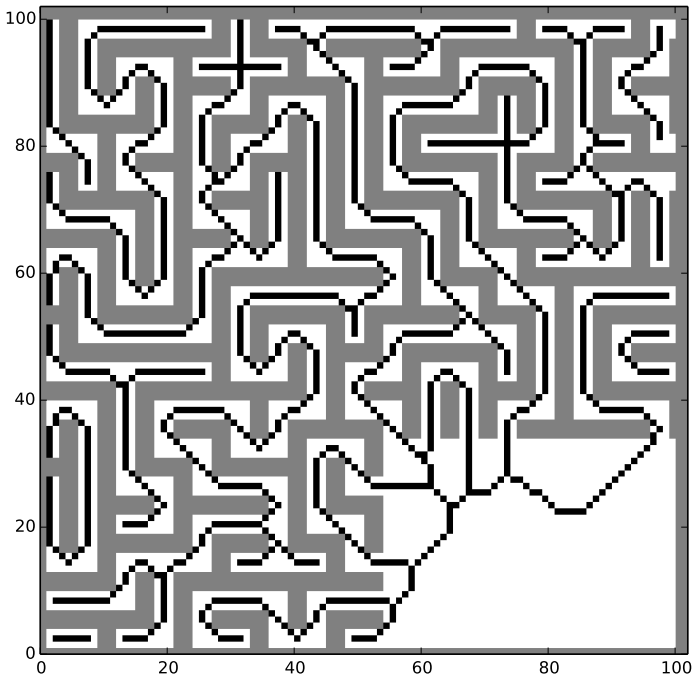


(b)

Figure 3.1: Example images of the obstacle (open) and maze (closed) type (a) An obstacle type image. (b) A maze type image. Results of skeletonisation with these images can be seen in Figures 3.2 and 3.3.

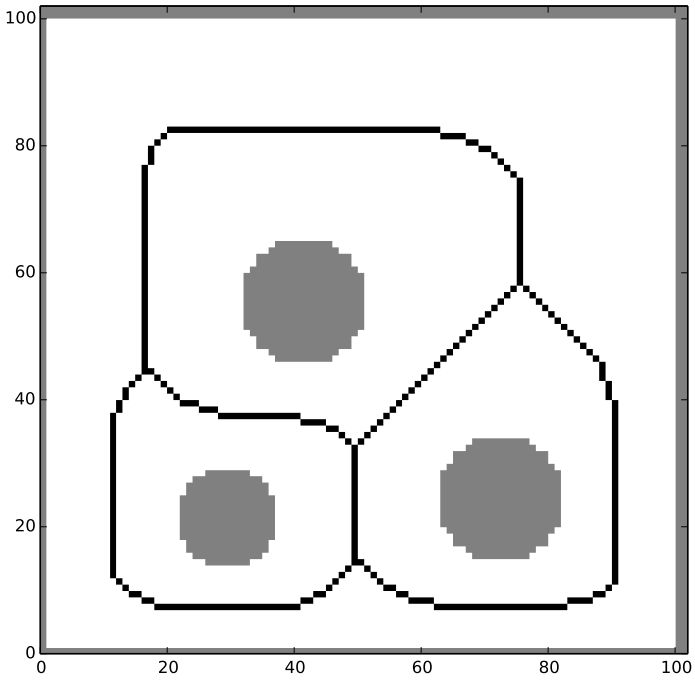


(a)

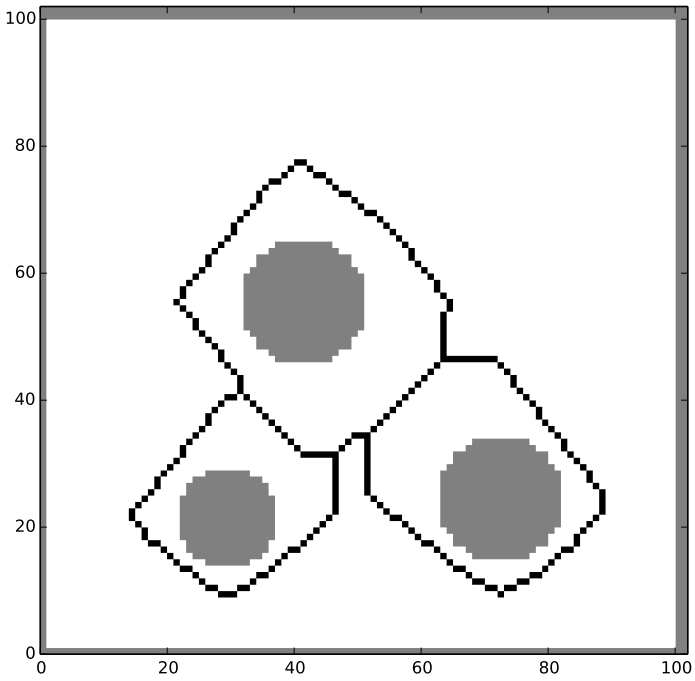


(b)

Figure 3.2: Results of skeletisation of the maze image in Figure 3.1(b) using the (a) Lee et al. algorithm (b) the Palágyi and Kuba algorithm. Grey pixels indicate obstacle, white pixels are free space, black pixels are the resultant skeleton.



(a)



(b)

Figure 3.3: Result of skeletisation of the obstacle image in Figure 3.1a using the (a) Lee et al. algorithm (b) the Palágyi and Kuba algorithm. Grey pixels indicate obstacle, white pixels are free space, black pixels are the resultant skeleton.

creasing trend. The medial axis transform showed a much larger increase in the number of nodes due to change in scale, with the number of nodes increasing by approximately 50% from the smallest to largest images.

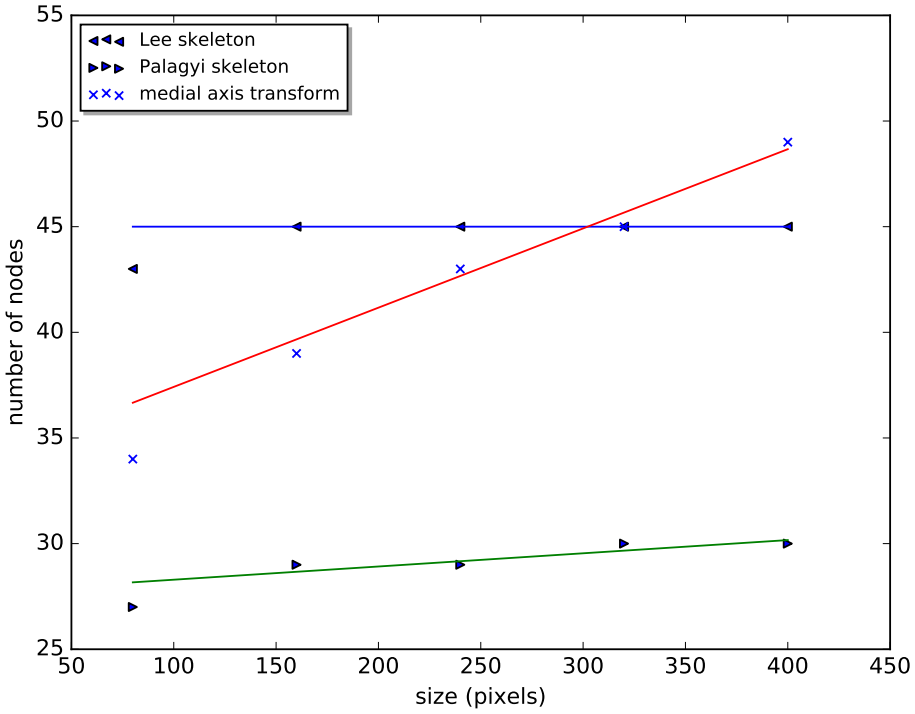
The smaller number of nodes in the Palágyi skeletonisation of the 80×80 sized image can be explained by the merger of two obstacles that were separated by a single pixel band in the source image as shown in Figures 3.5 and 3.6. With the removal of 96% of the pixels from the original image, these were merged into a single obstacle. This lends support to the hypothesis that the Palágyi skeleton is robust to changes in scale.

A similar approach was taken to evaluating the effect of rotation on the skeletonisation algorithms. An obstacle image was created and then used to generate rotated images at $30°$, $45°$, $60°$, $90°$, $120°$, $135°$, $160°$, and $180°$. Each image was skeletonised, the nodes counted, and a graph generated for each angle as shown in Figure 3.4(b). The Lee skeleton showed a large variation in the number of nodes generated depending upon the angle of the rotated image. Examination of the rotated skeletons showed that there was little variation in the closed loops of the graph, but a significant variation was found in the number of endpoints generated due to variations in the geometry around obstacles. The Palágyi skeleton only had a single change in the number of nodes generated. Examination of the $90°$ and $120°$ images shown in Figures 3.7 and 3.8 shows a pair of three-way nodes in the former image that were connected by a short branch had become joined into a single four-way node in the latter image. This artefact is not present if the image is flipped before skeletonisation.

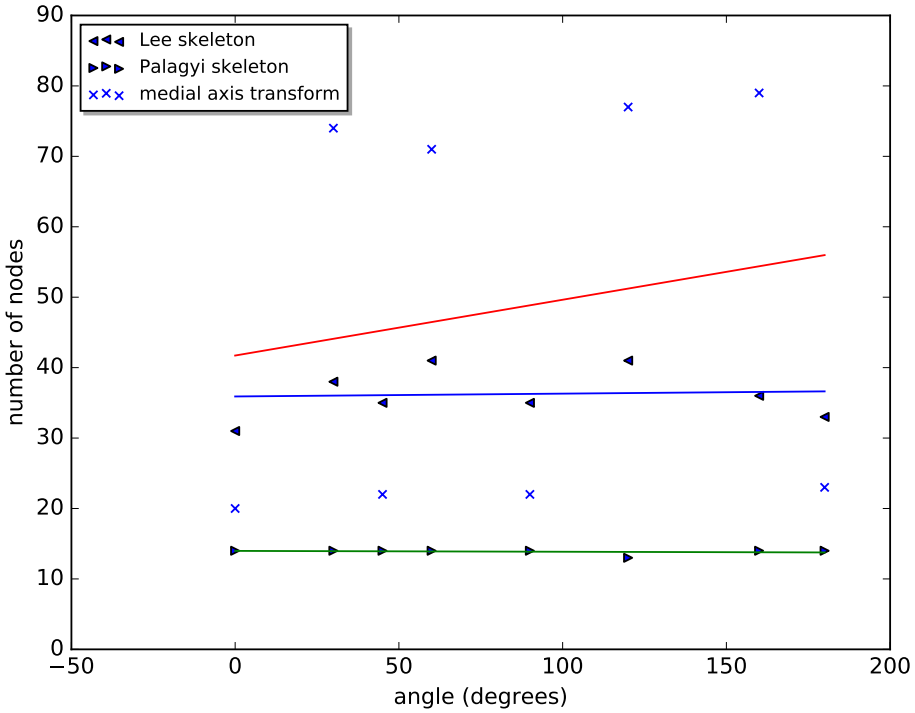
3.1.3 Evaluation of Topological Segmentation Algorithms

The skeletons generated are thin, as such they do not map directly to the space that the skeleton was generated from. To allow this relationship to be found, three segment reconstruction algorithms were selected in Chapter 2. An example of the effect of such an algorithm can be seen in Figure 3.11

As stated in the introduction, the core hypothesis of this thesis is that skeletonisation algorithms can be used to infer topological information that can then be used to inform planning algorithms. if the inferred information is based on the underlying topological information,

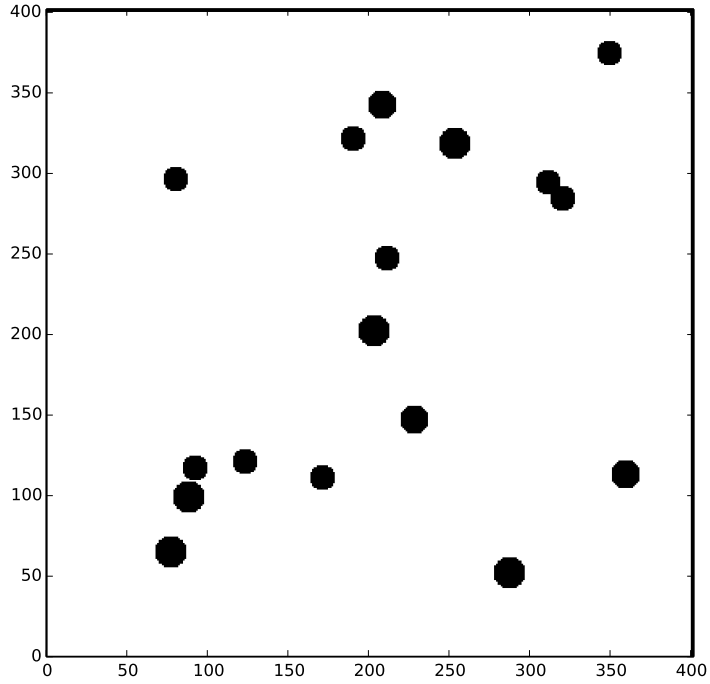


(a)

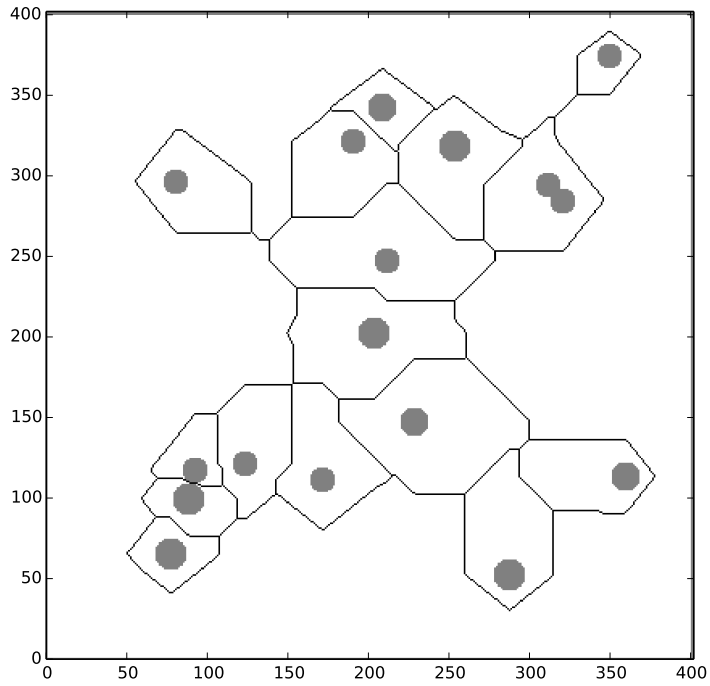


(b)

Figure 3.4: Number of nodes in the scaled and rotated images with least-squares regression lines. Original images can be seen in Figures 3.5(a) and 3.7(a). (a) Nodes vs size for scaled images. The 80 pixel value is excluded from the lines of best fit as an outlier. (b) Nodes vs angle for rotated images.

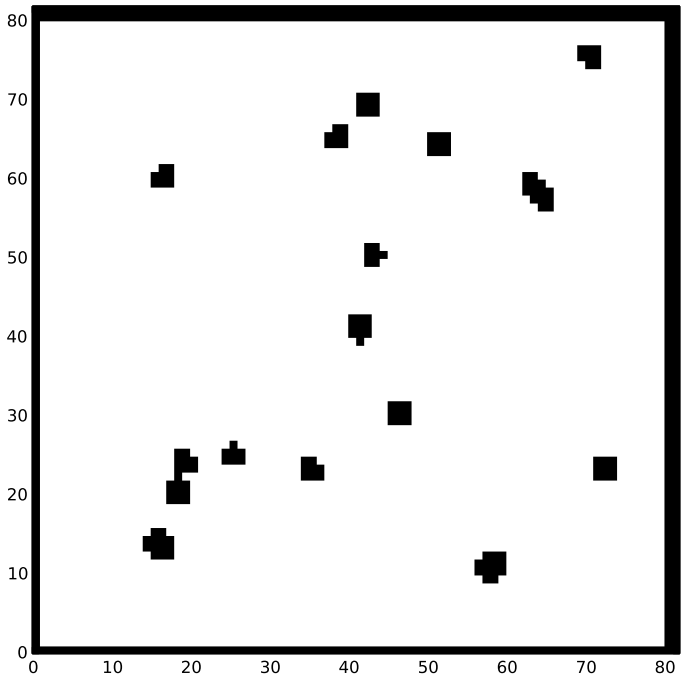


(a)

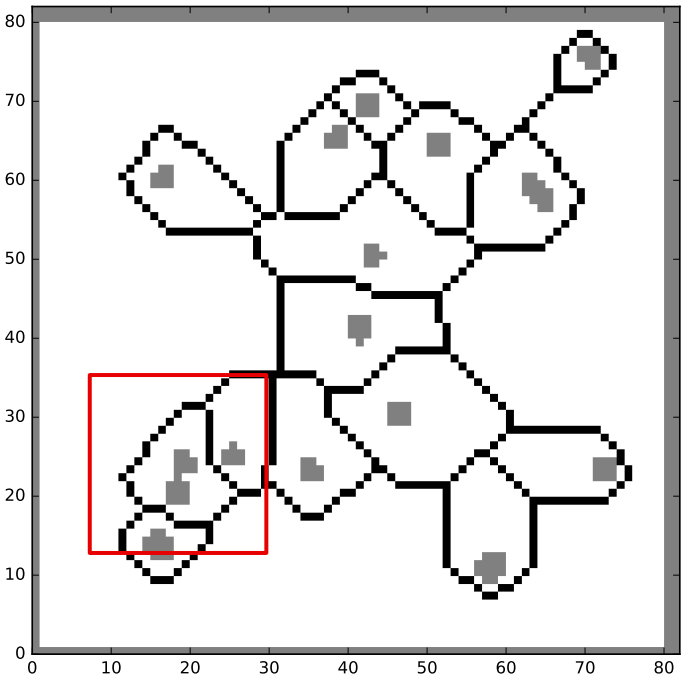


(b)

Figure 3.5: Scaling artifact for the Palágyi and Kuba skeleton. (a) The original full scale image (b) Palágyi and Kuba skeleton of the full scale image. Scaled down images are in Figure 3.6.

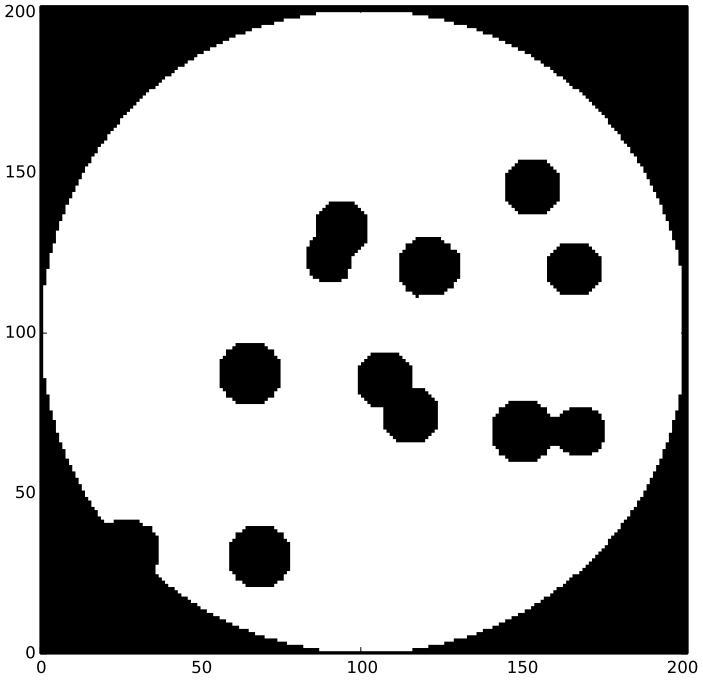


(a)

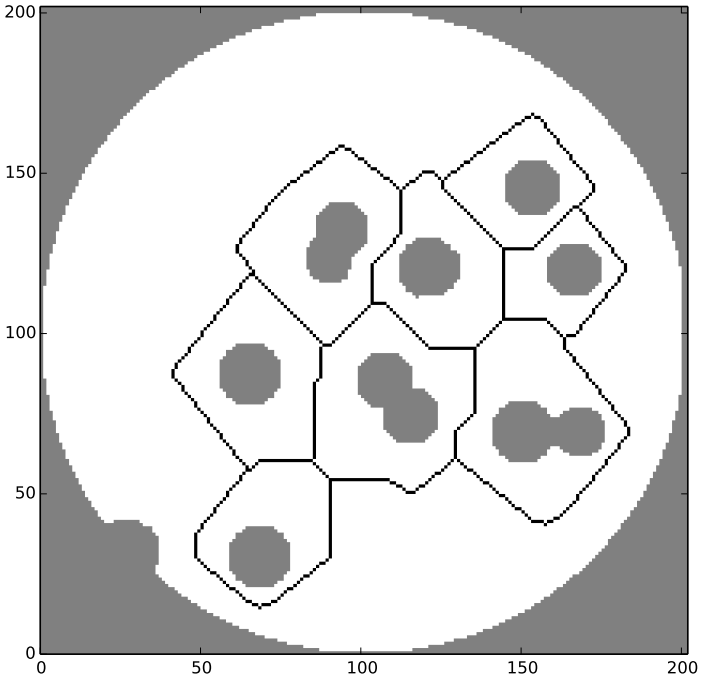


(b)

Figure 3.6: Scaling artifact for the Palágyi and Kuba skeleton. (a) image scaled down to 80 by 80 pixels. (b) Palágyi and Kuba skeleton of the scaled image. The merged obstacle is indicated by a red square.

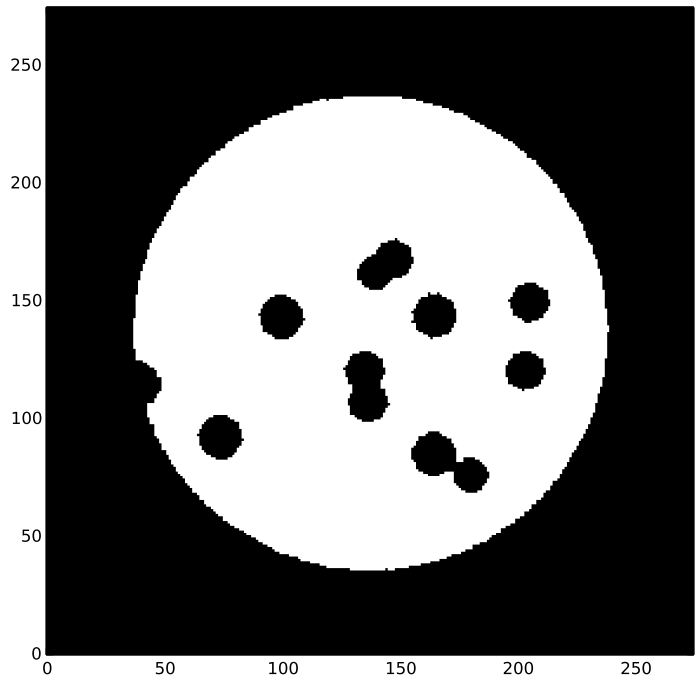


(a)

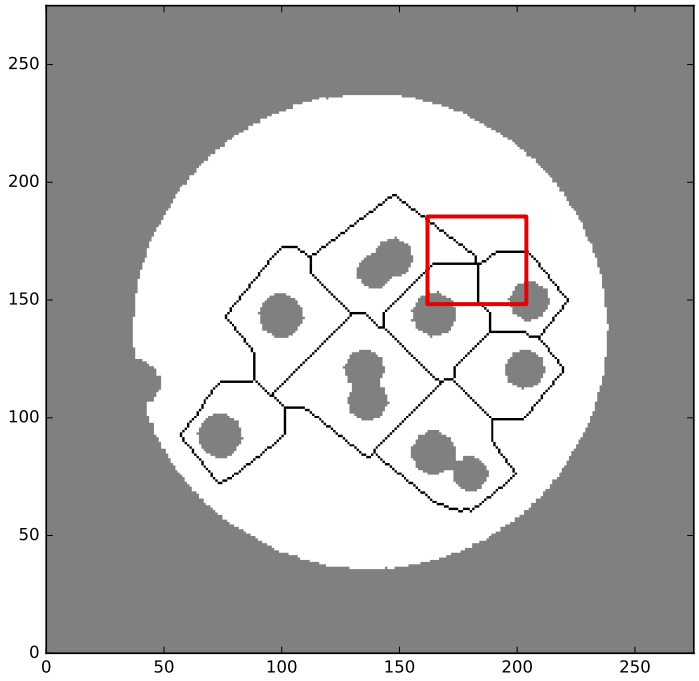


(b)

Figure 3.7: Rotation artifact for the Palágyi and Kuba skeleton. (a) The 90° image (b) Palágyi and Kuba skeleton of the 90° image. Differences in image size are to prevent rescaling due to the source and destination images being square. The 120° image is visible in Figure 3.8.

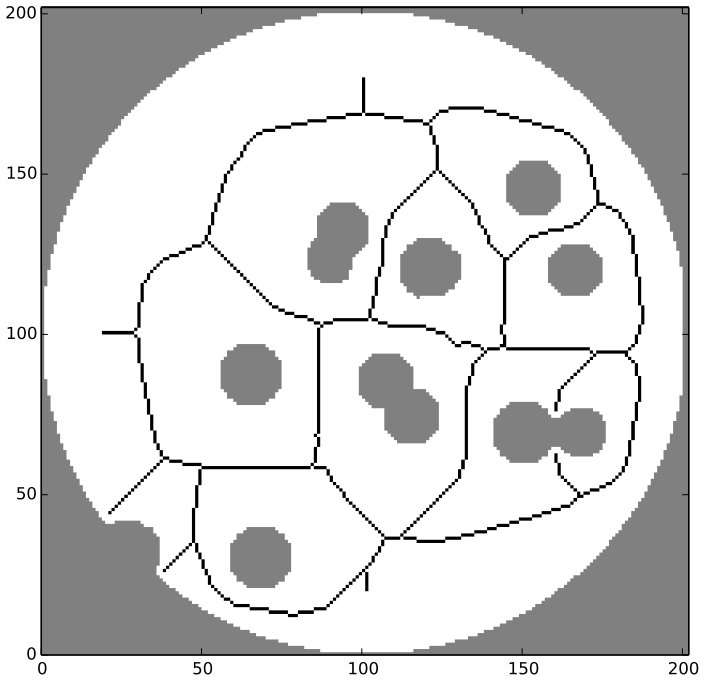


(a)

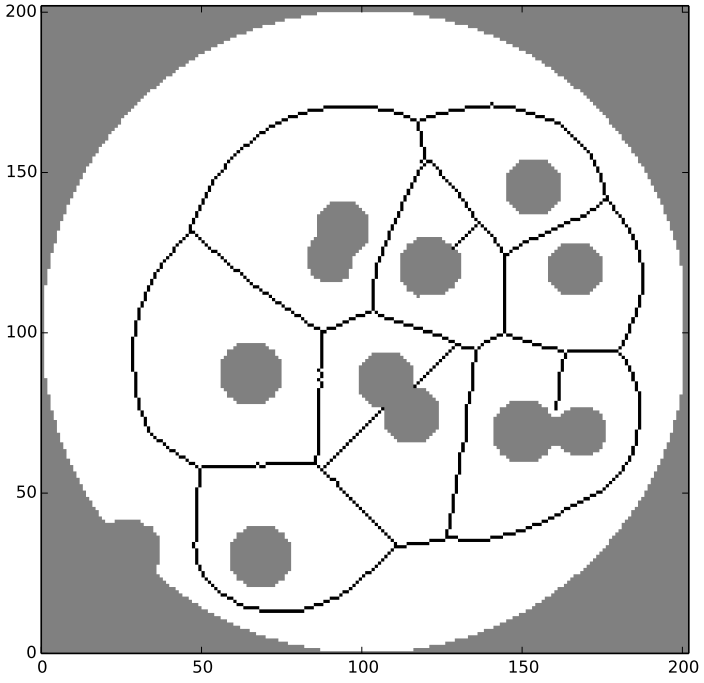


(b)

Figure 3.8: Rotation artifact for the Palágyi and Kuba skeleton. (a) the 120° image. (b) Palágyi and Kuba skeleton of the 120° image. Differences in image size are to prevent rescaling due to the source and destination images being square. Artifact is indicated by a red square in the 120° image.

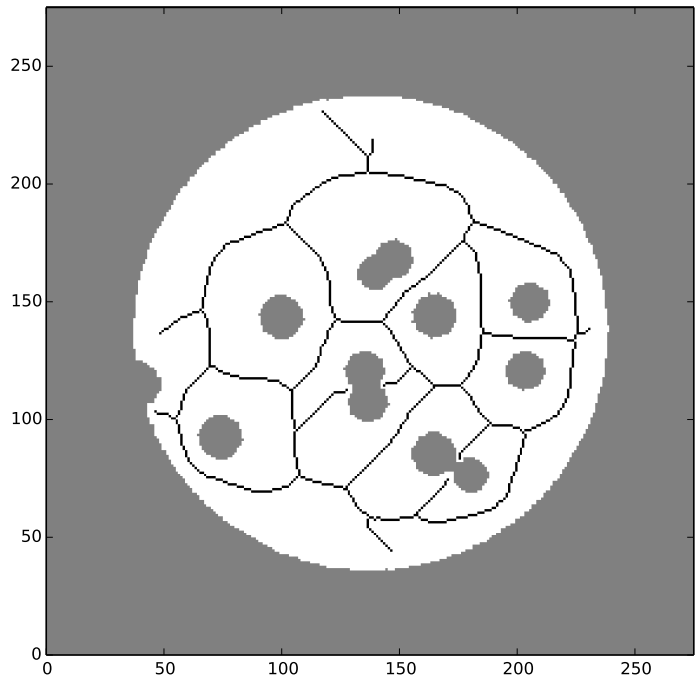


(a)

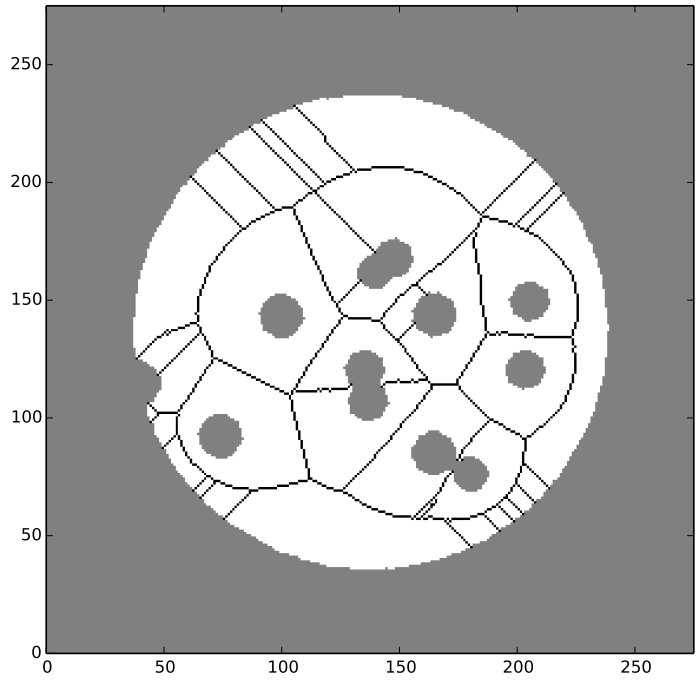


(b)

Figure 3.9: Lee and Medial skeletons generated from the rotation images in Figure 3.7. (a) Lee skeleton of the 90° image. (b) Medial axis skeleton of the 90° image.



(a)



(b)

Figure 3.10: Lee and Medial skeletons generated from the rotation images in Figures 3.7 and 3.8. (a) Lee skeleton of the 120° image. (b) Medial axis skeleton of the 120° image.

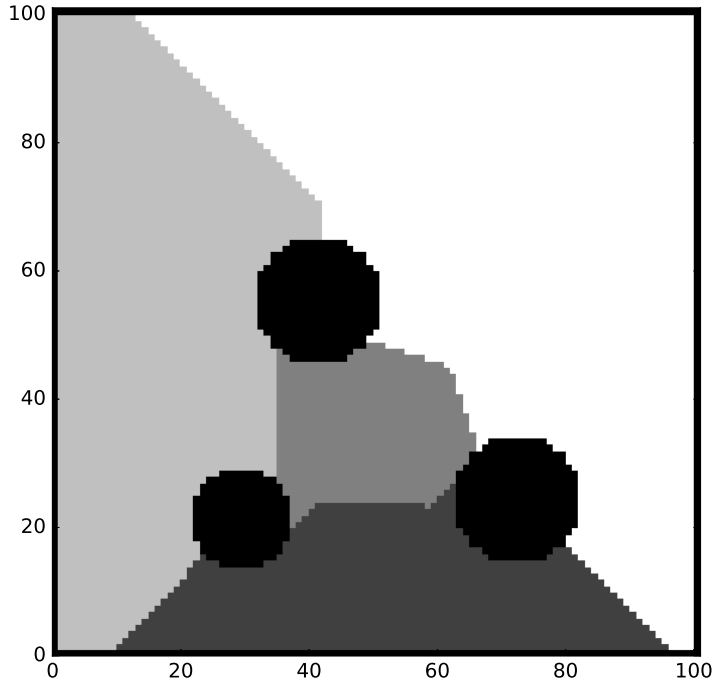


Figure 3.11: Reconstruction of segments from Lee et al. skeleton using seeded segmentation. Input is the three obstacle image in Figure 3.1a. Greycale areas are the reconstructed segments.

then the inferred topological elements should be themselves topologically simple - their fundamental group must not contain any loops. Such a shape can be contracted to a single voxel while maintaining homotopy.

To evaluate the proportion of segments that were homotopic to a single voxel, twenty obstacle images were generated with sizes randomly selected between one hundred and four hundred pixels on a side. Each image contains between 10 and 30 randomly placed circular obstacles. Twenty further maze images were generated with sizes between 100×100 and 400×400 pixels.

These images were skeletised using both the Lee and the Palágyi algorithms, with the total number of generated nodes shown in Table 3.1. These images were segmented with each of the algorithms and checked for constant homotopy. A segment was declared as non-contractible if a closed trajectory could be constructed within a cell that could not be collapsed to a single voxel.

The number of non-homotopic segments was identified for each image type, skeletisation algorithm and segmentation type and the result shown in Table 3.2. At most, 3% of reconstructed segments were non-homotopic for all skeletons and segmentation, with the maze type images consistently producing lower numbers of non-homotopic segments. For the maze type images, the Lee skeleton produced a lower proportion of non-homotopic segments than the Palágyi skeleton. In Section 2.4.3, the use of filtering techniques on the skeleton to make it more closely approach the fundamental group was discussed. To evaluate if node filtering can reduce the number of nodes within the graph, these techniques were applied to the obstacle and maze type images. The results of this operation are shown in

Table 3.1: Number of generated nodes for obstacle and maze type images. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 40$.

Skeletisation	Number of nodes	
	Obstacle	Maze
Lee et al.	582	9498
Palágyi and Kuba	521	9481
Medial Axis Transform	842	32611

Table 3.2: Proportion of non-homotopic segments for all obstacle and maze trials. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 53535$.

Skeletisation	Segmentation	Proportion of non-homotopic segments	
		Obstacle	Maze
Lee et al.	Region Growing	0.034	0.013
	Seeded	0.000	0.006
	Watershed	0.002	0.000
Palágyi and Kuba	Region Growing	0.026	0.010
	Seeded	0.019	0.005
	Watershed	0.000	0.001
Medial Axis Transform	Region Growing	0.017	0.000
	Seeded	0.005	0.000
	Watershed	0.0	0.000

Table 3.3. All of the skeleton types exhibited a decrease in the number of nodes, with the medial axis skeleton on the maze image demonstrating a sixteen-fold reduction. Segmentation with the unfiltered skeletons exhibited only a small number of non-homotopic segments. To evaluate if this property was maintained in the filtered skeleton, the segmentation process was repeated with the filtered skeleton. A summary of the proportion of homotopy can be seen in Table 3.4. With filtering applied, the number of non-homotopic segments increased, while the total number of segments decreased resulting in an overall increase in the proportion of non-homotopic segments.

Table 3.3: Number of generated nodes with skeleton filtering, for obstacle and maze type images. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 40$.

Skeletisation	Number of nodes	
	Obstacle	Maze
Lee et al.	309	1178
Palágyi and Kuba	273	1309
Medial Axis Transform	475	1945

Table 3.4: Proportion of non-homotopic segments for all obstacle and maze trials with skeleton filtering. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 5489$.

Skeletisation	Segmentation	Proportion of non-homotopic segments	
		Obstacle	Maze
Lee et al.	Region Growing	0.055	0.181
	Seeded	0.000	0.104
	Watershed	0.000	0.002
Palágyi and Kuba	Region Growing	0.055	0.144
	Seeded	0.037	0.088
	Watershed	0.000	0.015
Medial Axis Transform	Region Growing	0.034	0.085
	Seeded	0.013	0.074
	Watershed	0.002	0.033

3.1.4 Evaluation of Spatial Planning

As discussed earlier, the intent of the topological graph creation is to produce a compressed representation of the input topology suitable for planning since an efficient belief compression method would not be useful if it did not allow the planner to make decisions about the environment.

The planning effectiveness of the spatial graph was evaluated by the construction of an experiment where the sequence of nodes to travel between each possible combination of nodes was found. For each pair of nodes a plan was generated from the planning graph containing the sequence of nodes to traverse. The shortest distance between the nodes across the volume was evaluated using Scikit-images' [scikit-image development team, 2015a] *route through array* algorithm while constrained to lie within the segments in the plan. This distance can be compared with the distance calculated with the path without segment constraints, thus giving a relative measure of the increase in calculated distance when using the constrained trajectory. If the planned sequence of segment traversals is optimal, that is the choice of segment traversals contains a path equal to the least-cost traversal between start and goal positions, then the length of the trajectory generated through these segments will be equal in cost to a path generated though the unconstrained map. An example of such a

path test can be seen in Figure 3.12. In this reconstruction, the path generated through the segments was not optimal.

By evaluating these paired distances for a significant number of possible paths, the distance from optimality of the planning system can be evaluated. Table 3.5 shows the mean lengths of the constrained paths as normalised against the optimal paths. These values approach unity, but as shown in Figure 3.13, many non-optimal trajectories are still close to optimal.

To more clearly show how the length of the non-optimal paths are distributed, the results were broken up into two sections. Table 3.6 contains the proportion of trajectories that were optimal as generated for all skeleton and segmentation types grouped by the image type, while a histogram of residuals was calculated for all trajectories that were not optimal. The Palágyi skeleton consistently produced the highest proportion of optimal trajectories across all image and segmentation types. When grouped by segmentation method, the watershed algorithm produced the most optimal trajectories for maze type images, while region growing

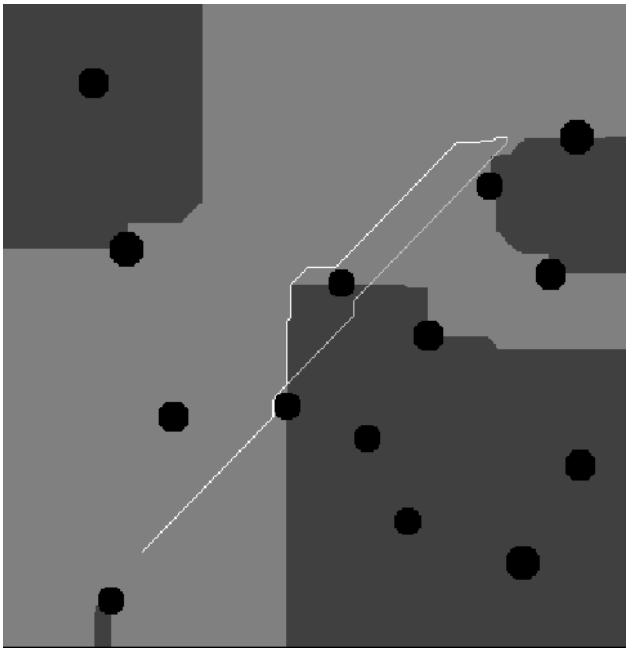


Figure 3.12: Test of path generation through plan segments. Black circles are obstacles, light area is the set of plan segments. Lighter path is constrained to pass through segments, darker path is unconstrained.

produced the most for obstacle type images. The seeded segmentation produced results that were in the middle of the other types for all image types and skeletons.

The residuals were calculated by finding the difference between the distance of each sub-optimal trajectory from the optimal and normalising the result. The residuals were then plotted as histograms grouped by image type, skeleton and segmentation. The resulting his-

Table 3.5: Mean of normalised path length for all obstacle and maze trials. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 254651$.

Skeletisation	Segmentation	Mean of constrained path length normalised to optimal path length.	
		Obstacle	Maze
Lee et al.	Region Growing	1.012	1.018
	Seeded	1.012	1.017
	Watershed	1.014	1.020
Palágyi and Kuba	Region Growing	1.007	1.012
	Seeded	1.008	1.011
	Watershed	1.012	1.011
Medial Axis Transform	Region Growing	1.013	1.023
	Seeded	1.012	1.021
	Watershed	1.011	1.014

Table 3.6: Proportion of optimal trajectories for obstacle and maze style images. Examples of an obstacle style image can be seen in Figure 3.1(a), and a maze style image can be seen in Figure 3.1(b). Total number of trials performed, $n = 254651$.

Skeletisation	Segmentation	Proportion of optimal trajectories	
		Obstacle	Maze
Lee et al.	Region Growing	0.80	0.43
	Seeded	0.79	0.45
	Watershed	0.79	0.45
Palágyi and Kuba	Region Growing	0.85	0.51
	Seeded	0.84	0.56
	Watershed	0.80	0.62
Medial Axis Transform	Region Growing	0.76	0.31
	Seeded	0.78	0.32
	Watershed	0.81	0.50

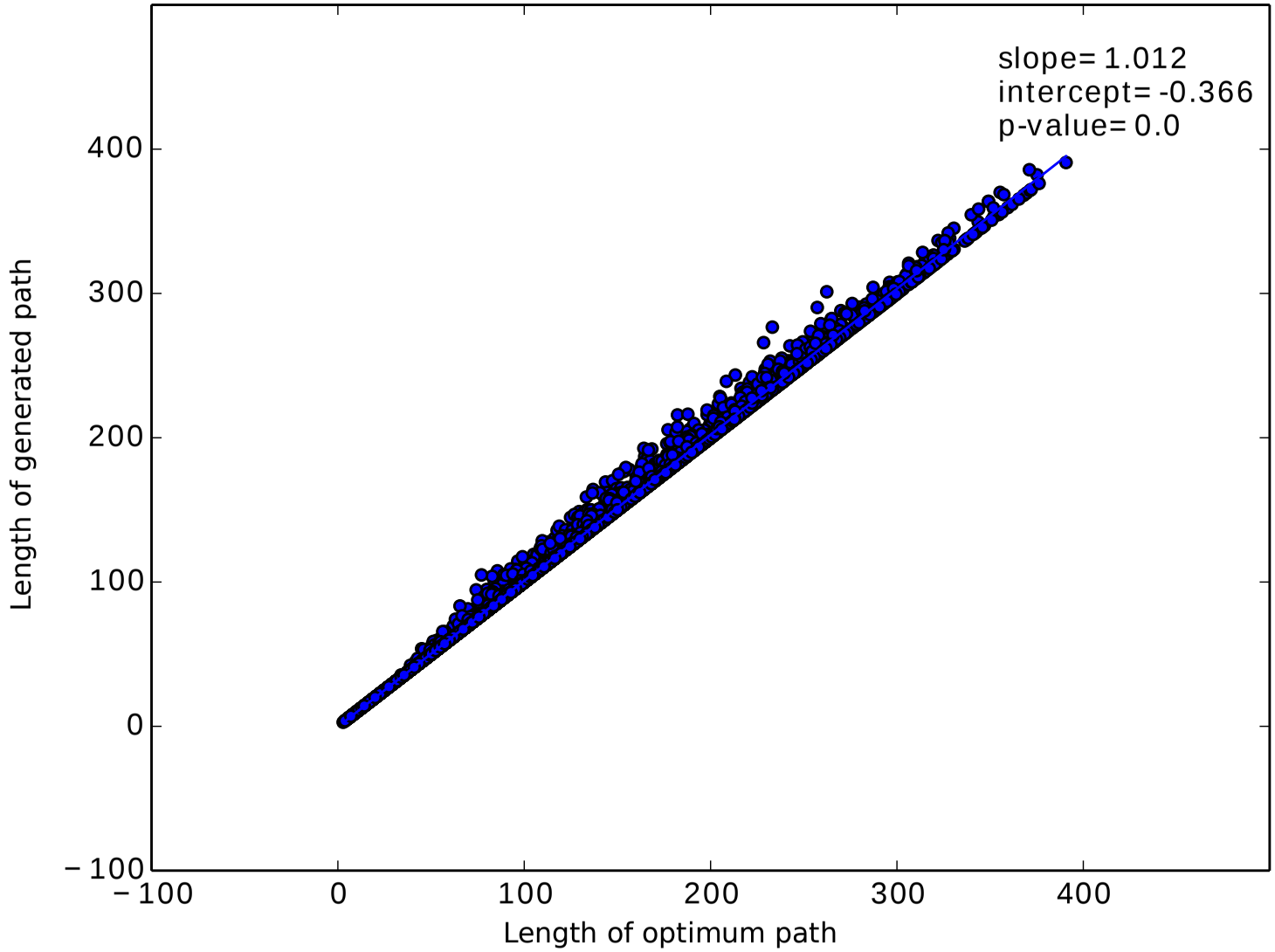
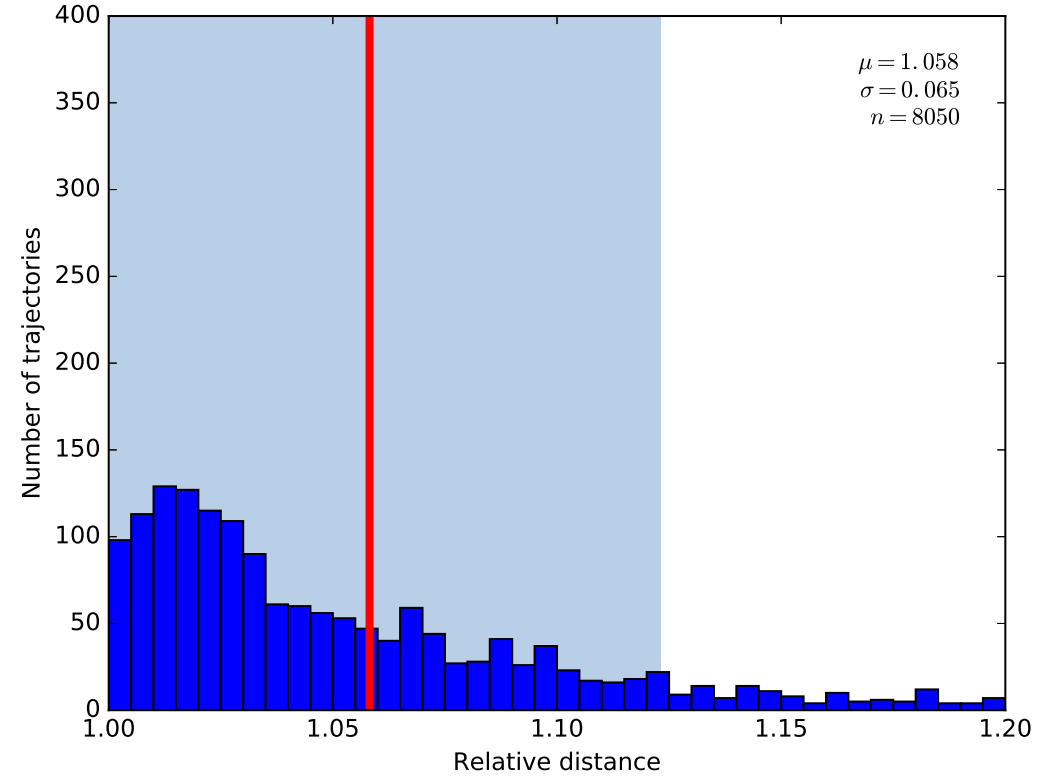
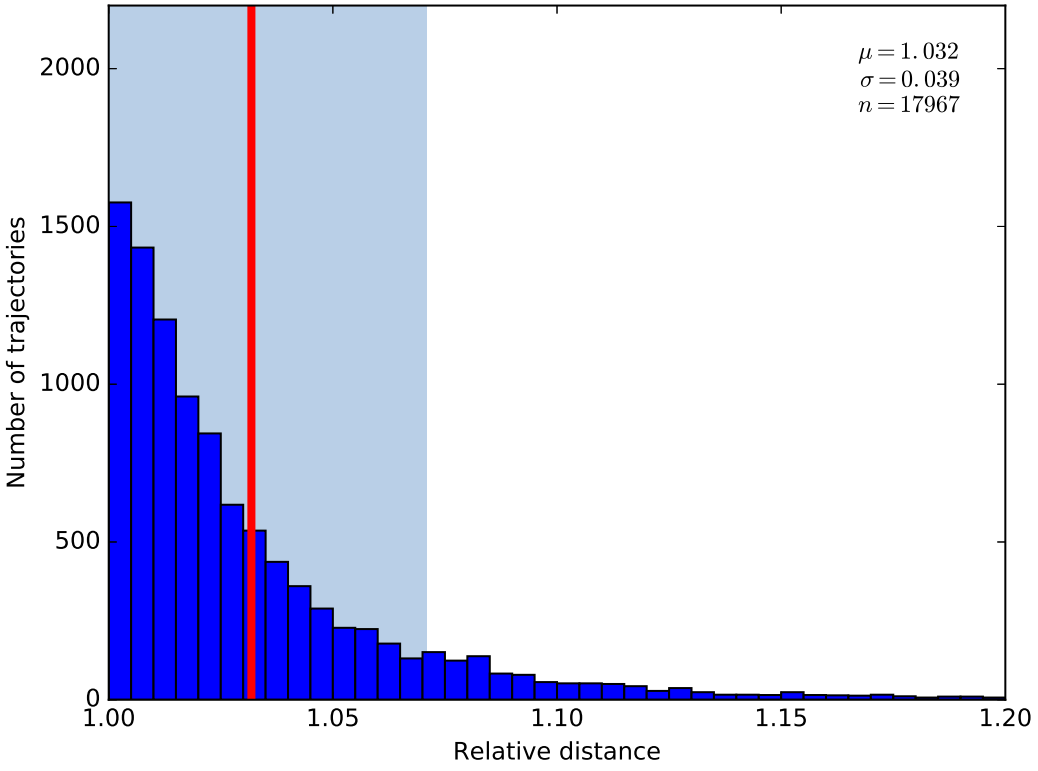


Figure 3.13: Scatterplot of length of unconstrained path vs constrained for all obstacle type images with Palágyi and Kuba skeleton and region growing reconstruction.

tograms can be found in Figures 3.14, 3.15, and 3.16 for obstacle type images and Figures 3.17, 3.18 and 3.19 for maze type images. These figures show that for sub-optimal planned trajectories, the distance that the planned trajectory exceeds the optimal trajectory follows a distribution that is clustered close to the origin. The mean and variance of these excess distances were found, with the lowest mean distance being 3.3% longer than optimal for the obstacle type image with the Palágyi skeleton and region growing segmentation, and 2.6% longer than optimal for the maze image and the seeded segmentation. For all conditions, the Palágyi and Kuba algorithm did not produce a mean path length that was greater than 2% longer than optimal.

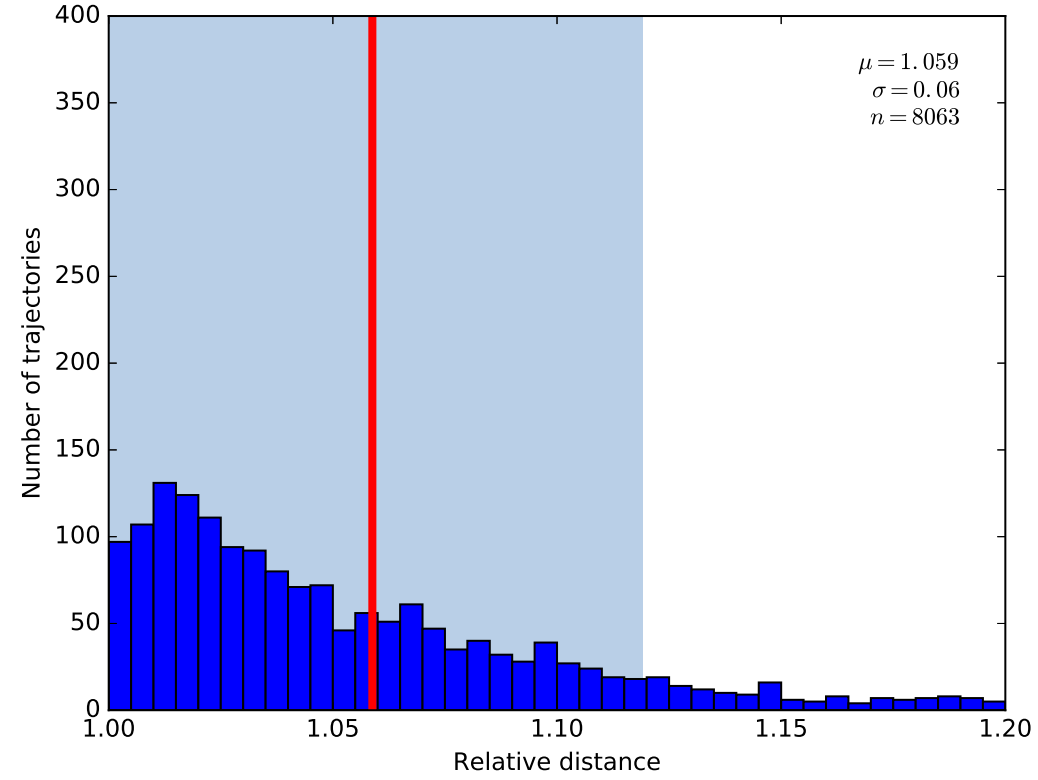


(a)

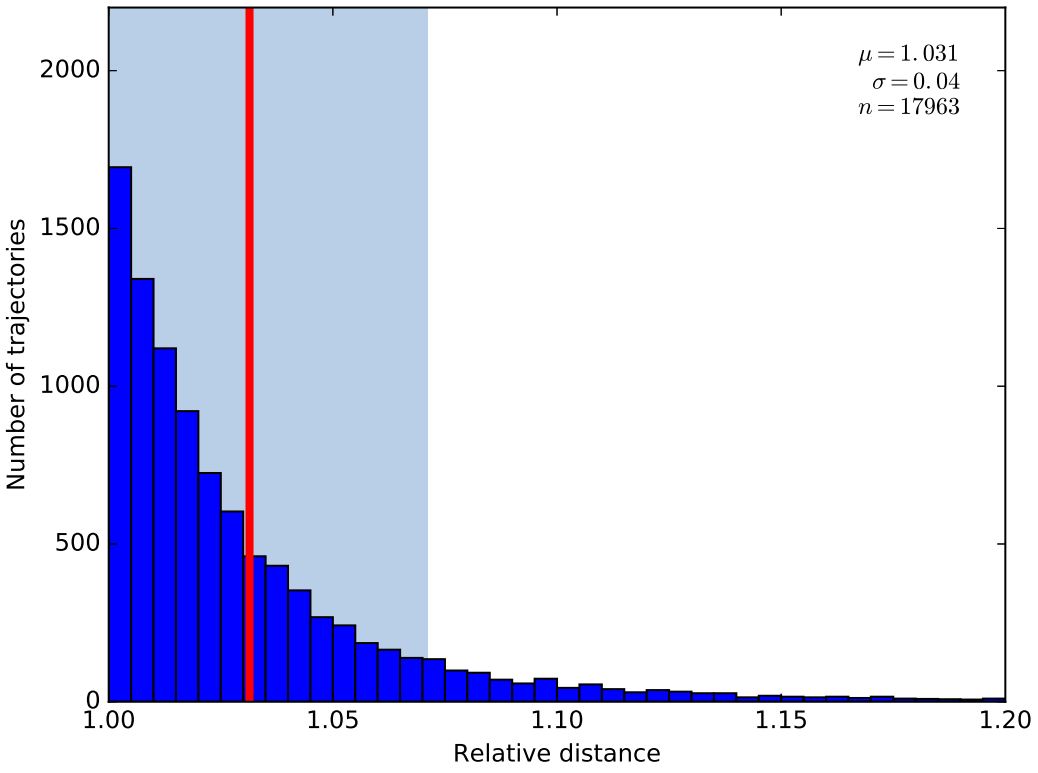


(b)

Figure 3.14: Histogram of relative distance in non-optimal paths for Lee et al. skeleton and region growing segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

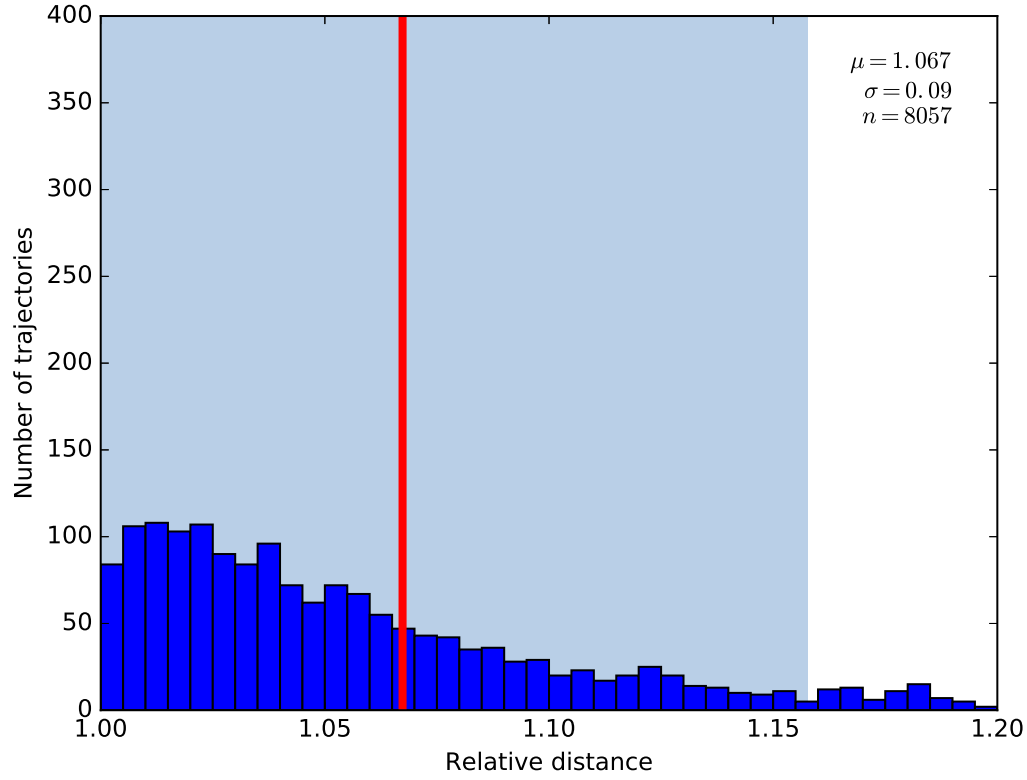


(a)

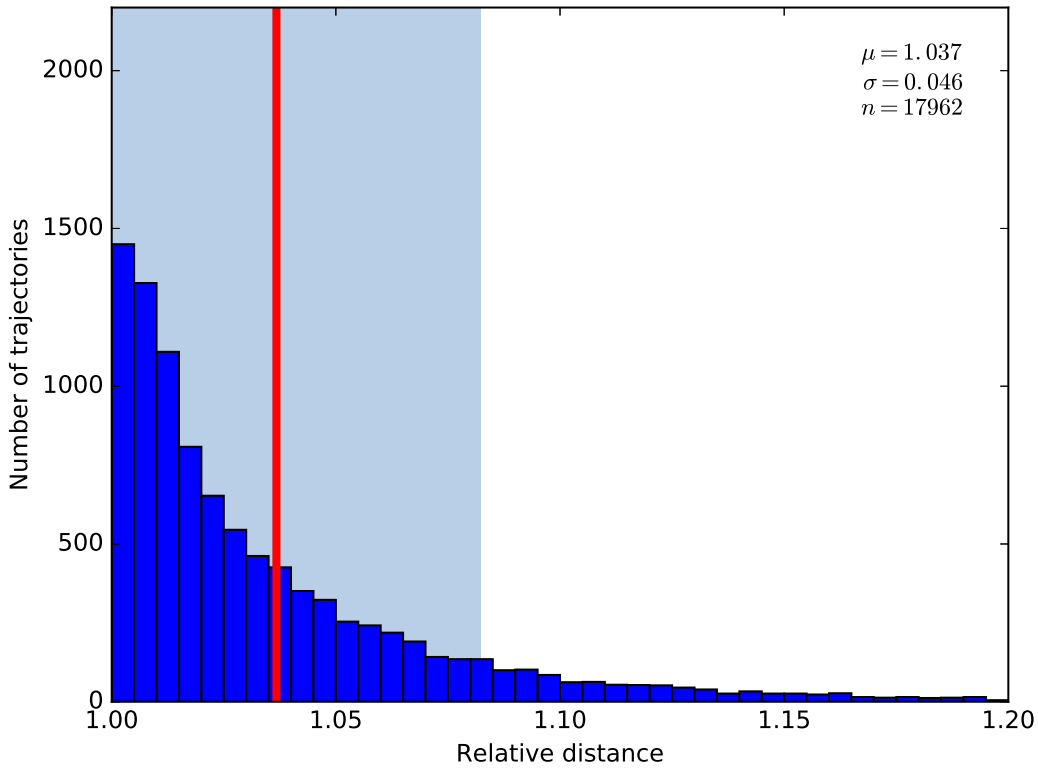


(b)

Figure 3.15: Histogram of relative distance in non-optimal paths for Lee et al. skeleton and seeded segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

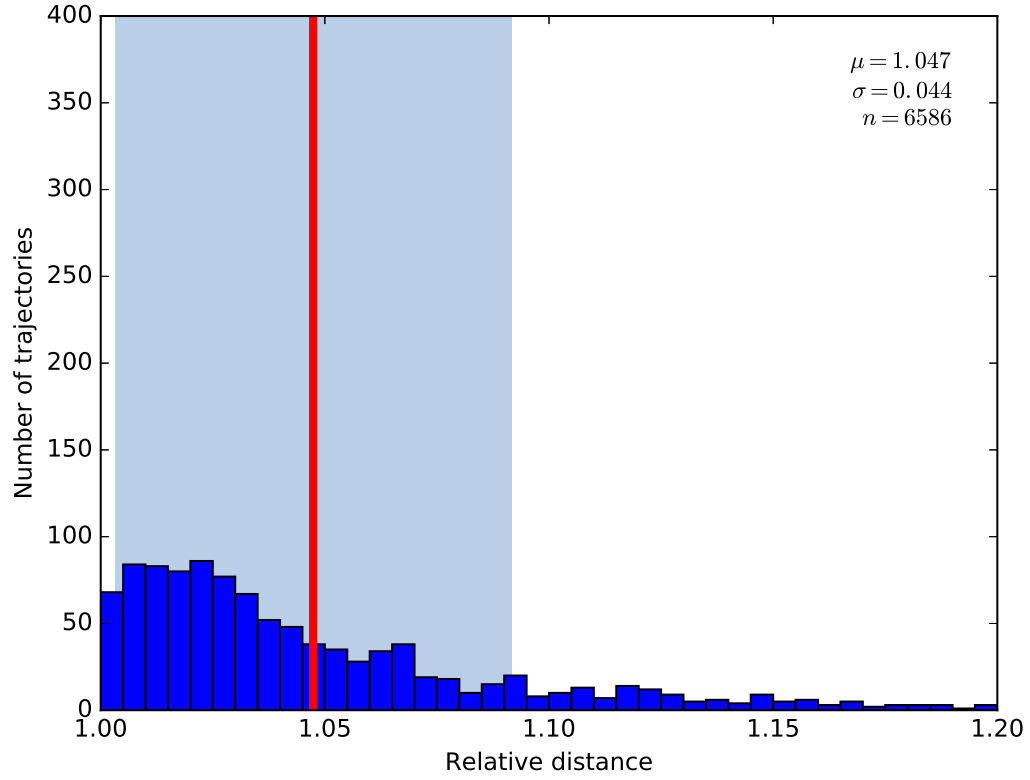


(a)

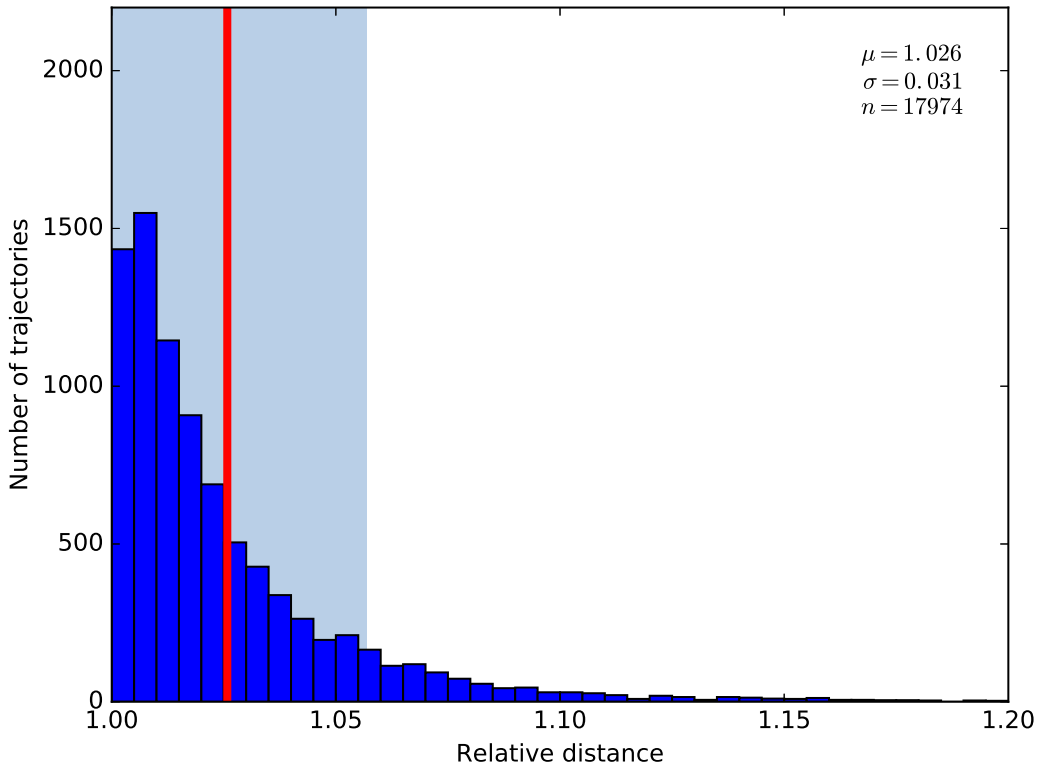


(b)

Figure 3.16: Histogram of relative distance in non-optimal paths for Lee et al. skeleton and watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

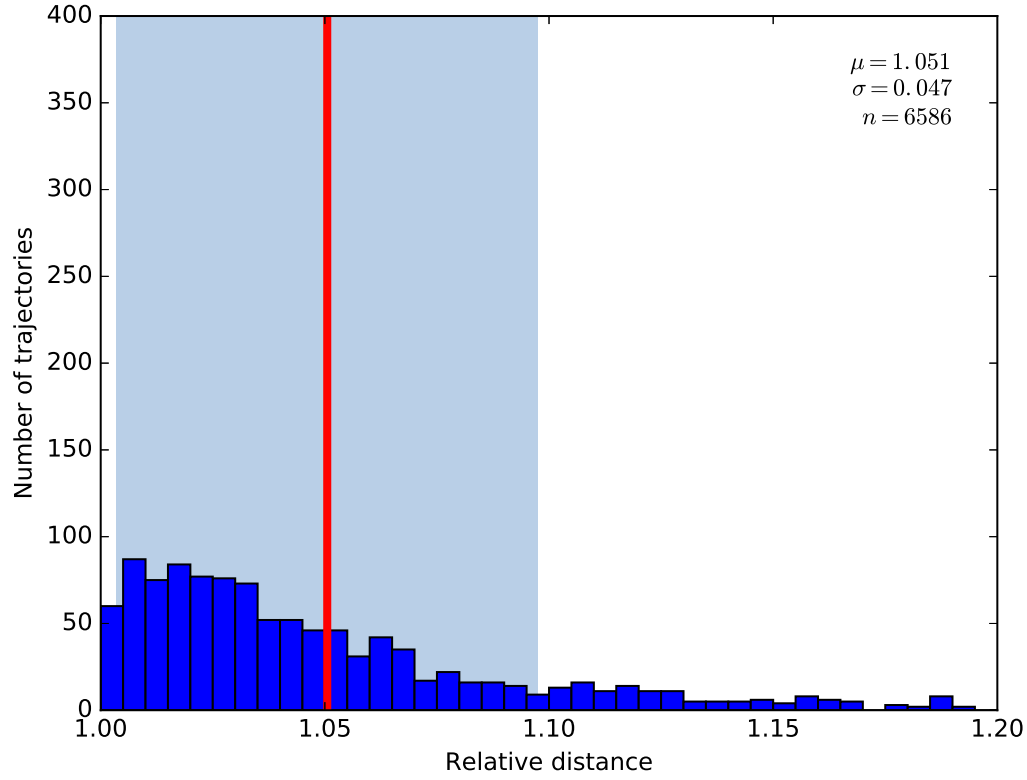


(a)

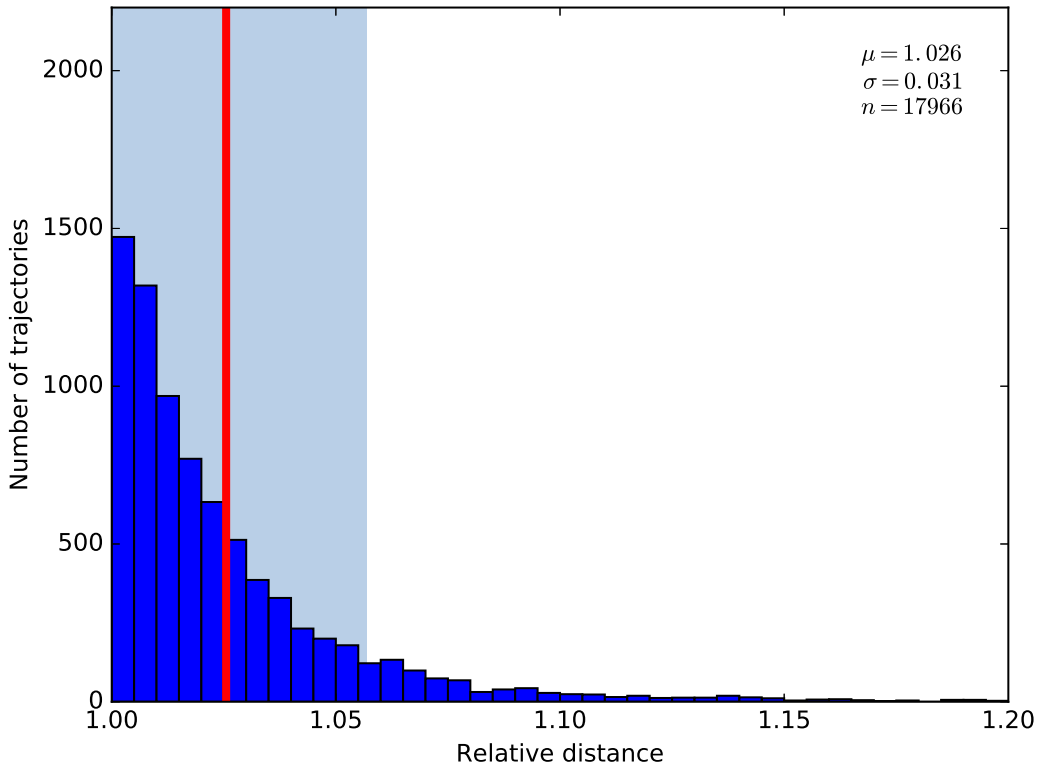


(b)

Figure 3.17: Histogram of relative distance in non-optimal paths for Palágyi and Kuba skeleton, and region growing segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

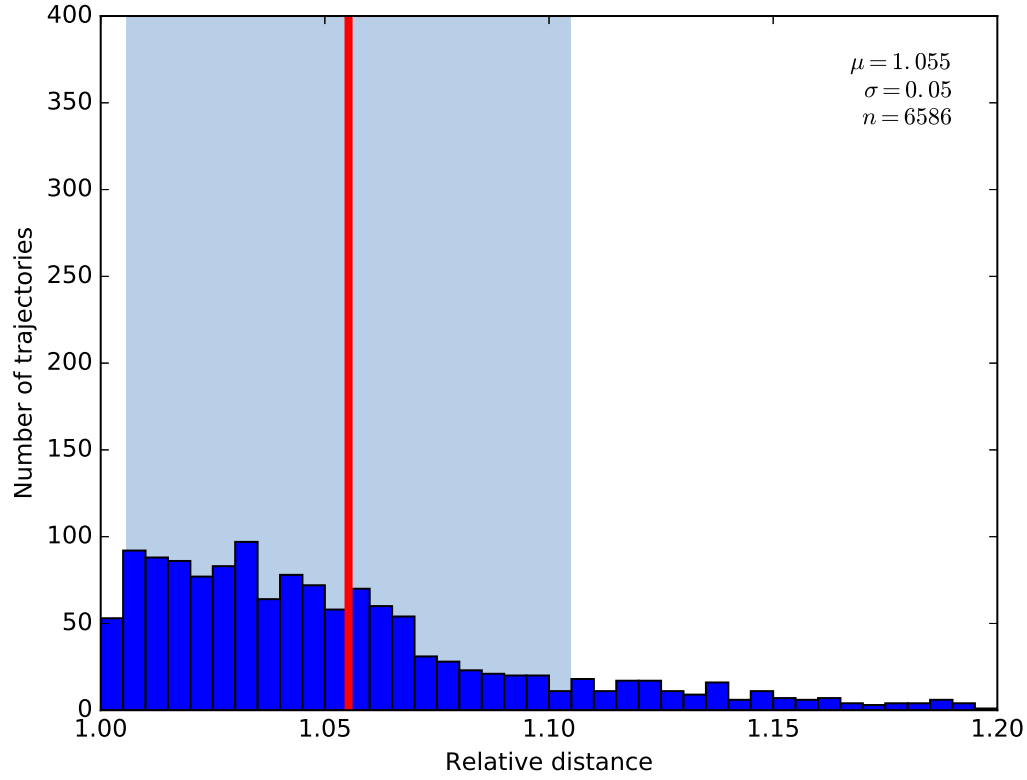


(a)

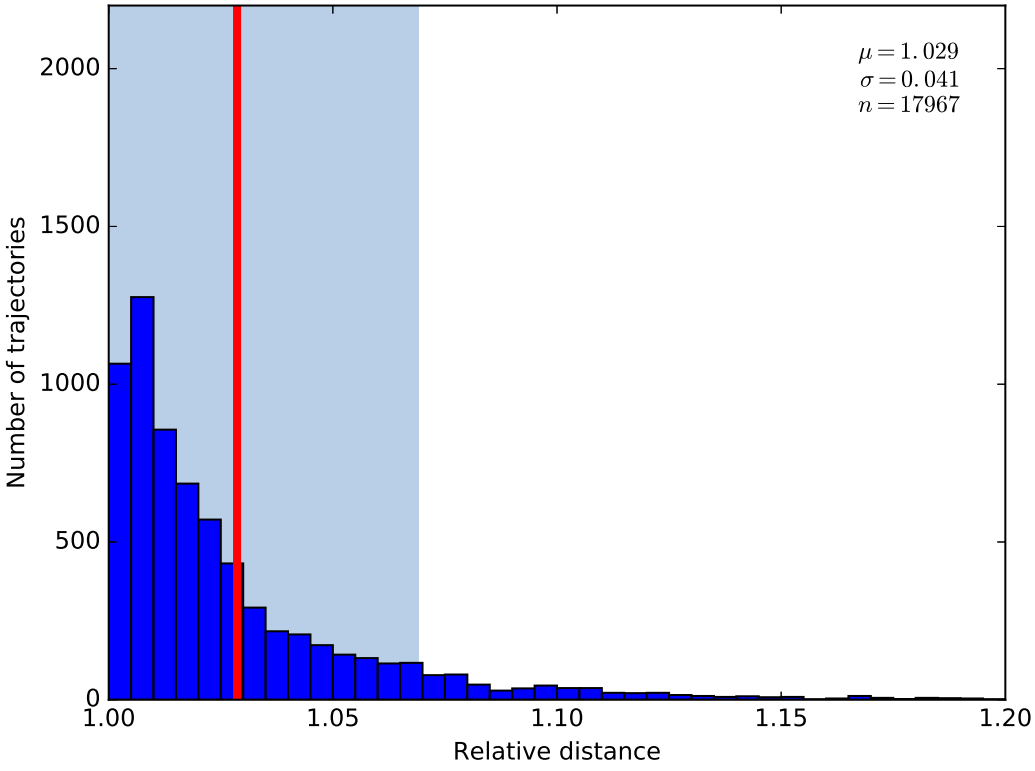


(b)

Figure 3.18: Histogram of relative distance in non-optimal paths for Palágyi and Kuba skeleton, and seeded segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

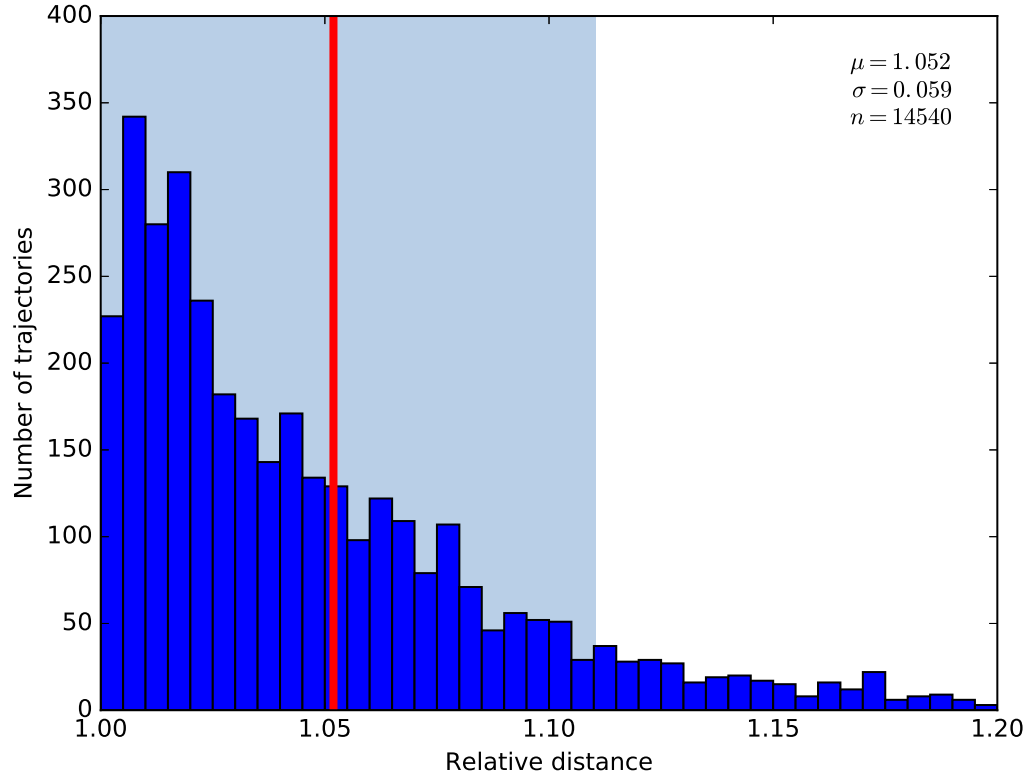


(a)

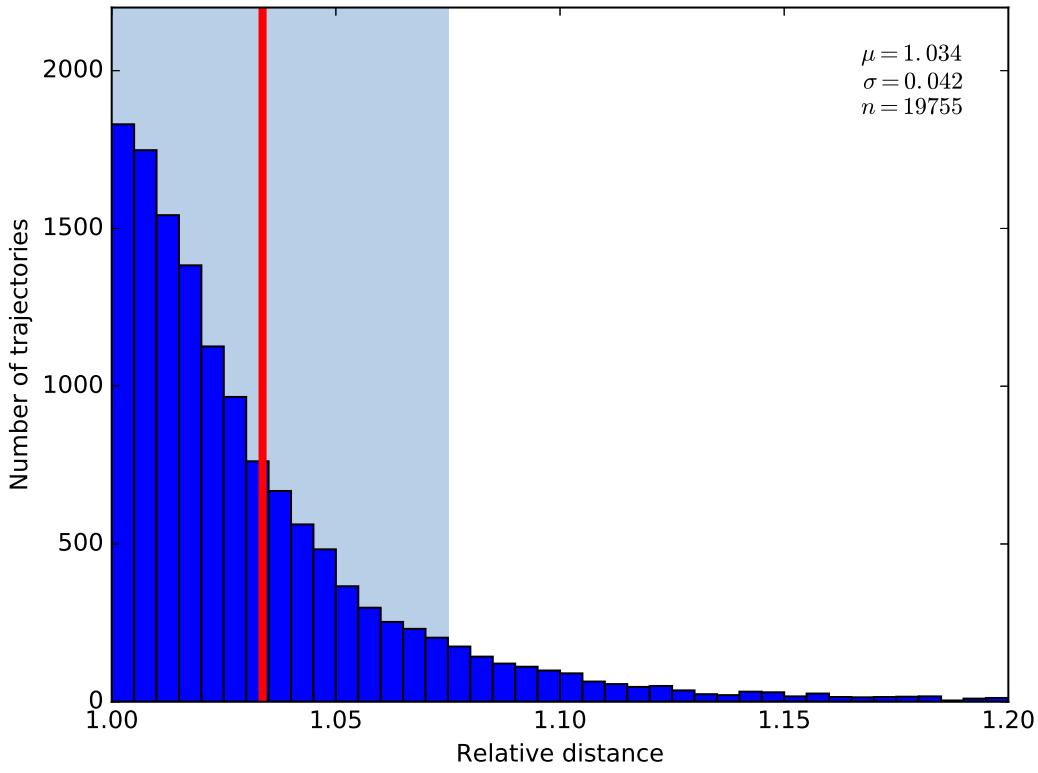


(b)

Figure 3.19: Histogram of relative distance in non-optimal paths for Palágyi and Kuba skeleton, and seeded segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

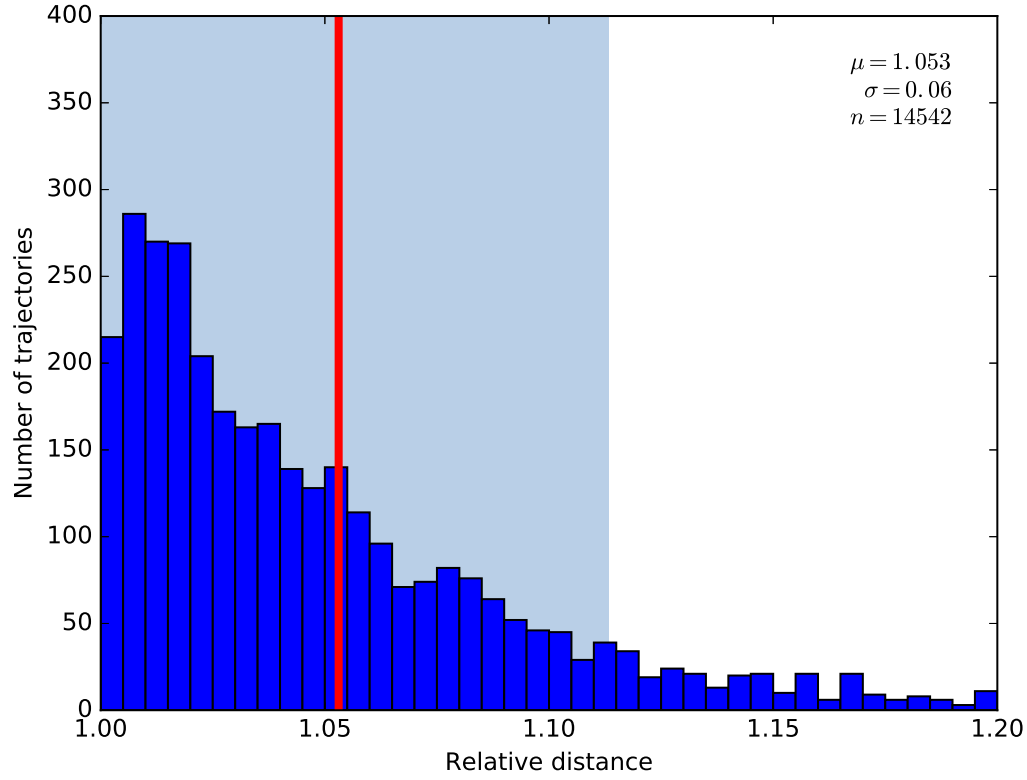


(a)

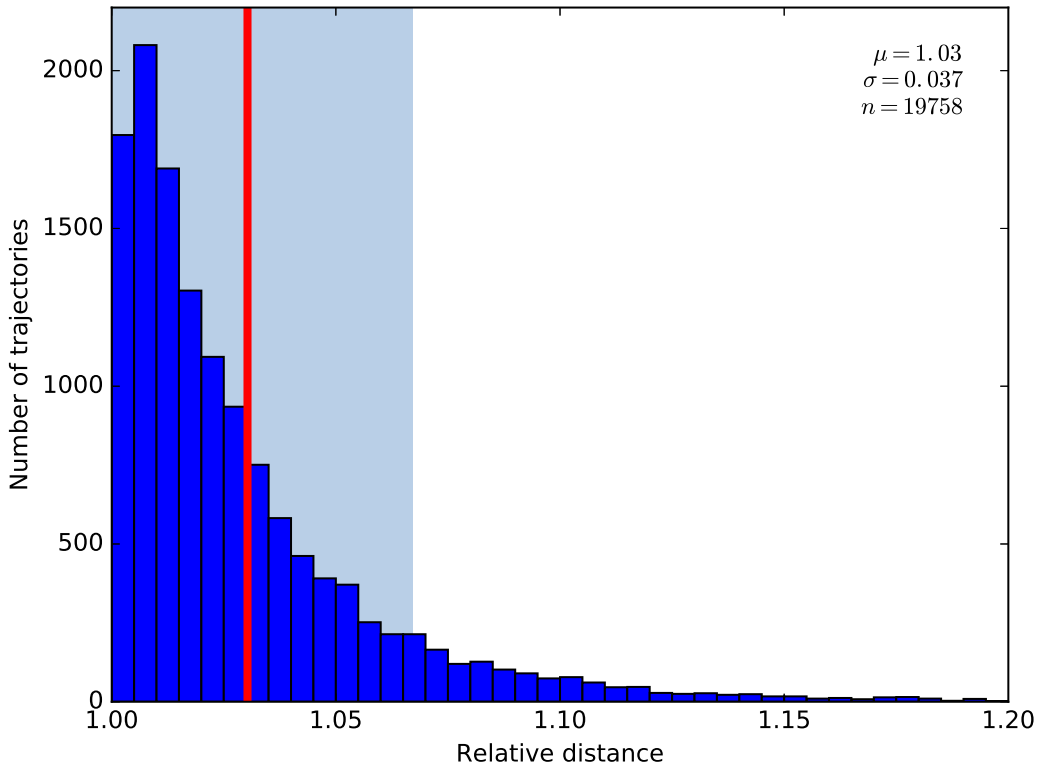


(b)

Figure 3.20: Histogram of relative distance in non-optimal paths for medial axis transform and region growing segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

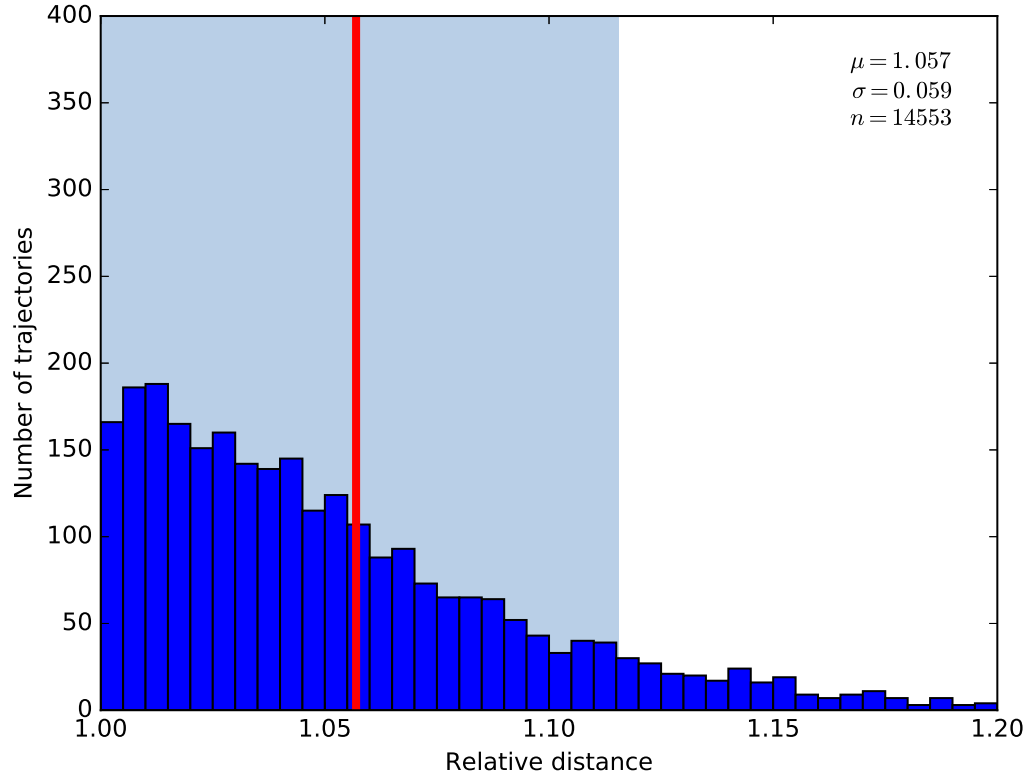


(a)

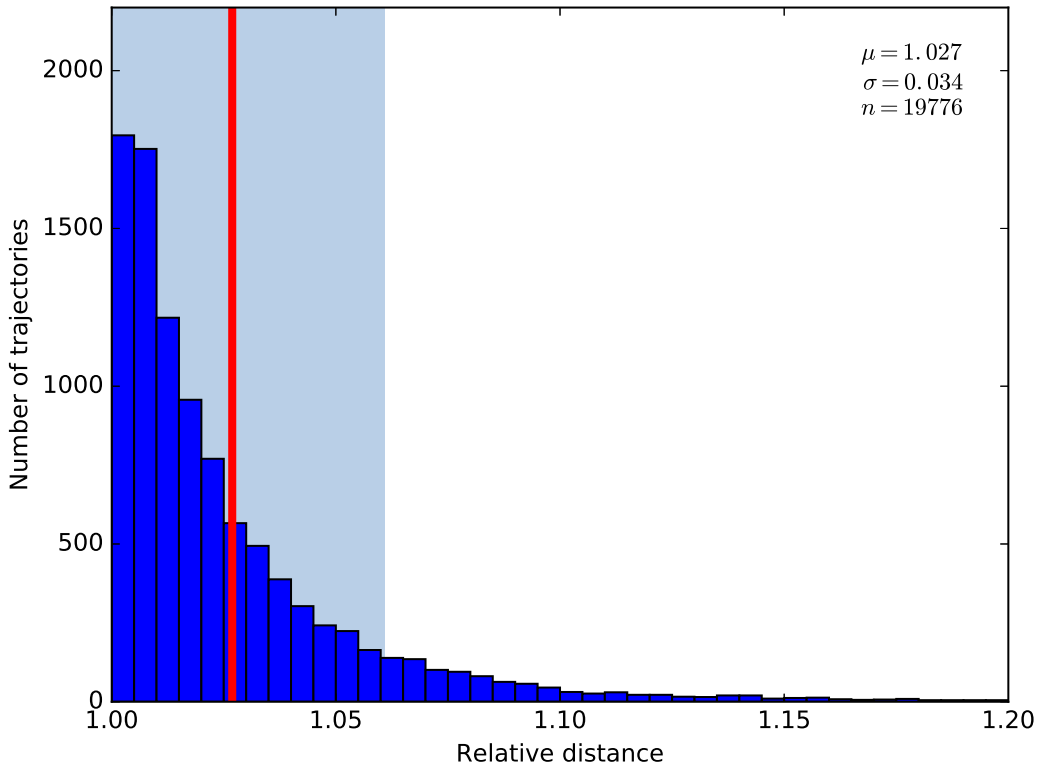


(b)

Figure 3.21: Histogram of relative distance in non-optimal paths for medial axis transform and seeded segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.



(a)



(b)

Figure 3.22: Histogram of relative distance in non-optimal paths for medial axis transform and watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) All obstacle images. (b) All maze images.

3.2 Planning Efficiency in Real World Data

The synthetic examples in the previous sections demonstrated the properties of the skeletisation algorithms, but testing for a real-world system should include real-world data. Naturally occurring spaces are not constrained to appear as geometric shapes. An example of this is the multi-beam bathymetric survey performed of Apra Harbour in Guam. This area includes a number of features including shoals, a drydock and several wrecks [Pacific Islands Benthic Habitat Mapping Center (PIBHMC) et al., 2010]. Human actions have imposed regular shapes upon the natural environment, but the border of the harbour area is still irregular. Some of the shoals are regular in shape, but the western shoals are irregular.

A map with 1m grid cells generated from the Apra Harbour map can be seen in Figure 3.23. This bathymetry data was converted into a pair of images, one covering the entire harbour and another centered on the central shoal area east of the dry dock. Figure 3.24 shows the area based on the harbour bathymetry while Figures 3.25 to 3.27 show the skeletisations of this image. Figure 3.28 shows an extract of the data centered around the shoals. Skeletisations of this image can be seen in Figures 3.29 to 3.31. As shown in Table 3.7, the Palágyi skeleton produced a smaller number of nodes. As with the earlier maze and obstacle images, the number of optimal trajectories were tabulated and can be seen in Table 3.8. Notably the Palágyi skeleton produced optimal trajectories for all trials on the shoals image while still outperforming all other skeleton types for the full harbour image. Histograms of the residual distances generated by the Lee and medial skeletons for the shoals image can be seen in Figures 3.32 to 3.35, while the residuals for the full Apra Harbour image can be seen in Figures 3.36 to 3.41.

All skeletisation and segmentation types produced good results for the Apra shoals image, since this contains obstacles that are all close to the centreline. The larger Apra image was more challenging to the algorithms due to containing a number of small obstacles close to the coastline. The Palágyi and Kuba algorithm outperformed the other skeletisation algorithms in this situation, producing optimal trajectories more than half the time for all segmentation types.

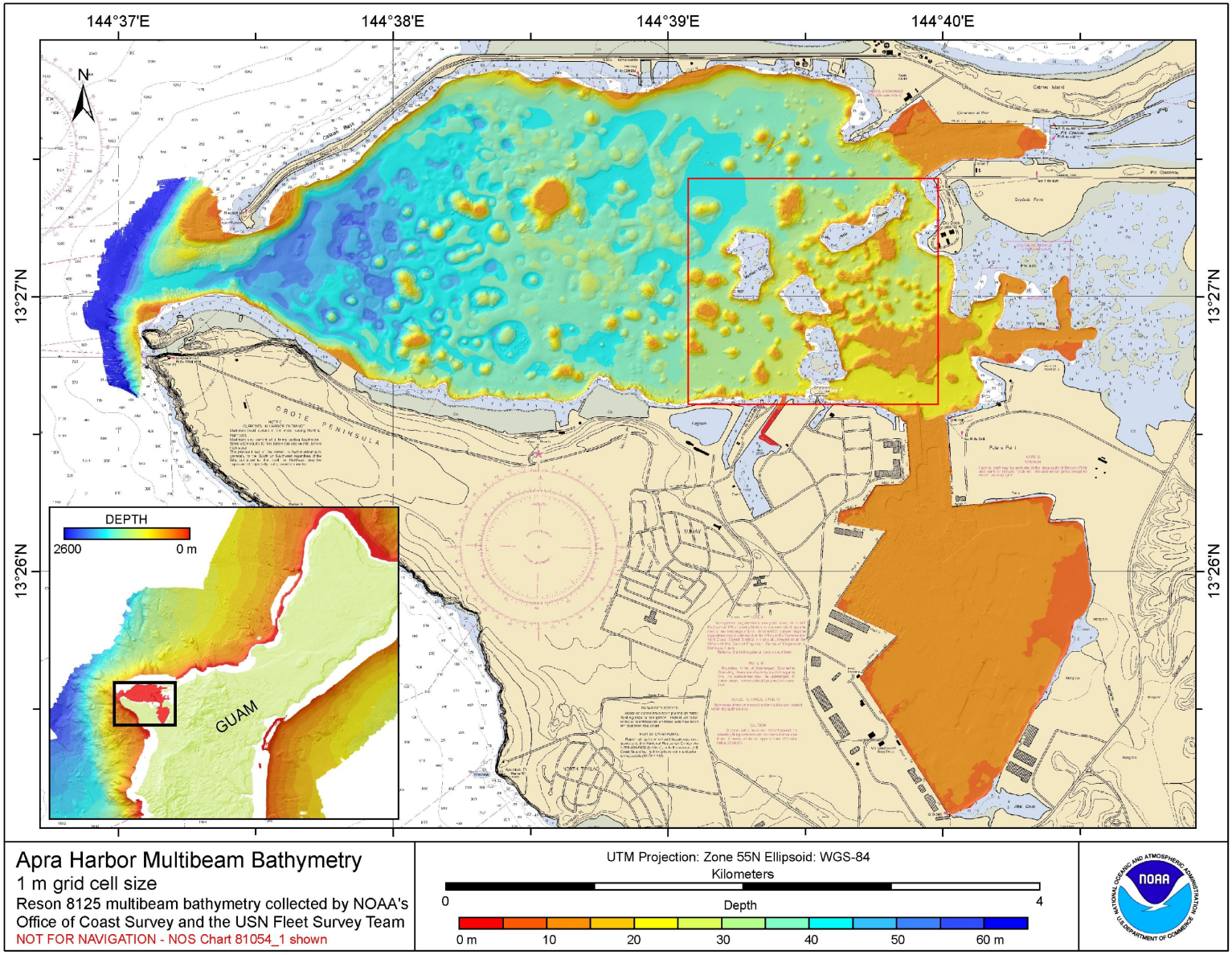


Figure 3.23: Apra Harbour bathymetry as captured in 2008 [Pacific Islands Benthic Habitat Mapping Center (PIBHMC) et al., 2010]. A red box has been added to mark the shoals used for the second harbour image.

Table 3.7: Number of generated nodes for the Apra Harbour and Apra shoals maps. Total number of trials performed, $n = 2$.

Skeletisation	Number of nodes	
	Apra Harbour	Apra shoals
Lee et al.	386	57
Palágyi and Kuba	114	10
Medial Axis Transform	411	70

Table 3.8: Proportion of optimal trajectories for Apra Harbour extract. Total number of trials performed, $n = 14006$.

Skeletisation	Segmentation	Proportion of optimal trajectories	
		Apra Harbour	Apra shoals
Lee et al.	Region Growing	0.21	0.39
	Seeded	0.31	0.48
	Watershed	0.45	0.63
Palágyi and Kuba	Region Growing	0.51	1.0
	Seeded	0.53	1.0
	Watershed	0.63	1.0
Medial Axis Transform	Region Growing	0.17	0.38
	Seeded	0.20	0.39
	Watershed	0.47	0.68

Figure 3.24: Input image of Apra harbour. Dataset was generated from Bathymetry data shown in Figure 3.23.

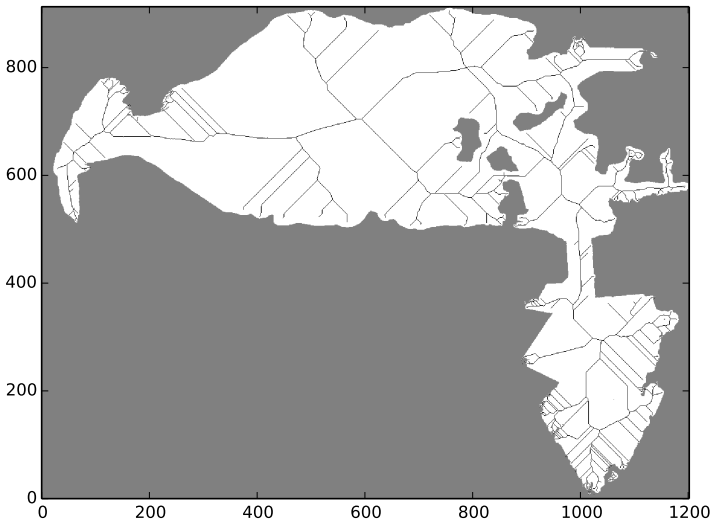


Figure 3.25: Skeletisation of Apra harbour image from Figure 3.24 using the Lee et al. algorithm.

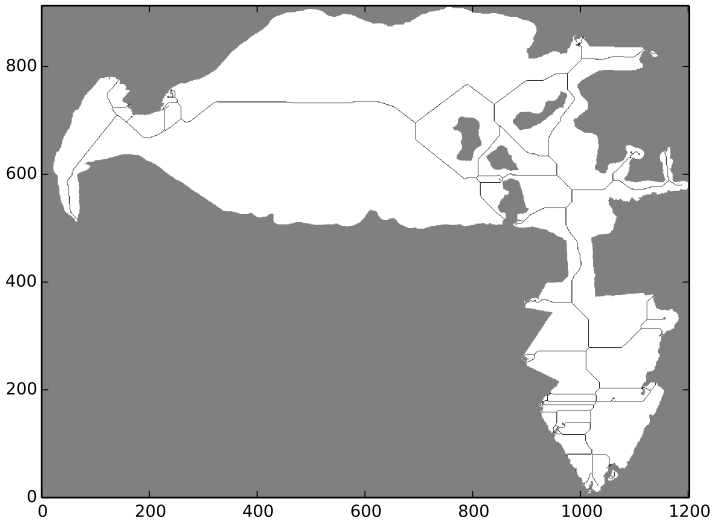


Figure 3.26: Skeletisation of Apra harbour image from Figure 3.24 using the Palágyi and Kuba algorithm

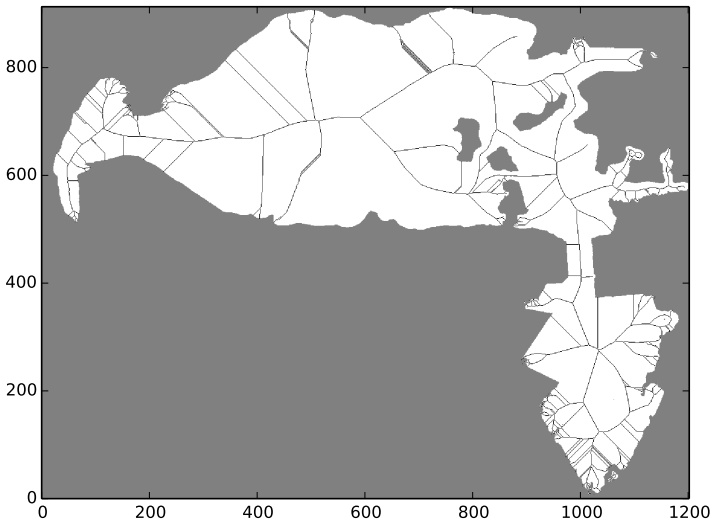


Figure 3.27: Skeletisation of Apra harbour image from Figure 3.24 using the medial axis transform algorithm.

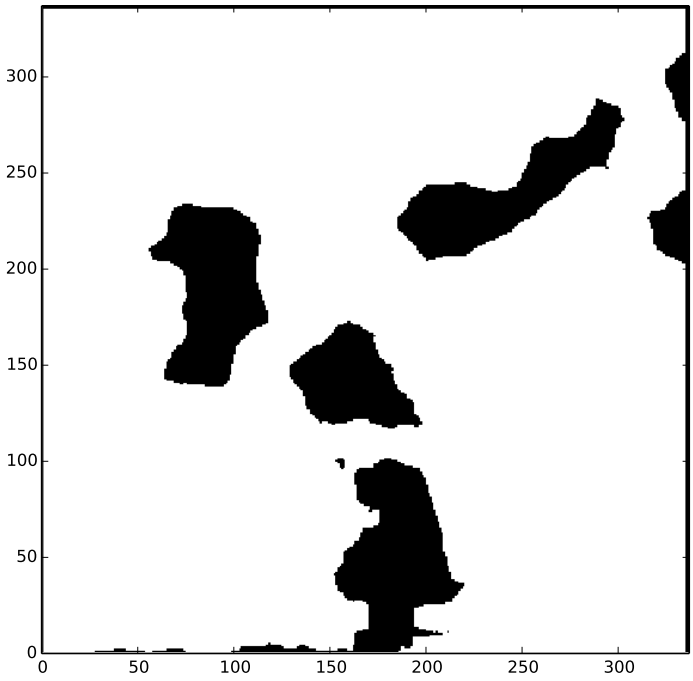


Figure 3.28: Segment of Apra harbour with the four central shoals east of the drydock area. This area is marked in red in Figure 3.23

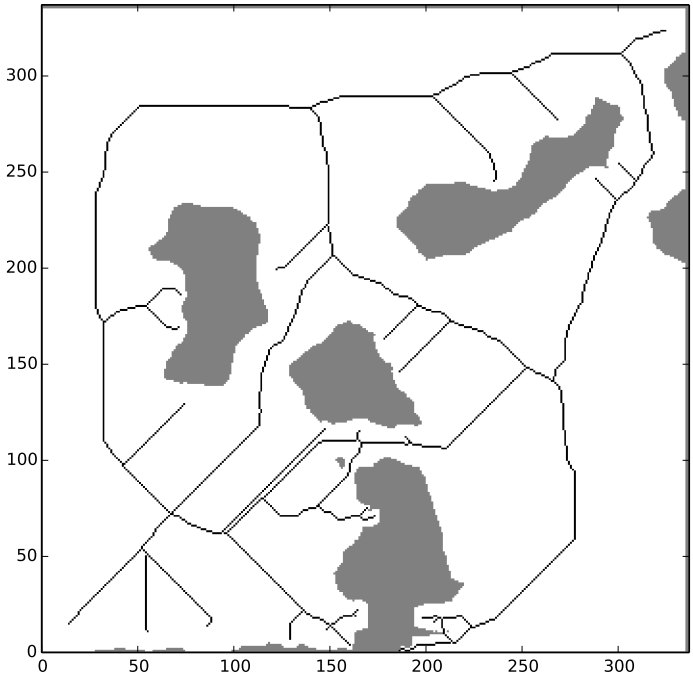


Figure 3.29: Skeletisation of Apra shoals image from Figure 3.28 using the Lee et al. algorithm.

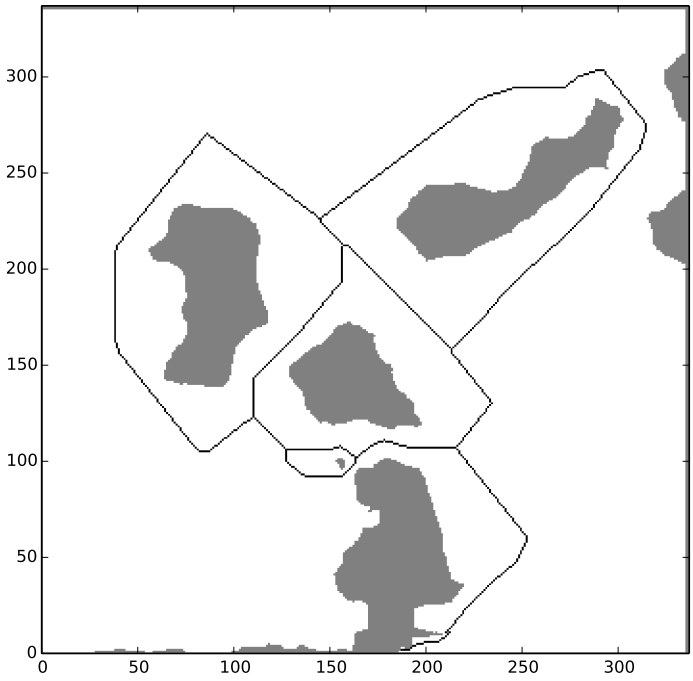


Figure 3.30: Skeletisation of Apra shoals image from Figure 3.28 using the Palágyi and Kuba algorithm.

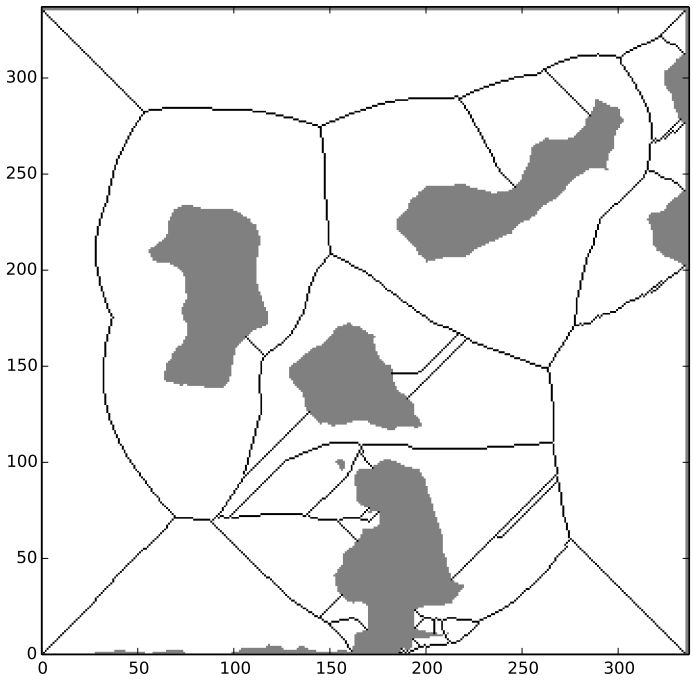
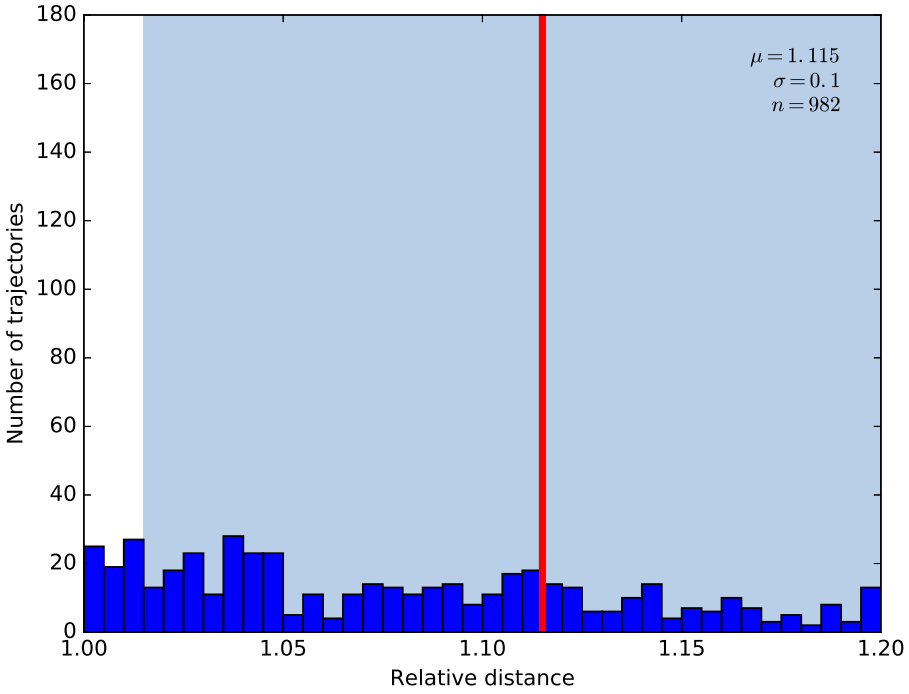
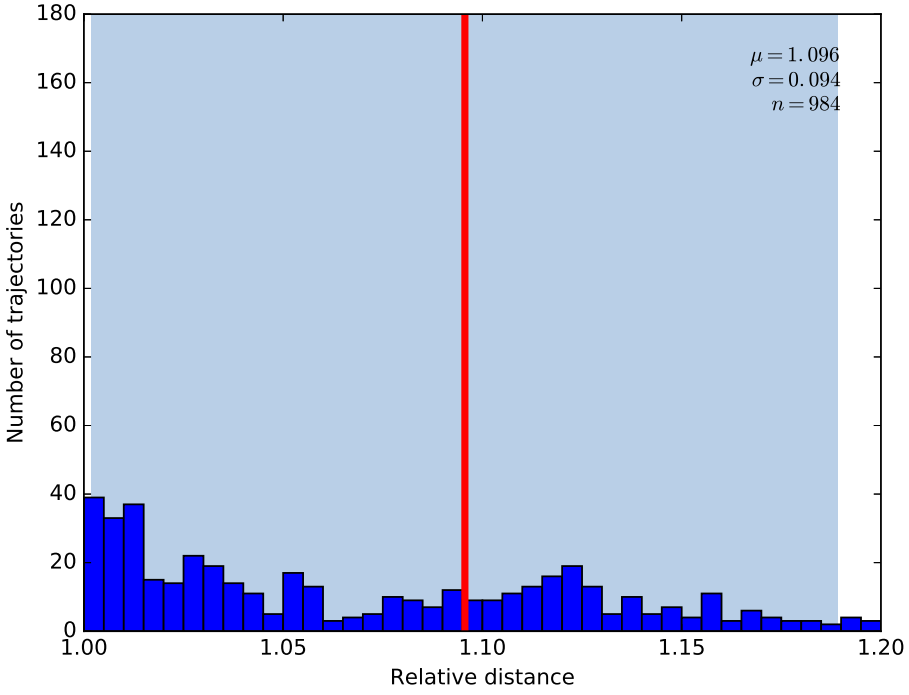


Figure 3.31: Skeletisation of Apra shoals image from Figure 3.28 using the medial axis transform algorithm.



(a)



(b)

Figure 3.32: Histogram of relative distance in non-optimal paths for Lee and medial skeletons and the Apra Harbour shoals. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. No residuals are included for the Palágyi skeleton since all paths were optimal. (a) Lee et. al. skeleton, region growing segmentation. (b) Lee et. al. skeleton, seeded segmentation.

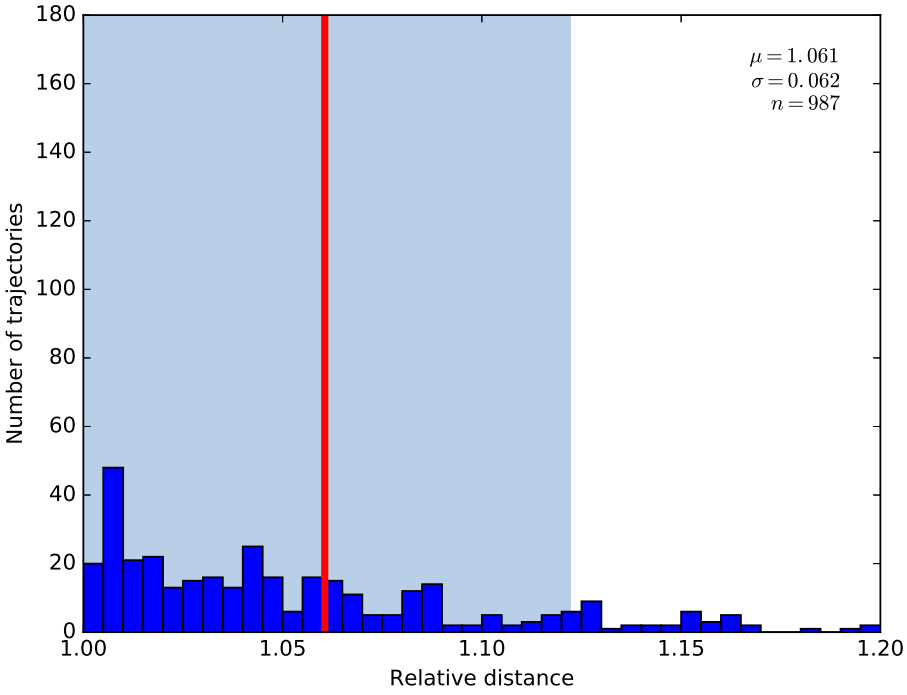
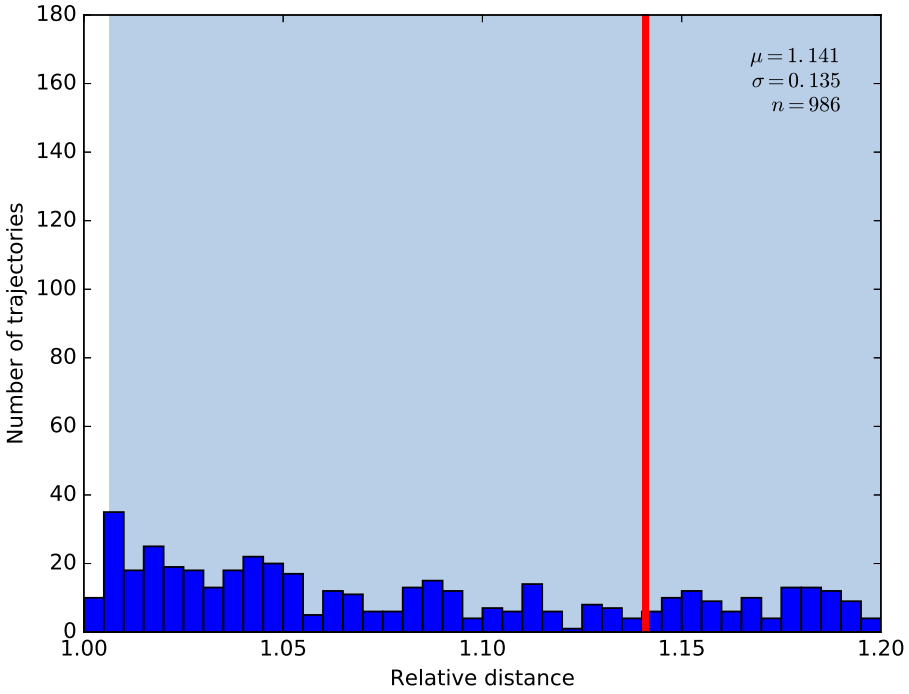
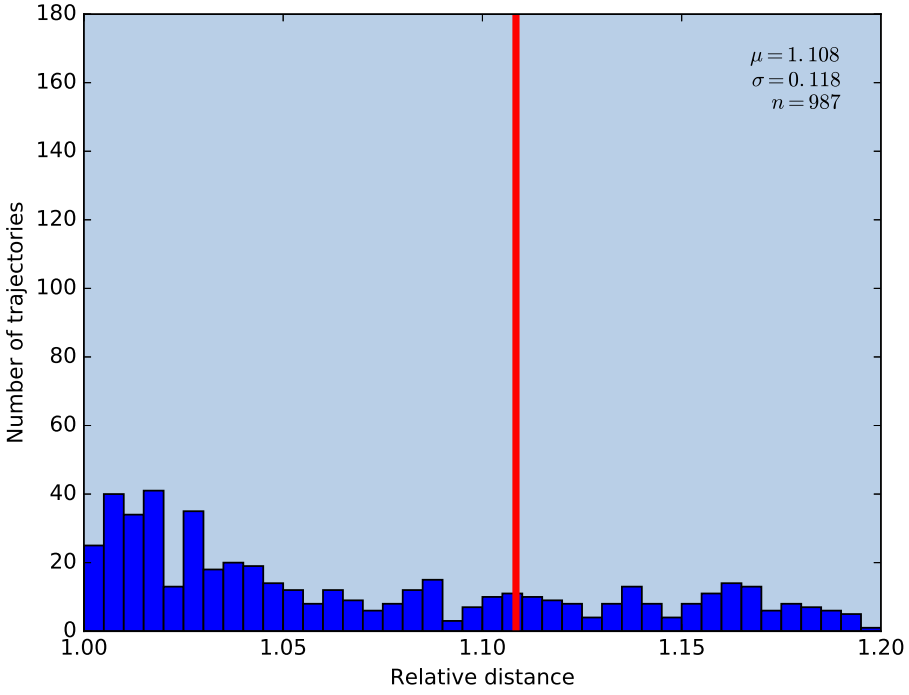


Figure 3.33: Histogram of relative distance in non-optimal paths for Lee et. al. skeleton and the Apra Harbour shoals using watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. No residuals are included for the Palágyi skeleton since all paths were optimal.

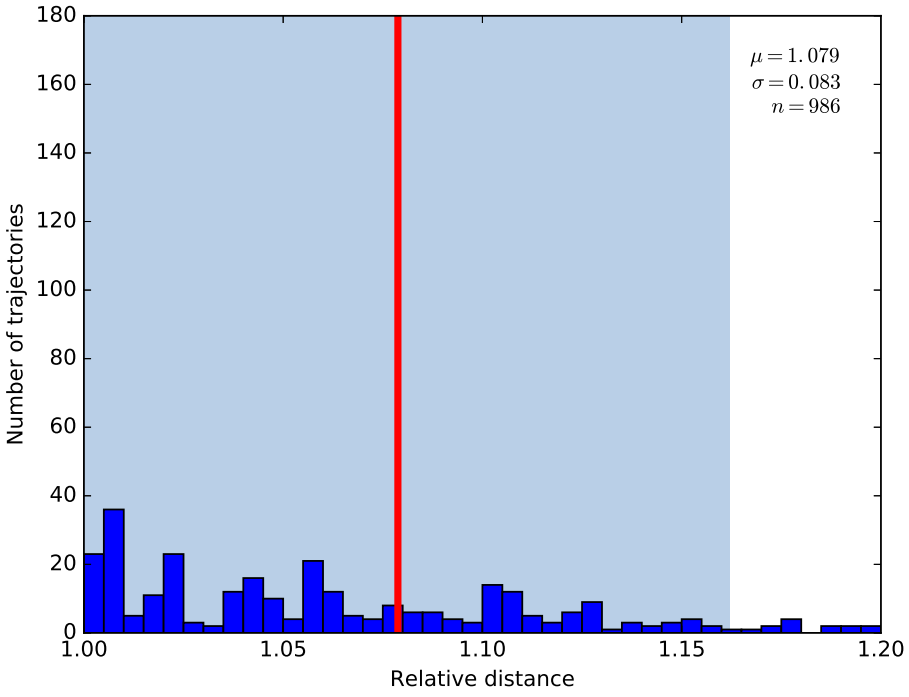


(a)



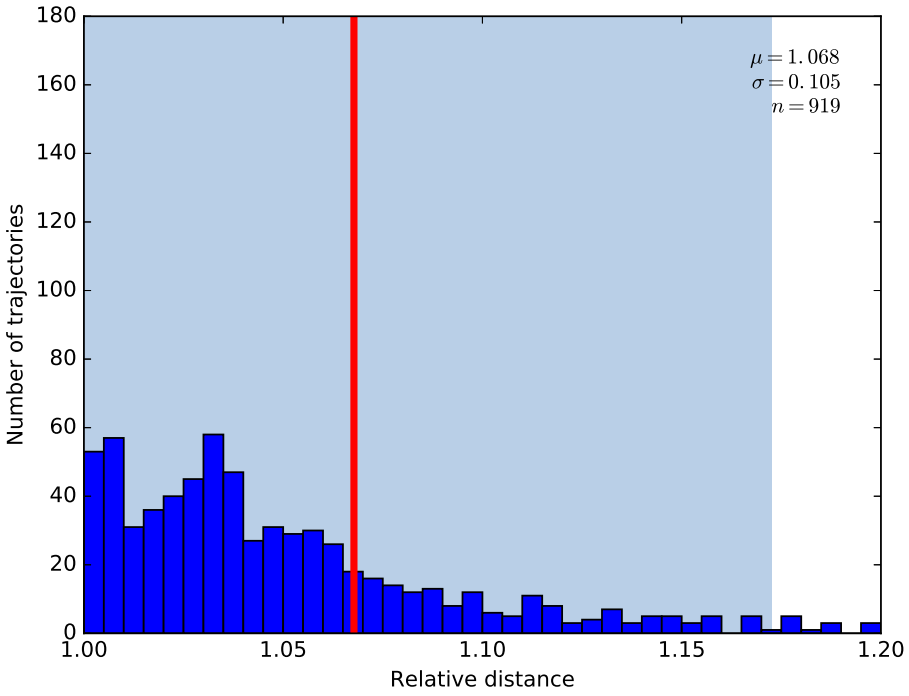
(b)

Figure 3.34: Histogram of relative distance in non-optimal paths for Lee and medial skeletons and the Apra Harbour shoals. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. No residuals are included for the Palágyi skeleton since all paths were optimal. (a) Medial axis transform, region growing segmentation. (b) Medial axis transform, seeded segmentation.

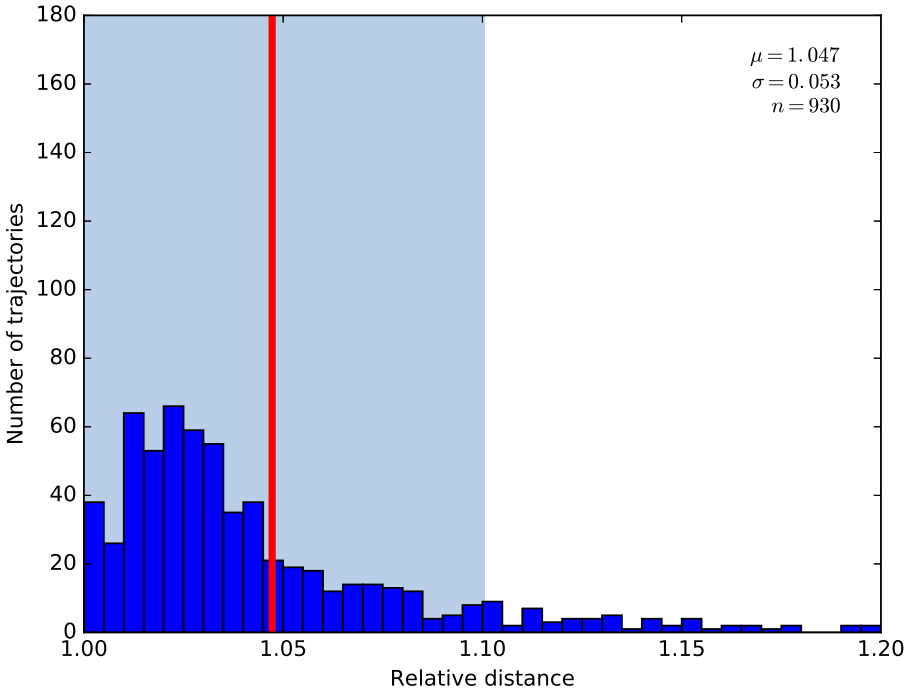


(a)

Figure 3.35: Histogram of relative distance in non-optimal paths for medial axis transform and the Apra Harbour shoals using watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. No residuals are included for the Palágyi skeleton since all paths were optimal.



(a)



(b)

Figure 3.36: Histogram of relative distance in non-optimal paths for all skeletons in the Apra Harbour image. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) Lee et. al. skeleton, region growing segmentation. (b) Lee et. al. skeleton, seeded segmentation.

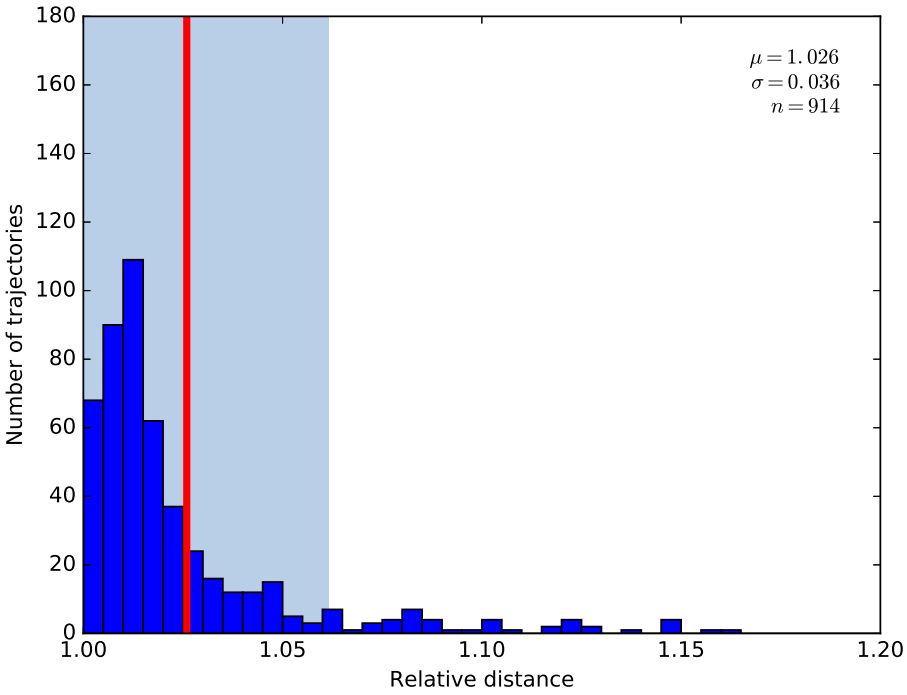
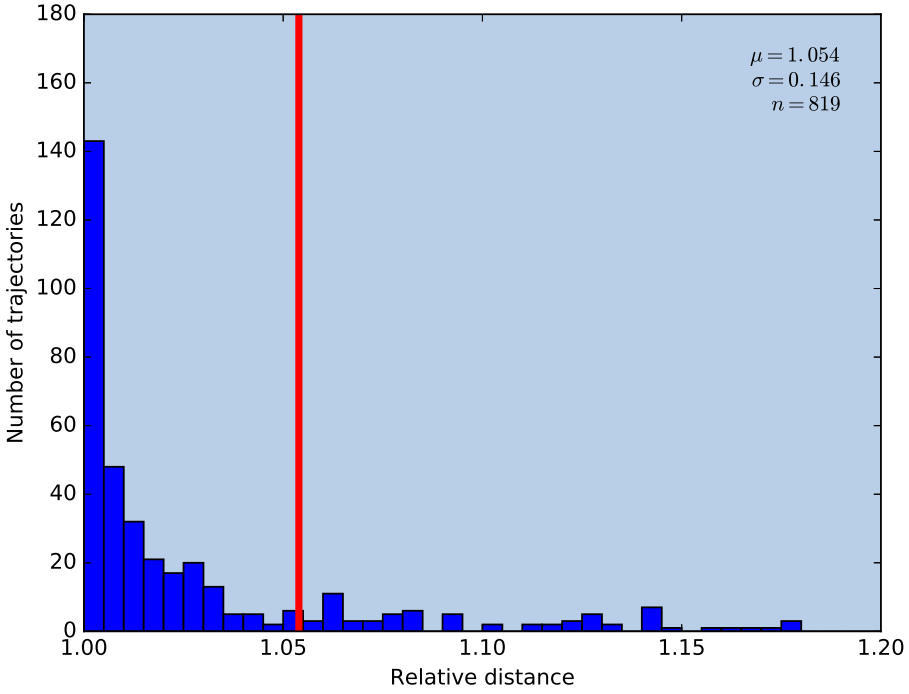
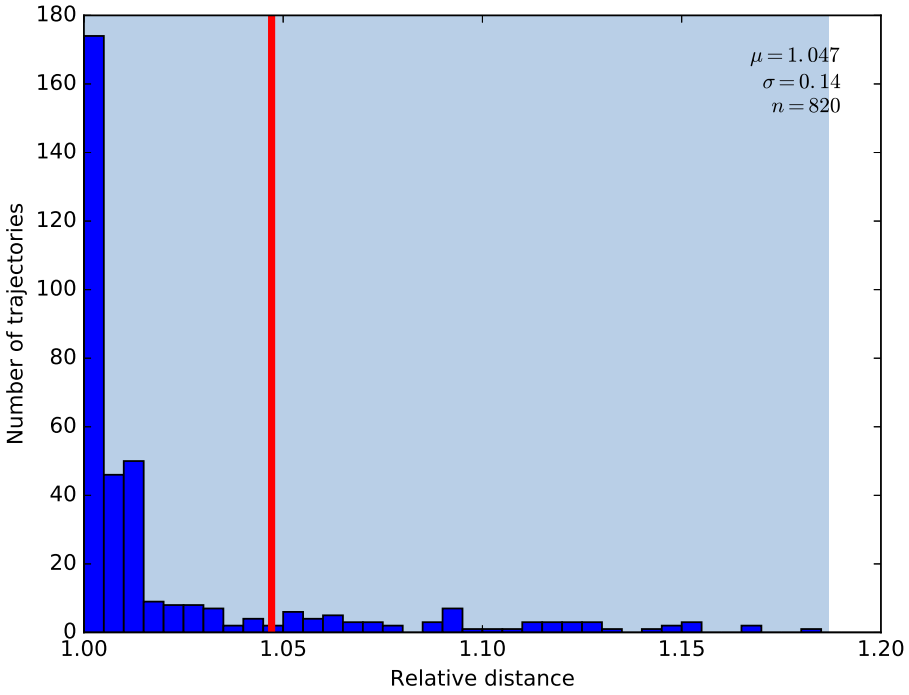


Figure 3.37: Histogram of relative distance in non-optimal paths for the Apra Harbour image using the Lee et. al. skeleton and watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean.



(a)



(b)

Figure 3.38: Histogram of relative distance in non-optimal paths for the Apra Harbour image. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) Palágyi and Kuba skeleton, region growing segmentation (b) Palágyi and Kuba skeleton, seeded segmentation.

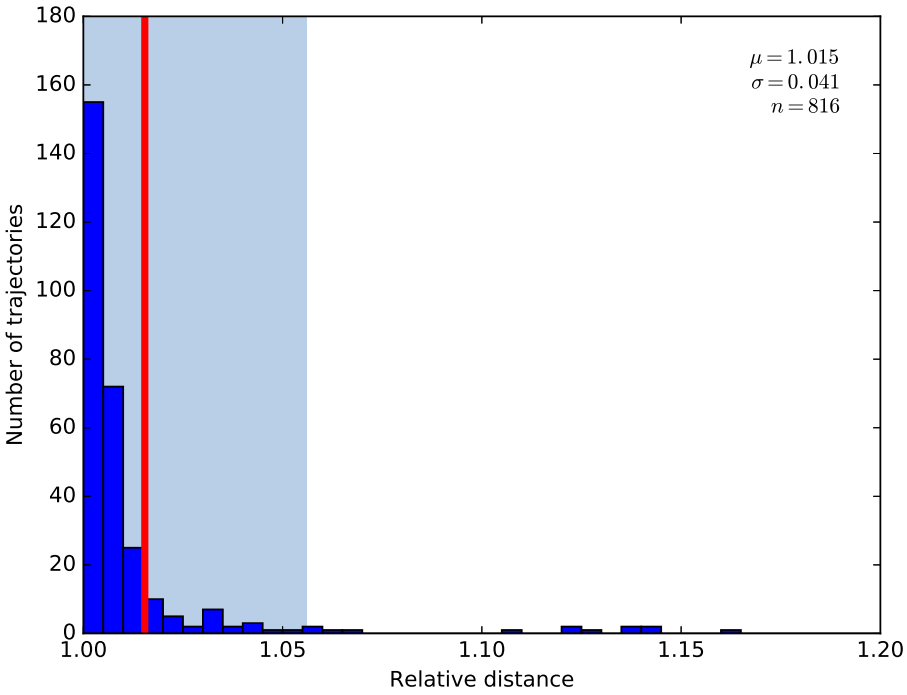
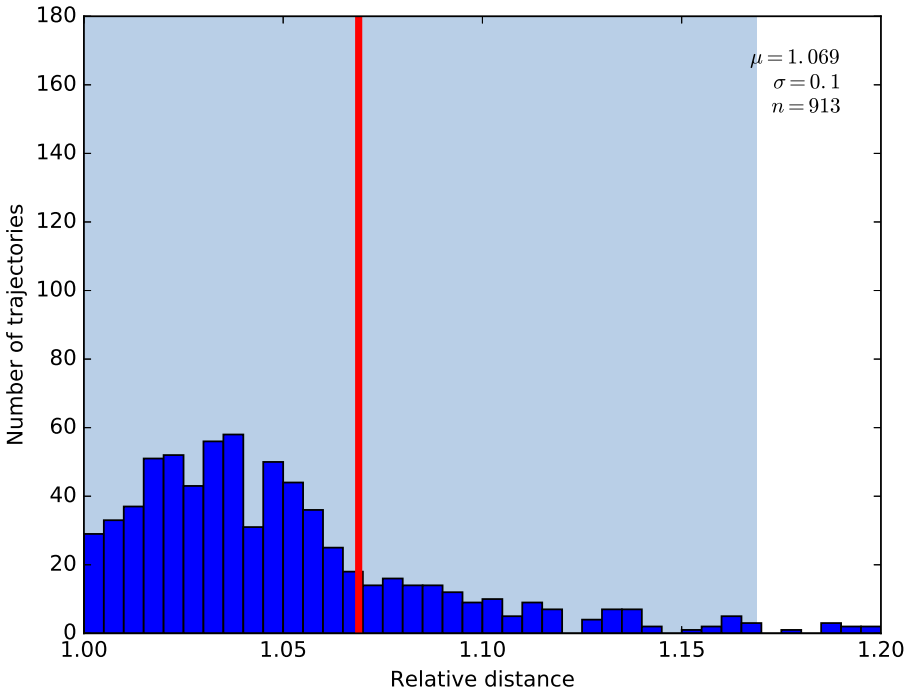
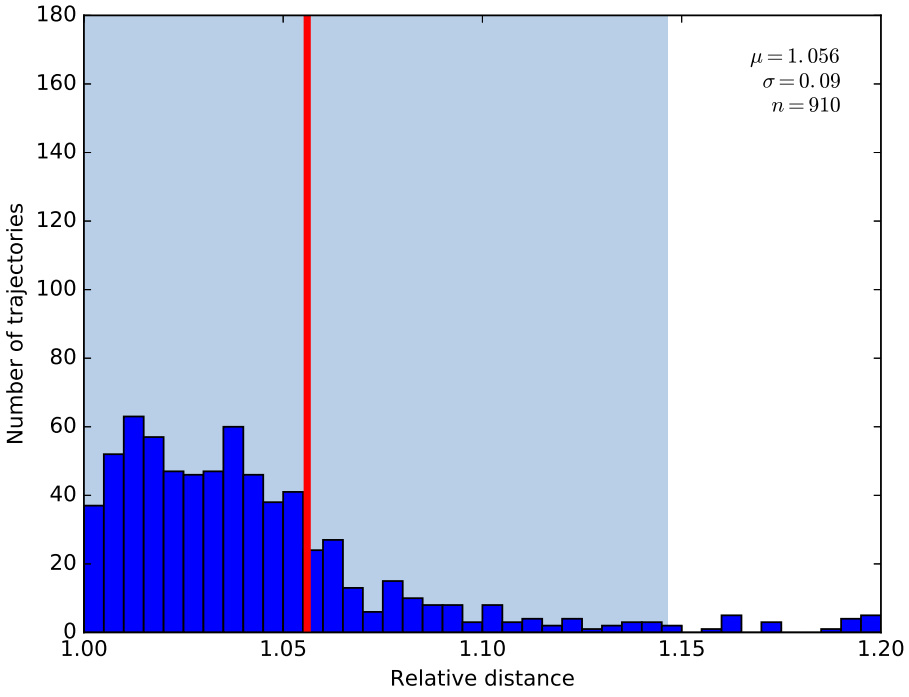


Figure 3.39: Histogram of relative distance in non-optimal paths for the Apra Harbour image using the Palágyi and Kuba skeleton and watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean.



(a)



(b)

Figure 3.40: Histogram of relative distance in non-optimal paths for the Apra Harbour image. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean. (a) Medial axis transform, region growing segmentation. (b) Medial axis transform, seeded segmentation.

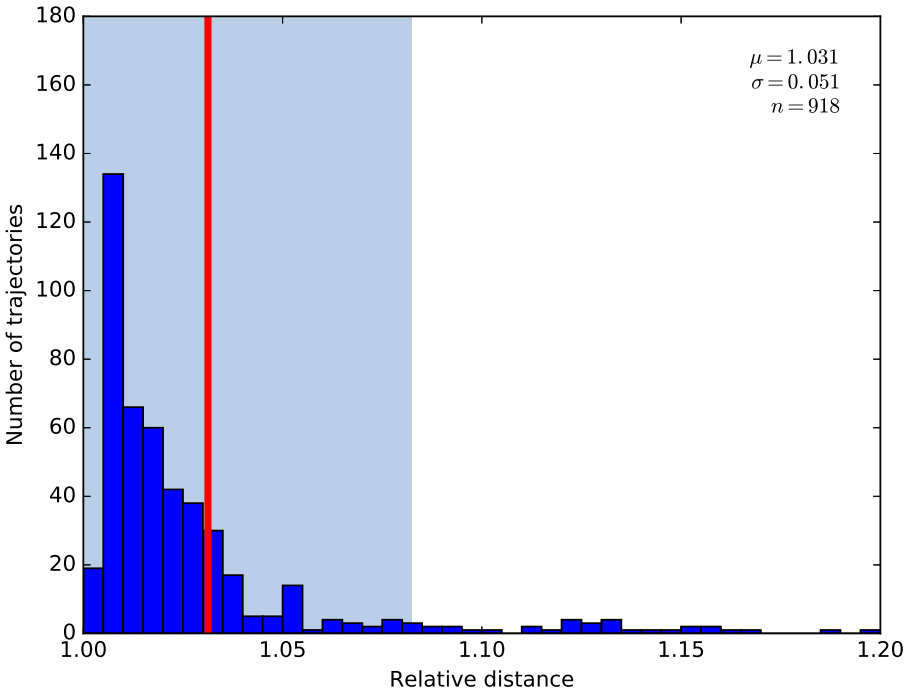


Figure 3.41: Histogram of relative distance in non-optimal paths for the Apra Harbour image using the Medial axis transform and watershed segmentation. Red line indicates the mean of the non-optimal paths. Shaded portion is +/- one standard deviation from the mean.

3.3 Conclusion

This chapter has demonstrated that the algorithms described in the previous chapter can produce simplified graphs representing the complex spatial data contained in maps and images. These graphs can then be used to make high-level path planning decisions with the intention that they could then be integrated into a mission planning system. A summary of the properties of these algorithms can be seen in Table 3.9, while Table 3.10 identifies which form of segmentation produced the highest proportion of optimal trajectories.

The complexity of the graph produced by skeletisation with the Palágyi and Kuba algorithm has been shown to be affected only slightly by scale and rotation, while maintaining homotopic segmentation for at least 95% of the segments tested. This algorithm has also been shown to produce trajectories that are homotopic and optimal up to 85% of the time. As such, the individual trajectories generated by the use of the planner may be less efficient than performing a trajectory search on the entire volume for each action, but the normalised mean length of these trajectories was found to be no more than 2% of the optimal length. This excess distance is sufficiently small that it could be considered negligible in most cases. The problem can thus be reduced to the point where it would be practical to use this as a pre-processing system for symbolic planning based executives.

Skeleton filtering can further lower the complexity of the skeleton, but increases the proportion of non-homotopic segments. This further decrease in complexity does not justify the increase in segmentation errors.

The next chapter will examine the ability of domain independent planning systems to integrate path generation with mission planning tasks.

Table 3.9: Comparison of Spatial Compression Algorithms

	Lee at al.	Palágyi and Kuba	Medial Axis Transform
Method of operation	Iterative removal of non-simple border elements	Iterative removal of border elements matching templates	Iterative removal of border elements
Sensitivity to Geometry	Some Sensitivity	Invariant	Sensitive
Sensitivity to Rotation	Some Sensitivity	Invariant when $\theta \neq 120^{\circ}$	Sensitive
Sensitivity to Scale	Some Sensitivity	Invariant when $scale > 20\%$	Sensitive

Table 3.10: Segmentation Method Resulting in Highest Proportion of Optimal Trajectories

Map type	Lee at al.	Palágyi and Kuba	Medial Axis Transform
Obstacle	Region Growing	Region Growing	Watershed
Maze	Seeded and Watershed	Watershed	Watershed
Apra Harbour	Watershed	Watershed	Watershed
Apra shoals	Watershed	Region Growing, Seeded, and Watershed	Watershed

Chapter 4

Evaluation of Spatial Planning Performance Using a Simulated Environment

4.1 Introduction

In the previous chapter, a method for performing the compression of a robot's belief space was covered. This chapter will cover the implementation of a planning system that utilises the Problem Domain Description Language (PDDL) [McDermott et al., 1998].

Domain independent planning systems are extremely flexible, but this flexibility does result in lower efficiency when compared to a domain specific planning system. This chapter will examine the suitability and scalability of a domain independent planner at performing spatial planning tasks using a synthetic task involving maritime rescue based on Project ICARUS [De Cubber et al., 2013], discussed in Section 1.4.

4.2 The Problem Domain Description Language

Before a plan can be generated, a method of describing the problem domain is required. As covered in Section 1.5.4, arguably the most influential problem description language is the Problem Domain Decription Language (PDDL). Due to the broad support of PDDL, it can provide an interface that is supported by multiple planning systems. Thus, the language can be used as a way of abstracting the planner from the system. This allows the profiling and testing of algorithms and planners to find which ones meet the requirements of the project.

The original dialects of the PDDL language could only support textual predicates and did not support numerical variables which would prevent the encoding of non-unity action costs. PDDL version 2.1 included support for numerical fluents, predicates that could represent numerical values in a planning system. In the International Planning Competition (IPC), a specific section was set aside for the optimisation of PDDL plans. Using the new PDDL requirement called ':action-costs' adds support for a specific configuration of numerical fluent designed to support plan optimisation using the ':metric' stanza [Do et al., 2009]. In this chapter the effectiveness of this form of plan optimisation will be evaluated for the task of spatial planning.

4.2.1 Experimental Evaluation of Optimisation in Symbolic Planners

Three planners were initially considered due to their current usage in ROS planning systems, including the Popf-2 planner [Coles et al., 2011] used in ROSPLAN [Cashmore et al., 2015], the Fast Downward planner [Helmert et al., 2014], and the FF planner [Hoffmann, 2000] both packaged in the jsk_3rdparty package [Okada and Ueda, 2016] [5]. The FF-X version of the FF planner was considered due to its support of PDDL 2.1, but while it did support numerical fluents, it did not support the ":action-costs" requirement. Testing was thus concentrated on only the first two planners.

Fast Downward is described by Fawcett et al. as a planning framework [Fawcett et al., 2011] rather than a singular planner. It uses a plugin based architecture for both search algorithms

[5] The authors listed in this citation are the primary listed contributors on the jsk_3rdparty github page.

and heuristics, allowing a number of algorithms to be supported with the planner. The Fast Downwards project website provides six example configurations[6] of search and heuristics that can be performed with the planner [Helmert et al., 2014]. These combinations are collated in Table 4.1. This table also includes citations where available for the heuristic types, and a short name that will be used to refer to that combination of search and heuristic in the remainder of this document. As recommended by the authors[7], the Fast Downward planner was built as a 32-bit executable, with the corresponding limits to memory size. All trials were made on an Intel Core i7-4700HQ processor clocked at 2.40GHz. Despite the single threaded nature of all the tested planners, only a single benchmark was run at a time to prevent the CPUs turbo boost feature or cache size altering the execution time. Planners were executed sequentially on the same domain and task files, so file caching may alter the results. However, unlike a competition domain, the task files are being written to disk before being executed, as such they should already be present in the file cache.

These planners all work by constructing a graph of potential future states. A *successor function* is used to select states that could be added to the graph - the *frontier*. The search algorithm will continue to add states until the goal is reached.

Since the majority of actions will be movements across a regular grid, it is expected that the *branching factor*, the number of possible successor states from each state, will be four for most states.

The next sections will detail the planners, search types and heuristics listed in Table 4.1.

4.2.1.1 A* search

The A* search algorithm [Hart et al., 1968] selects states to expand from the frontier, the set of possible successor states under consideration, based on the cost to reach the current state $g(n)$, and the value of a heuristic $h(n)$ estimating the cost to reach the goal. A* uses a metric for expanding a node equal to the cost of reaching that node plus the heuristic cost.

$$f(n) = g(n) + h(n) \tag{4.1}$$

[6] These sample configurations can be found on the Fast Downward webpage http://www.fast-downward.org/
[7] Planner usage for Fast Downward can be found on http://www.fast-downward.org/PlannerUsage

Table 4.1: Combinations of search and heuristic types used with the Fast Downward planner

Search	Heuristic	short name
Lazy Greedy	Fast Forward [Hoffmann and Nebel, 2001]	FF
Lazy Greedy	Context-Enhanced Additive Heuristic [Eyerich et al., 2012]	CEA
Lazy Greedy	Combined Fast Forward and Context-Enhanced Additive Heuristic	Dual
A*	Landmark-Cut [Pommerening et al., 2014]	LM-cut
A*	Blind	Blind
A*	Pattern Database [Haslum et al., 2007]	iPDB

The selected state is the one that satisfies the relationship;

$$\min_{n \in frontier} f(n) \tag{4.2}$$

If the heuristic is optimal, A* will grow in the direction of the goal, but a sub-optimal heuristic may result in the search expanding nodes that are not required for the solution. If the heuristic is *admissable*, it will not overestimate the cost of reaching the goal from the current state.

The use of A* may be useful for the task of planning in a spatial environment, since the number of states that can be expanded will be relatively large. Since most of these states will be spatial states representing the path of the vehicle, an optimal search will leave the majority unexpanded.

4.2.1.2 Lazy Greedy search

Lazy refers to the deferred heuristic calculation used by Fast Downward. This algorithm does not calculate the heuristics for all possible states, but only calculates $h(n)$ when a state is added to the graph. Unlike A* which selects from the frontier, the greedy search selects already expanded nodes from the graph that have the lowest heuristic cost;

$$\min_{n \in graph} h(n) \tag{4.3}$$

All successor nodes of the node are then expanded [Helmert, 2006].

The Lazy Greedy search will attempt to guide the growth of the graph towards the goal as represented by the heuristic, but since the cost of the reaching the current state is not included in the minimisation operation, the path will not be restricted to the lowest plan execution cost. The lazy component of the search is identified by Helmert as an attempt to optimise search in tasks that have a large branching factor and expensive heuristics [Helmert, 2006]. In the spatial search domain with its low branching factor, this is not likely to improve the effectiveness of the search.

4.2.1.3 Fast Forward Heuristic

Developed for the Fast Forward planner, the Fast Forward heuristic is a form of search through a *relaxed problem*, a simplification of the planning task where the preconditions of an action are ignored [Hoffmann and Nebel, 2001]. Bylander noted that the complexity of the plan satisfication problem increases with the number of preconditions, a search with no preconditions and multiple postconditions was in polynomial space, while the same with a single precondition was PSPACE-complete [Bylander, 1994]. Thus, by removing preconditions the search task should be greatly simplified.

4.2.1.4 Context-Enhanced Additive Heuristic

The Context-Enhanced Additive heuristic combines the effect of the Causal Graph heuristic and the Additive heuristic by performing a search from the goal state back to the current state [Eyerich et al., 2012]. This search produces an estimate of the number of steps that will be required to reach the goal.

4.2.1.5 Dual Heuristic

The Dual heuristic uses Fast Forward's ability to use plugins to combine the results of multiple heuristics. In this configuration, the Lazy Greedy search will alternate the use of the Fast Forward and Context-Enhanced Additive heuristics [Helmert et al., 2014].

4.2.1.6 Landmark-Cut Heuristic

The Landmark-Cut Heuristic uses a method based on a *justification graph* which is a relaxation of the domain. The heuristic is calculated by taking this graph and finding *landmarks* states that are common across all possible plans that are also *cuts* states where the removal will result in disconnections between sets [Helmert and Domshlak, 2009].

Landmark-Cut may improve the efficiency of spatial planning since in more complex domains discussed later in this chapter, there are a number of actions that need to be performed that are separated by many possible paths. These individual actions constitute landmarks that could be detected by the system.

4.2.1.7 Blind Heuristic

Blind is a simple heuristic which assigns zero to a goal state, or the cheapest action cost in non-goal states [Fawcett et al., 2011]. When combined with the A* algorithm and with unit action costs, the effect would be that the lowest cost path would always be expanded. Since this would not be guided towards the goal state, many more states would be expanded than necessary. However, the final path can be expected to be the lowest cost.

Blind with A* appears to be a solution that is low in CPU usage compared to relaxation based heuristics, and will produce efficient plans. However the domain must be a suitable one where a solution can be found before the planner fails due to memory exhaustion. With its low branching factor, the spatial domain could be suitable for this planner and heuristic combination.

4.2.1.8 Pattern Database Heuristic

The Pattern Database Heuristic uses the concept of *patterns*-small sections of the planning problem that can be isolated and solved exhaustively. These patterns are then stored in a database and accessed to create the cost estimate to reach the goal. According to Haslum et al., the difficulty in developing such an heuristic is the selection of the set of patterns to

be stored. Their implementation of the pattern database heuristic uses an algorithm that generates the pattern database from the problem instance [Haslum et al., 2007].

4.2.1.9 Popf-2

The Popf-2 planner is a partial-order chaining planner developed for the task of planning temporal environments [Coles et al., 2011]. A partial-order planner is one that finds actions that are required to be performed, but does not order them unless constrained by preconditions. Barrett and Weld observed that such an approach can have significant performance improvements in some domains [Barrett and Weld, 1994].

4.2.1.10 Summary

This set of planners, search algorithms and heuristics covers algorithms for both low and high branching problems, together with three of the four families of heuristics, delete relaxation, abstraction, and landmarks, identified by Helmet and Domshlak in their paper on the Landmark Cut heuristic [Helmert and Domshlak, 2011]. The next sections will cover a number of experiments of increasing complexity covering simple path-finding in a spatial environment, path-finding in an environment with asymmetric action costs, the scheduling of actions to minimise plan cost, and scheduling of actions under preconditions. These will allow the evaluation of the efficiency of these systems at the generation of plans, and the responsiveness of the planning systems. Measuring the duration of plan generation is of interest since in an off-line planning system, plan execution cannot begin until the plan is complete. As such, the time between the planning operation being requested and the plan being available will impact on the repsonse time of the overall planning system.

Using these planning runs, a combination of planner, search algorithm and heuristic can be selected for the task of mission planning with spatial information.

4.2.2 Pathfinding

One of the key research questions of this thesis is that if a domain-independent symbolic planner can combine mission planning with high-level path planning? The previous section has introduced seven possible combinations of search and planning system, six different configurations of the Fast Downward planner as listed in Table 4.1, and the Popf-2 planner. The remainder of this chapter will evaluate the use of these planning combinations by the use of a synthetic spatial environment. Due to space restrictions, full detail of the PDDL code used will not be included, however information on the predicates and actions used can be found in Appendix C. Similarly, for spacing some figures have been moved to Appendix D.

The base spatial environment was designed as a grid with each grid square representing a possible location for an Autonomous Surface Vessel (ASV). Such an environment does have advantages, the scale of the planning problem could be varied by altering the size of the grid. By programmatic creation of the domain and task file, a planning problem can thus be created with a variable number of nodes to be searched. By profiling the time and effectiveness of plan creation, an estimate of the performance and scalability of the planners could be developed. This can be used to inform the potential applicability of spatial planning with domain independent planners.

To encode this space as an PDDL domain, the vehicle, each possible position and the connections between the spatial locations were declared as objects of type *entity*, *node* and *edge* respectively. To store the relations between entities, nodes, and edges, a pair of predicates were declared, *at*, declaring that an entity was located at a node, and *accessible* which specified that a node was accessible from another node via a particular edge. To allow an entity to change it's node, a *move* action was declared, allowing an entity to change the node it was *at* as long as the origin and destination nodes had an *accessible* predicate, the code for which can be seen in Figure C.1. The symbolic planning example in Appendix B uses this set of actions and predicates.

To perform a trial, randomly selected start and end positions were generated, and the planner tasked with generating a sequence of actions that would take the ASV from the start to

the finish locations. Randomly placed obstacles prevented the ASV from traversing a node. Unlike buoys and marks as used in the International Regulations for Prevention of Collisions at Sea (COLREGS), there is no constraint for the vehicles path to either between a set of buoys or to a particular side or direction. These obstacles merely prevent the trivial solution of the path planning problem.

Conversion of this domain into a PDDL file suitable for planning wwas performed using a library called pddl_libs developed for this thesis. As described in Appendix C, this library has the ability to generated PDDL domain and task files from both pre-defined files or using function calls. This functionality was used to generate a PDDL task file using that encoded the nodes, their accessibility, and the initial and goal positions of the ASV. This task file is then used with each planner configuration, ensuring that all planning results were directly compatible. To explore the possible effects of different spatial configurations, 100 trials were generated for each possible condition of the experiment.

The plan execution costs, planning time, and memory usage of the planners have been visualised using notched boxplots created using the matplotlib library [Hunter, 2007]. The box of these plots extends from the lower to upper quartiles with a line at the median. As such, half of the readings will lie within the box. The whiskers extend to the range of the dataset, with outliers marked as crosses. The boxplots also have a *notch*, a narrowing of the box the extent of which is calculated using a Gaussian-based asymptotic approximation. Differences between the medians of boxplots should only be considered significant if their notches do not overlap. Matplotlib cites McGill et al. [McGill et al., 1978] and Kendall and Stuart [Kendall and Stuart, 1967] as the source of the notch creation algorithm.

Later experiments will extend this methodology with variable actions costs, multiple action types and preconditions.

The experiment was performed five hundred and three times on an $m \times m$ array - one hundred times[8] each for array sizes $m = 10, 20, 30, 40, 50$. In each test run the location of the obstacles, start position of the ASV, and the goal position were randomised, then each planner, heuristic and search combination was executed. This arrangement ensured that the

[8]Extra trials were performed on the $m = 10$ case as part of the development process. These have been left in to avoid biasing the result.

results for each planning combination were directly comparable, each being executed with an identical problem configuration.

The result of each individual planning run was checked to ensure that a valid plan was produced, and if a planner failed to produce a result, all results for that randomly generated planning problem were discarded. This prevents the case of a planner receiving an artificially lower plan cost due to being unable to plan more complex environments.

When an array size of $m = 60$ was attempted, the Fast Downward planner failed to correctly validate the plan, throwing an exception while allocating an associative map container during the checking of the object types. The failure in Fast Downward may be caused by the number of objects in the planning problem simply exceeding the ability of the map object to contain them. With $m = 60$, the pathfinding task will include 3,590 spatial nodes, connected by almost 14,000 edges - each of which is stored as an individual object within the task. If each of these objects is stored as 32-bit pointer, then the total storage space will exceed 2^{16} bytes, thus it can be suspected that the required space exceeds what the map data structure can store. This limitation would not normally be reached since problems used to test scaling of planners typically use a maximum of a few hundred nodes. As an example, the twentieth task in the transport domain from the 2014 IPC competition had just over one hundred objects [IPC benchmark authors, 2010]. The closest workload may be the Visit-All domain which also implements an $m \times m$ grid. The planners are required to generate a plan that visits every cell in the grid at least once. In the document by Lipovetzky, the largest tested task was for $m = 50$, with the LAMA [Richter and Westphal, 2010] planner only achieving $m = 30$ before errors occurred [Lipovetzky, 2010].

The planners were executed and the plan cost, plan length and planning time required to generate each plan was recorded. For this domain, the cost of executing each action was set to one, so that the plan length and plan cost were equal. To examine the way these planners generated paths, a visualisation was created showing how the plan lead from the initial to goal positions. In these images the start location is represented by the image of the ASV, the obstacles are represented by local obstacle buoys, and the goal position is represented as a safe water mark as seen in Figure 4.1. A visualisation of planning with the Greedy and A* planners can be seen in Figures 4.2 and 4.3 respectively. Examination

of these sample images showed that the three combinations of Fast Downward that used Greedy search, and the three that used A* search produced identical plans. This similarity shows that in the examined case, the effect of heuristic choice on the generated plan is minimal.

The execution times, costs of the generated plans, and the memory used during search were recorded and averages for time and cost tabulated in Tables 4.2 and 4.3. These results were then graphed in Figures 4.4, 4.5 and 4.6 as notched boxplots.

The planners were all capable of operating in the case of $m = 50$, but exhibited increases in planning times as the total number of nodes in the spatial environment increased. The Popf-2 planner exhibited shorter planning times than the Fast Downward based planning runs. This is related to the structure of the planners - Popf-2 is a single executable that reads the domain and task files, solves the problem, and outputs the result. The Fast Downward planning system uses separate executables in both C++ and Python to read the inputs, translate to its own internal description language, plan the result and validate the plan. Examination of the output of Fast Downward shows that the majority of the execution time was spent in the *instantiation* stage, where the templated actions and grounded predicates are converted into a set of grounded actions. The actual search times reported by Fast Downward were only a small component of the overall time.

Memory usage of the search component of Fast Downward was also recorded. Examination of Figure 4.6 shows that the majority of the Fast Downward systems used similar memory amounts. The exceptions being the Lazy Greedy search with the CEA and Dual heuristics

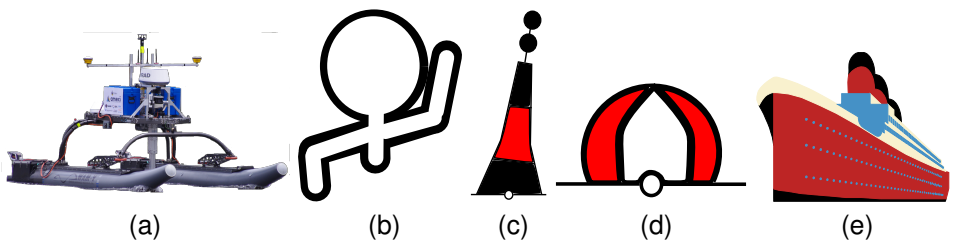


Figure 4.1: Markers used to visualise search and rescue plans (a) TopCat Autonomous Surface Vessel (ASV) (b) Outline representing subject requiring rescue (c) Isolated danger mark representing a point obstacle (d) Safe water mark representing a goal location (e) Ship representing a supply of stored lifeboats. Buoy and ship symbols were downloaded from www.openclipart.org

when $n \geq 20$. Since Dual is a combination of the Fast Forward and CEA heuristics, it can be inferred that CEA has a significantly larger memory usage in this domain than the other tested heuristics.

As mentioned in Section 4.2.1, caching effects could bias the results of the planners. Examination of the results shows that the first executed planner (Fast Forward with Lazy Greedy search and FF heuristic) did not produce times that differed significantly from the other Lazy Greedy searches despite the variation in conditions. As such, it can be expected that any cache effects are small compared to the search task.

Table 4.2: Average times for path generation through $m \times m$ array. All times are in seconds. Total number of trials performed, $n = 3521$.

Planner	Width of environment m (elements)				
	10	20	30	40	50
Number of trials	103	100	100	100	100
Fast Downward, FF	0.13	0.36	0.79	1.41	2.3
Fast Downward, CEA	0.13	0.36	0.79	1.43	2.32
Fast Downward, Dual	0.13	0.36	0.78	1.43	2.33
Fast Downward, LM-cut	0.13	0.36	0.82	1.56	2.65
Fast Downward, Blind	0.13	0.36	0.78	1.42	2.3
Fast Downward, iPDB	0.13	0.36	0.79	1.43	2.32
Popf-2	0.01	0.04	0.12	0.32	0.65

Table 4.3: Average costs for pathfinding through $m \times m$ array. Costs have been scaled by a factor of 8 so that they are comparable to those in Table 4.5. Total number of trials performed, $n = 3521$.

Planner	Width of environment m (elements)				
	10	20	30	40	50
Number of trials	103	100	100	100	100
Fast Downward, FF	57.48	129.51	174.91	254.71	319.36
Fast Downward, CEA	57.48	129.51	174.91	254.71	319.36
Fast Downward, Dual	57.48	129.51	174.91	254.71	319.36
Fast Downward, LM-cut	55.18	125.93	172.0	258.21	318.34
Fast Downward, Blind	57.03	125.73	172.0	258.21	318.34
Fast Downward, iPDB	57.04	125.73	172.0	258.21	318.34
Popf-2	57.09	120.56	172.67	258.88	321.66

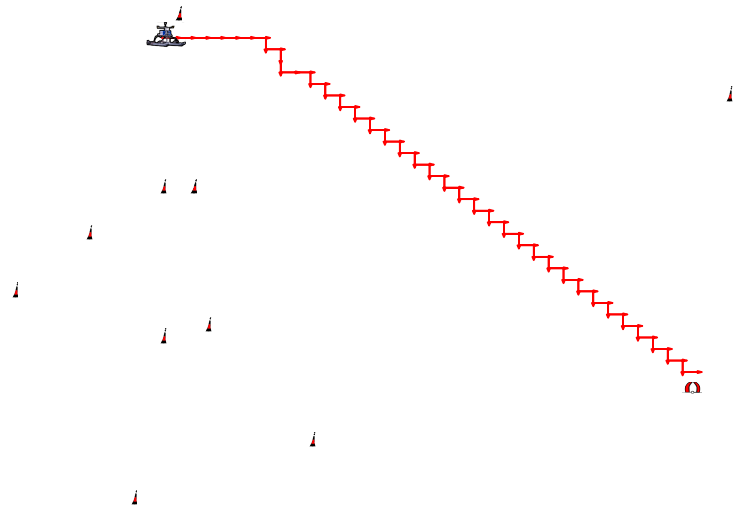


Figure 4.2: Sample motion plan generated by Fast Downward with Lazy Greedy search and the Fast Forward (FF) heuristic. The Context-Enhanced Addidtive (CEA) and Dual heuristics produced identical plans. For completeness, these results can be seen in Appendix D with the result of using the CEA heuristic seen in Figure D.1, and Dual seen in Figure D.2

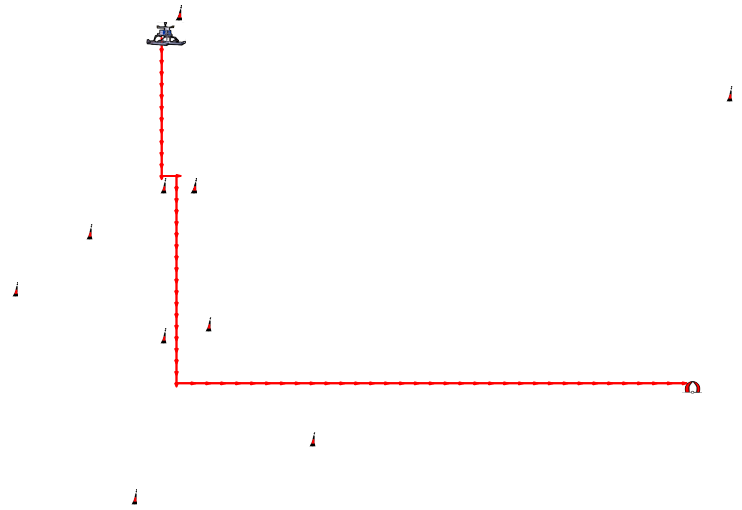


Figure 4.3: Sample motion plan generated by Fast Downward and A* search with the Landmark-Cut (LM-cut) heuristic. The Blind and Pattern Database (iPDB) heuristics and the Popf-2 planner produced identical results. For completeness, these results can be seen in Appendix D with the result of using the Blind heuristic seen in Figure D.3, iPDB seen in Figure D.4, and Popf-2 seen in Figure D.5

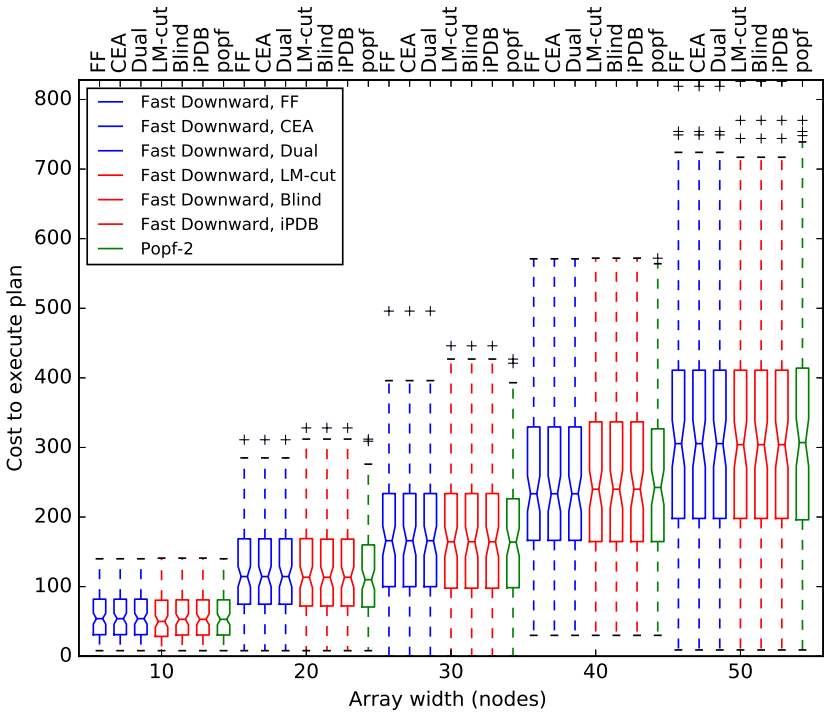


Figure 4.4: Execution cost for generated plans for $m \times m$ nodes. Costs have been scaled by a factor of 8 so that they are comparable to those in Table 4.5. Total number of trials performed, $n = 3521$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

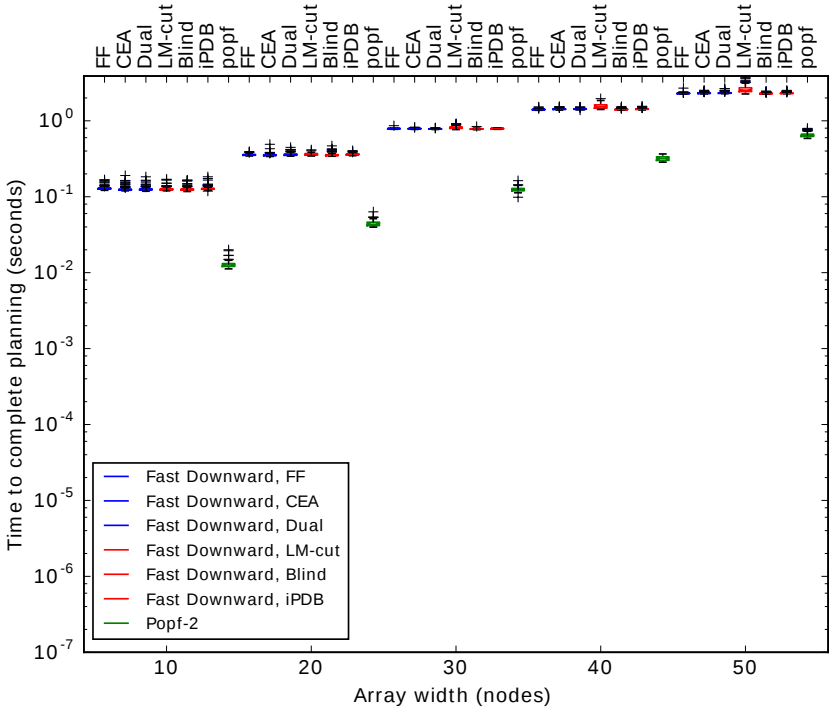


Figure 4.5: Time to complete a planning run for the motion planning domain for $m \times m$ nodes. All times are in seconds. Total number of trials performed, $n = 3521$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

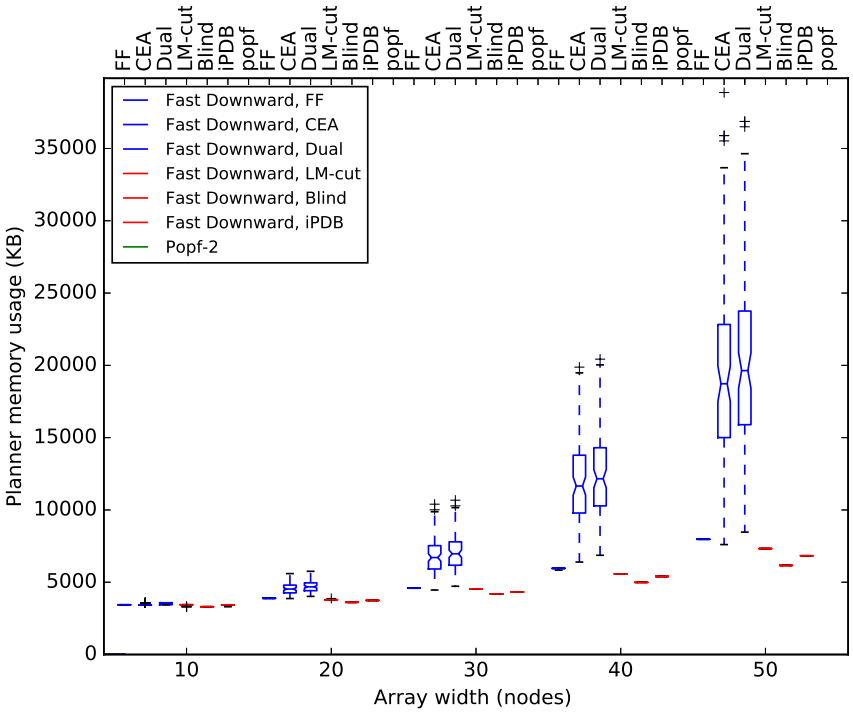


Figure 4.6: Memory used in search for the motion planning domain and $m \times m$ nodes. Total number of trials performed, $n = 3018$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

4.2.3 Pathfinding with Asymmetric Action Costs

In the previous section, routes were planned with equal costs in all directions. However, the effect of wind and water action may further result in motion costs that are asymmetric - movement in the direction of currents will be lower in cost than movements that oppose the currents. This experiment will examine the effect of adding variable action costs to the test domain described in Section 4.2.2.

While local currents may exhibit random or variable nature, Carton wrote that at a sufficiently large scale, ocean currents can be modelled as sets of vortices [Carton, 2001]. Zeng et al. used this conclusion to optimise Autonomous Underwater Vehicle (AUV) trajectories using genetic algorithms and particle swarm optimisation [Zeng et al., 2012].

The Lamb vortex is a solution to the Navier-Stokes equations in cylindrical co-ordinates where the tangential velocity of the fluid can be specified by Equation 4.4 [Meunier and

Villermaux, 2003].

$$v_{tangential} = \frac{\Gamma}{2\pi r}\left(1 - e^{-\frac{r^2}{a^2}}\right) \tag{4.4}$$

where;

$v_{tangential}$	tangential velocity
Γ	vortex circulation
r	radius
a	core radius

(4.5)

This model of vortex velocity was used to create a path planning system with asymmetric action costs. Equation 4.4 was used to create a pair of equations with the velocity components shown in Equation 4.6.

$$\begin{aligned} v_x &= \frac{v_{tangential} \times y}{r} \\ v_y &= \frac{-v_{tangential} \times x}{r} \end{aligned} \tag{4.6}$$

For each domain, a pair of vortices were added with random centres and vorticity values in the range of $[-1, 1]$. For each node in the domain, the velocity components in the x and y directions for each vortex were calculated as shown in Equation 4.6. The sum of the vortices' velocity components was recorded as the water velocity for that node.

When constructing the PDDL code describing the task, the action cost for moving between nodes was set as shown in Equation 4.7.

$$\begin{aligned} cost_{north} &= \max\left(integer\left(10 + 8 \times v_y\right), 1\right) \\ cost_{east} &= \max\left(integer\left(10 + 8 \times v_x\right), 1\right) \\ cost_{south} &= \max\left(integer\left(10 - 8 \times v_y\right), 1\right) \\ cost_{west} &= \max\left(integer\left(10 - 8 \times v_x\right), 1\right) \end{aligned} \tag{4.7}$$

With $\Gamma = a = 1$, the peak tangential velocity of a vortex is 0.63. The scaling value of 8 allows a range of possible costs in the range $[1, 20]$. This allows a significant range of possible action costs while ensuring that the plan costs are monotonically increasing.

Movement costs were encoded in the PDDL task file as numerical predicates of type *en-*

ergyrequired, with the move action updated to increase the value of the *total_cost* value by the energy cost of the traversed edge as encoded by this predicate. The planner was then directed to minimise the total_cost value using the :metric stanza.

The testing process was similar with the start position, end position and obstacle locations being randomised as described for the path-finding test as outlined in Section 4.2.2. In addition to the factors that were randomised for the pathfinding task, the location and strength of the vortices were also randomised. This problem was then executed for all planners with a variety of different sizes for m, and the results recorded. Sample images showing the path of the planned vehicle can be seen in Figures 4.7 and 4.8. This uses the same iconography for representing the vehicle and goal, but with the addition of streamlines showing the trajectory of the water flow. The colour of the streamlines shows the velocity with colours towards the red end of the spectrum representing high velocities, and the blue end representing lower velocities.

The average plan costs for all Fast Downward plans were within ± 1 of each other, while the Popf-2 planner generated slightly higher cost plans. From the boxplot in Figure 4.9, this effect does not appear to be significant. As with the previous environment, the Fast Downward planners all took similar lengths of time to generate plans, while the Popf-2 planner had a much shorter average planning time.

As with the equal cost metric example, Figure 4.11 shows that memory usage for search that used the CEA heuristic was higher than the remaining cases when $n \geq 20$. Comparison between Tables 4.3 and 4.5 showed that all planners exhibited reduced average plan execution costs in the asymmetric cost domain compared to plans generated in the previous equal-movement cost domain for $m = [40, 50]$. This result shows that a domain independent planning system can utilise enviropnmental effects to reduce overall plan cost.

4.2.4 Ordering Actions

In the previous experiments, the planners produced lists of actions that moved the vehicle through environments with symmetric or asymmetric costs - in effect producing high-level path plans for the vehicle. This could be performed by a more specialised path planning

Table 4.4: Average planning times for pathfinding through $m \times m$ array with asymmetric costs. All times are in seconds. Total number of trials performed, $n = 3479$.

Planner	Width of environment m (elements)				
	10	20	30	40	50
Number of trials	99	99	99	100	100
Fast Downward, FF	0.18	1.23	6.33	21.54	59.49
Fast Downward, CEA	0.17	1.24	6.35	21.57	59.72
Fast Downward, Dual	0.18	1.27	6.39	21.65	59.71
Fast Downward, LM-cut	0.17	1.29	6.58	22.09	61.69
Fast Downward, Blind	0.17	1.25	6.35	21.4	59.85
Fast Downward, iPDB	0.17	1.25	6.41	21.39	59.92
Popf-2	0.02	0.14	0.65	1.94	4.7

Table 4.5: Average plan costs for pathfinding through $m \times m$ array with asymmetric costs. Total number of trials performed, $n = 3479$.

Planner	Width of environment m (elements)				
	10	20	30	40	50
Number of runs	99	99	99	100	100
Fast Downward, FF	59.12	120.51	183.42	200.01	293.0
Fast Downward, CEA	59.12	120.51	183.42	200.01	293.0
Fast Downward, Dual	59.12	120.51	183.42	200.01	293.0
Fast Downward, LM-cut	59.24	120.86	183.78	200.2	293.16
Fast Downward, Blind	58.84	120.27	183.34	199.89	292.78
Fast Downward, iPDB	59.24	120.86	183.78	200.2	293.16
Popf-2	62.47	128.57	190.2	203.55	302.58

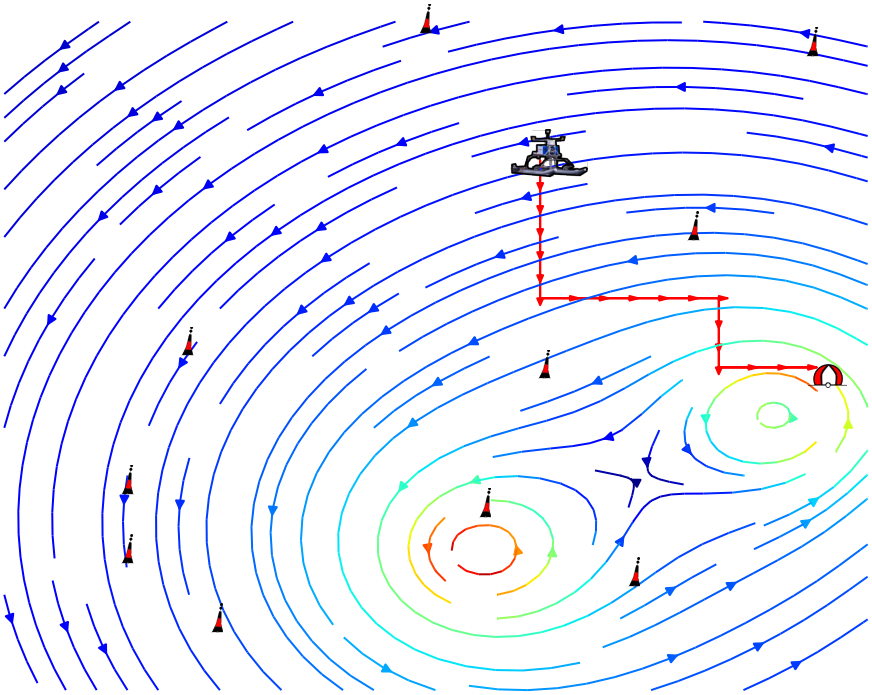
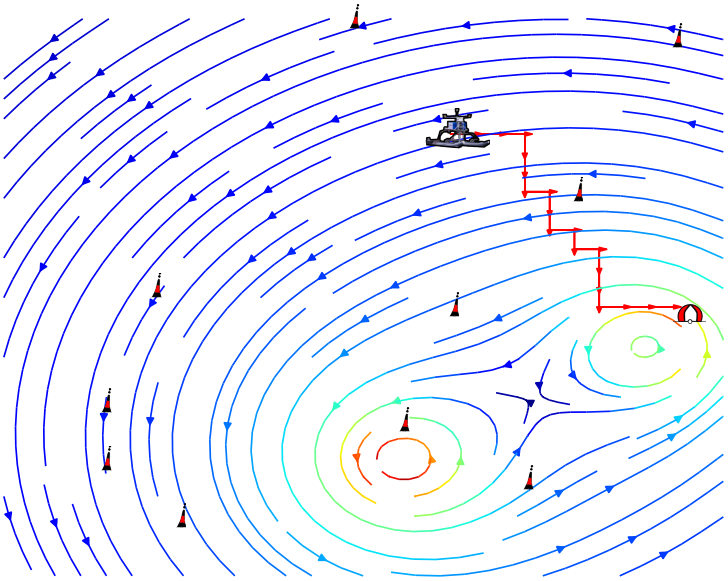
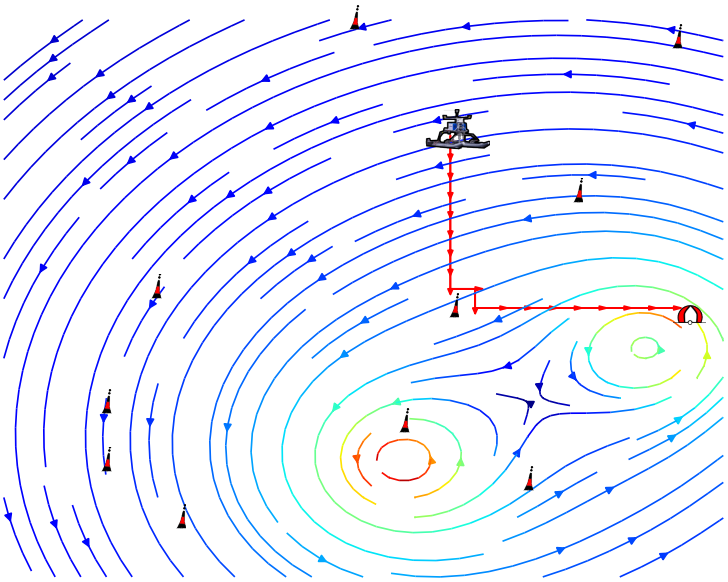


Figure 4.7: Sample motion plans generated by Fast Downward with Lazy Greedy search, and the Fast Forward (FF) heuristic. The Context Enhanced-Additive (CEA) and Dual heuristics produced identical plans.



(a)



(b)

Figure 4.8: Sample motion plans generated through an asymetric cost environment. Streamlines represent the direction and strength of the water current. (a) Fast Downward with Landmark-Cut (LM-cut) (b) Popf-2 planner. Fast Downward with A* and the Blind and Pattern Database (iPDB) heuristics were similar to LM-cut.

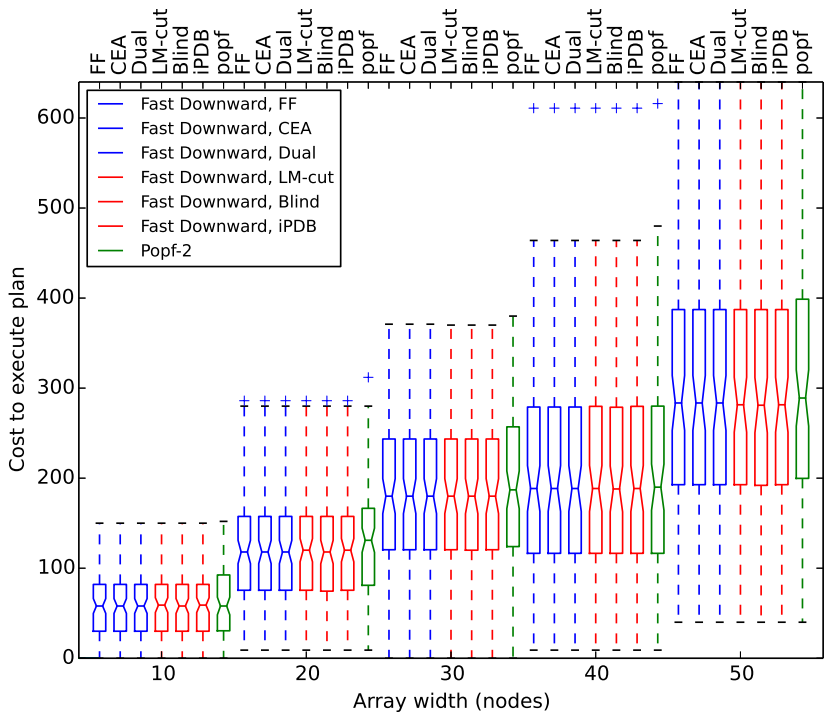


Figure 4.9: Boxplot of execution cost of generated plans for problem containing $m \times m$ nodes. Total number of trials performed, $n = 3479$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

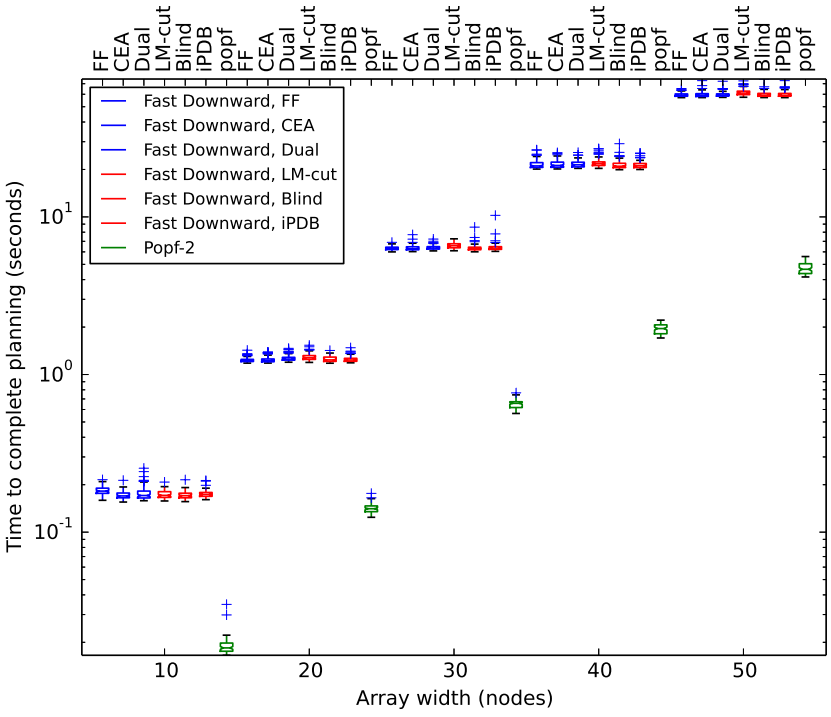


Figure 4.10: Boxplot of time required to generate plans for problem containing $m \times m$ nodes. All times are in seconds. Total number of trials performed, $n = 3479$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

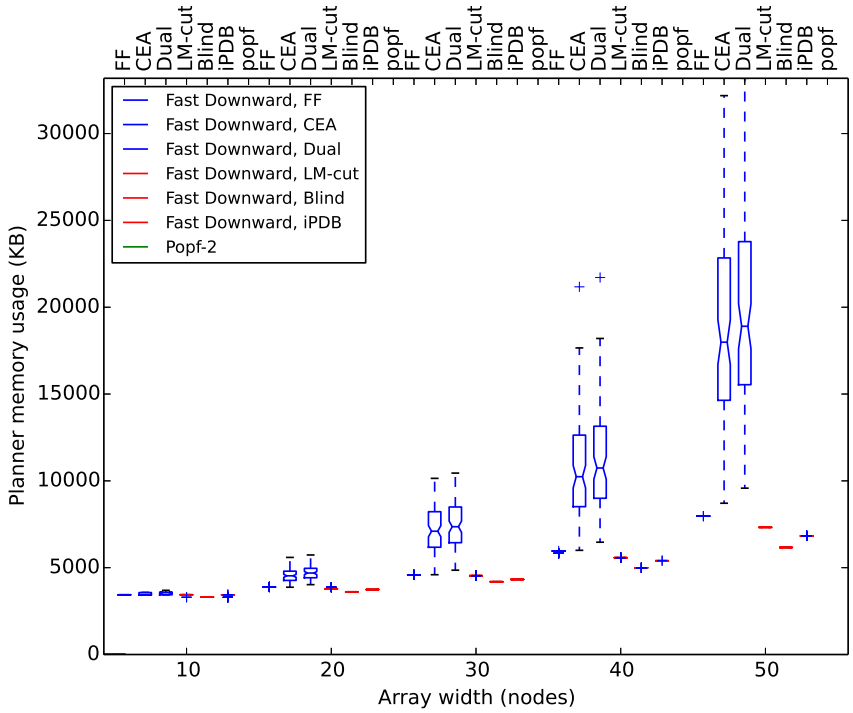


Figure 4.11: Memory used in search for the motion planning domain and m nodes. Total number of trials performed, $n = 3000$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

algorithm such as that developed by Zeng et al. [Zeng et al., 2012]. A scheduling system, however may require the ordering of a set of actions to minimise the time for them to be executed.

The advantage of a domain independent planner is that it can generate more complex plans consisting of multiple action types that satisfy prerequisites. To test the applicability of generating such plans in a spatial environment, a correspondingly more complex task is required.

To provide an illustrative scenario for this problem, the ICARUS project discussed in Section 1.4 was chosen. In such a scenario, a number of survivors in danger of drowning have been located by an Unmanned Aerial Vehicle (UAV). The ASV is required to visit each subject at least cost. Since the position of each survivor is known, the surface vehicle can optimise the scheduling of tasks to minimise the time until rescue of the survivors. For this to occur, the planning system must generate efficient plans in a timely manner.

The planning environment described in Sections 4.2.2 and 4.2.3 was extended by adding randomly located survivors requiring rescue. The vehicle was required to visit each subject in an order that minimised the overall plan cost. As such, this domain includes not only the

planning of spatial actions, but also the scheduling of tasks. By combining the task satisfication with spatial information the search can expand only those solution states necessary to reach the goal.

This task was implemented in PDDL by adding the survivors as entities, and a new action *get* which allows an entity to collect another. To distinguish between active entities and those requiring rescue, another predicate *canact* was added. The presence of this predicate was a precondition to the move and get actions. The goal of the wamv being at a location was replaced with the *has* predicate being true for all survivors. The PDDL code for the *get* action can be found in Appendix C in Figure C.2.

With the size of the grid set at $m = 30$, the largest domain that produced a plan within a responsive time, the planners were run one hundred times each for between one and five survivors, and the results were recorded and tabulated. Sample motion plans with five survivors can be seen in Figures 4.12 and 4.13. The average planning times and plan execution costs are tabulated in Tables 4.6 and 4.7, and plotted in Figures 4.14 and 4.15. The most efficient plans were generated by Fast Downward with the A* based planning runs (LM-cut, Blind, iPDB). These were dramatically shorter than the Lazy Greedy based planning runs (FF, CEA, Dual), with the Popf-2 planner falling in-between.

As with the previous domain, the Fast Downward planning system gave consistent times for planning with the exception of the A* search with the LM-cut heuristic. The median time for LM-cut was several times larger than the other conditions. The Popf-2 planner which had been consistently faster than Fast Downward produced outliers that exceeded the median planning time for the non-LM-cut Fast Downward conditions.

Examination of Figure 4.12 showed that the ordering of actions using the Fast Forward heuristic appeared to follow the direction of current flow. The Context-Enhanced Additive heuristic, while building paths that appeared to follow the water currents, scheduled actions that appeared to require backtracking with a correspondingly higher plan cost. Figure 4.13 showed that the A* based search demonstrated a similar strategy to the Lazy Greedy with Fast Forward heuristic, differing primarily in the handling of the last two actions. The Lazy Greedy search followed a low cost trajectory to both of the last two survivors before heading

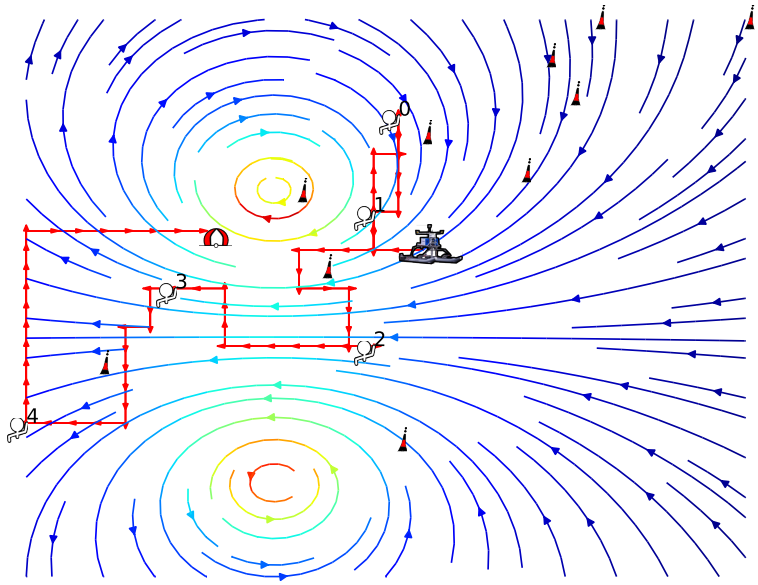
to the safe water mark, while the A* based systems only followed a low cost trajectory to the furthest survivor before turning against the current to collect the final survivor and reach the safe water mark. This strategy required higher cost movements, but allowed an overall lower cost solution by shortening the plan length.

Table 4.6: Average plan generation times for ordering actions in an array with asymmetric costs. All times are in seconds. Total number of trials performed, $n = 3479$.

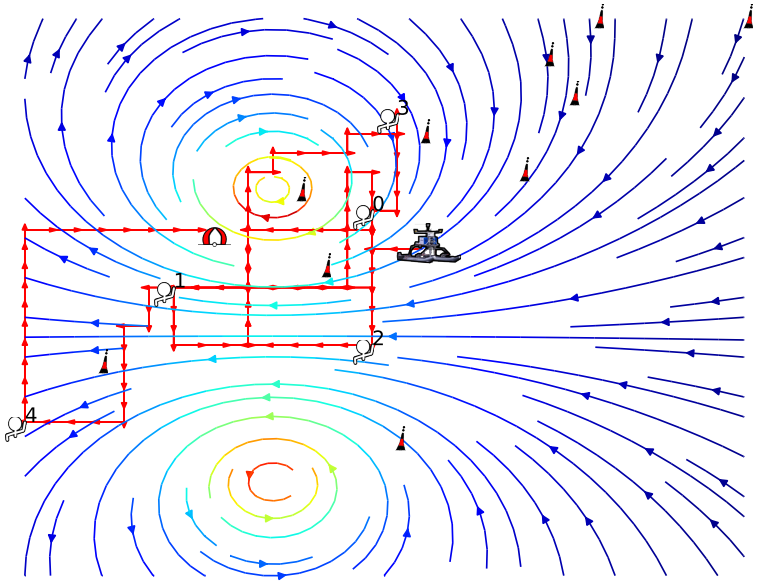
Planner	Number of survivors				
	1	2	3	4	5
Number of trials	100	100	100	99	98
Fast Downward, FF	6.48	6.48	6.52	6.35	6.74
Fast Downward, CEA	6.47	6.52	6.58	6.42	6.71
Fast Downward, Dual	6.56	6.57	6.6	6.46	6.86
Fast Downward, LM-cut	9.67	15.55	24.05	39.69	63.05
Fast Downward, Blind	6.52	6.51	6.51	6.37	6.73
Fast Downward, iPDB	6.56	6.6	6.52	6.3	6.71
Popf-2	0.98	1.12	1.52	1.89	2.74

Table 4.7: Average plan execution costs for ordering actions in an array with asymmetric costs. Total number of trials performed, $n = 3479$.

Planner	Number of survivors				
	1	2	3	4	5
Number of trials	100	100	100	99	98
Fast Downward, FF	392.74	625.13	890.3	1112.2	1325.69
Fast Downward, CEA	388.8	634.74	931.66	1166.8	1439.69
Fast Downward, Dual	392.54	623.39	900.68	1084.84	1380.36
Fast Downward, LM-cut	365.67	493.68	601.5	658.48	724.97
Fast Downward, Blind	365.32	493.42	601.08	658.18	724.71
Fast Downward, iPDB	365.67	493.68	601.5	658.48	724.97
Popf-2	379.99	583.54	831.76	936.38	1137.3

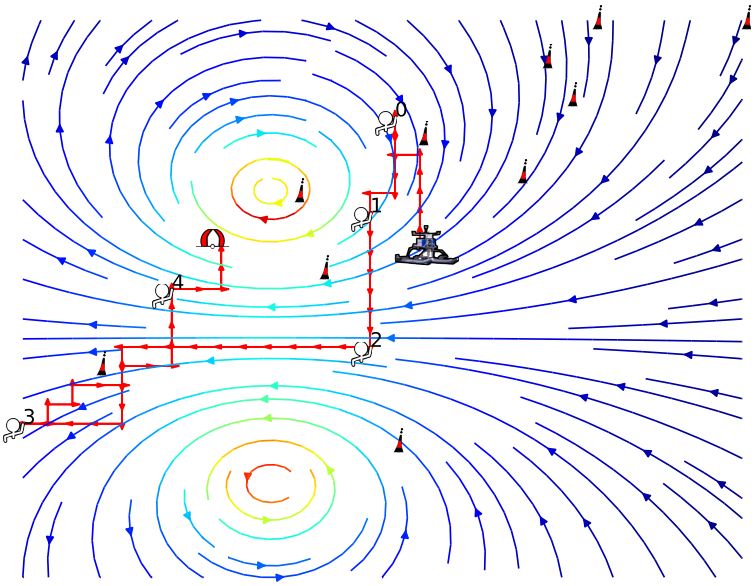


(a)

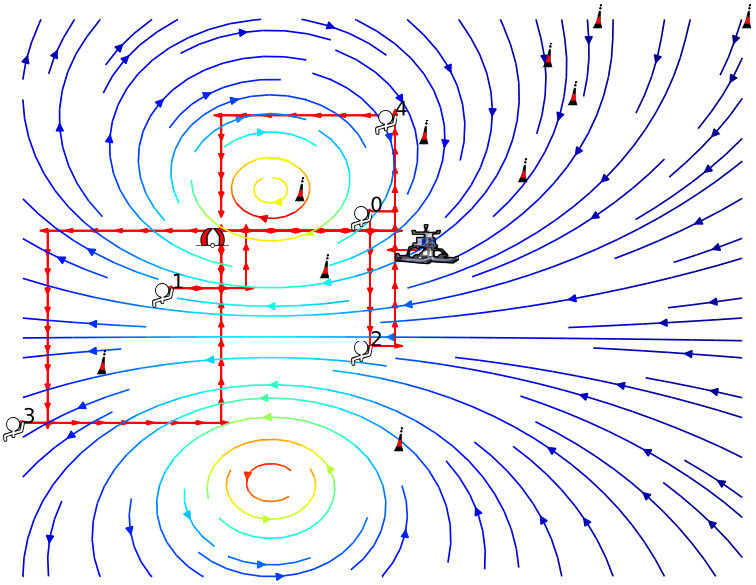


(b)

Figure 4.12: Sample mission plans to rescue five survivors as generated by Fast Downward with Lazy Greedy search through an asymetric cost environment. Numbers indicate the order in which survivors were rescued. (a) Fast Downward with Fast Forward (FF) (b) Fast Downward, Context Enhanced-Additive (CEA). The plot for the Dual heuristic can be seen in Appendix D, in Figure D.6.



(a)



(b)

Figure 4.13: Sample mission plans to rescue five survivors as generated by Fast Downward with A* search, and the Popf-2 planner through an asymmetric cost environment. Numbers indicate the order in which survivors were rescued (a) Fast Downward with Landmark-Cut (LM-cut) (b) Popf-2 planner. Fast Downward with Blind and Fast Downward with Pattern Database (iPDB) were similar to Fast Downward with LM-cut and can be seen in Appendix D, in Figure D.7 (a) for the Blind heuristic and Figure D.7 (b) for the iPDB heuristic.

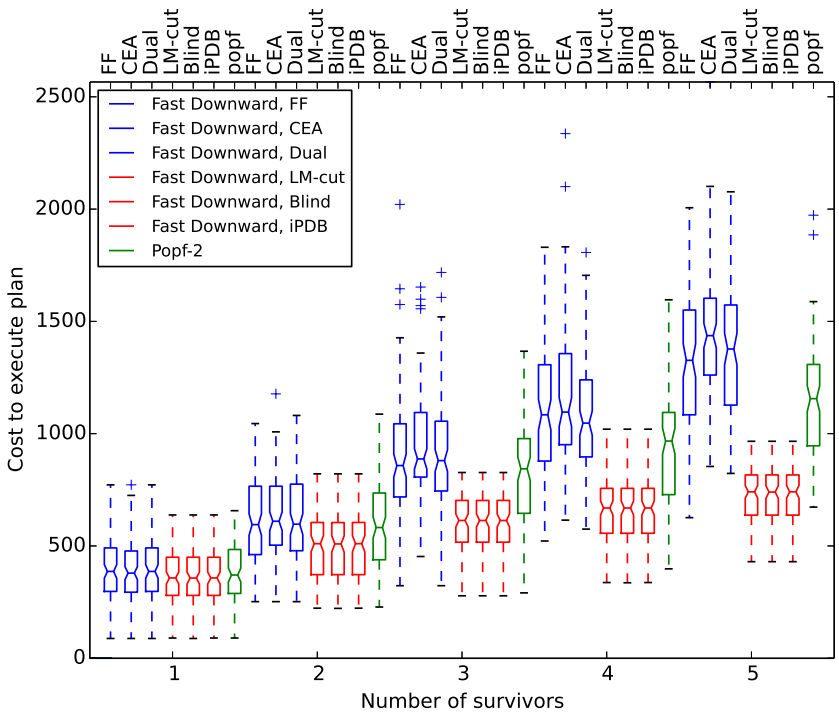


Figure 4.14: Execution cost for generated plans for m survivors to rescue. Total number of trials performed, $n = 3479$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

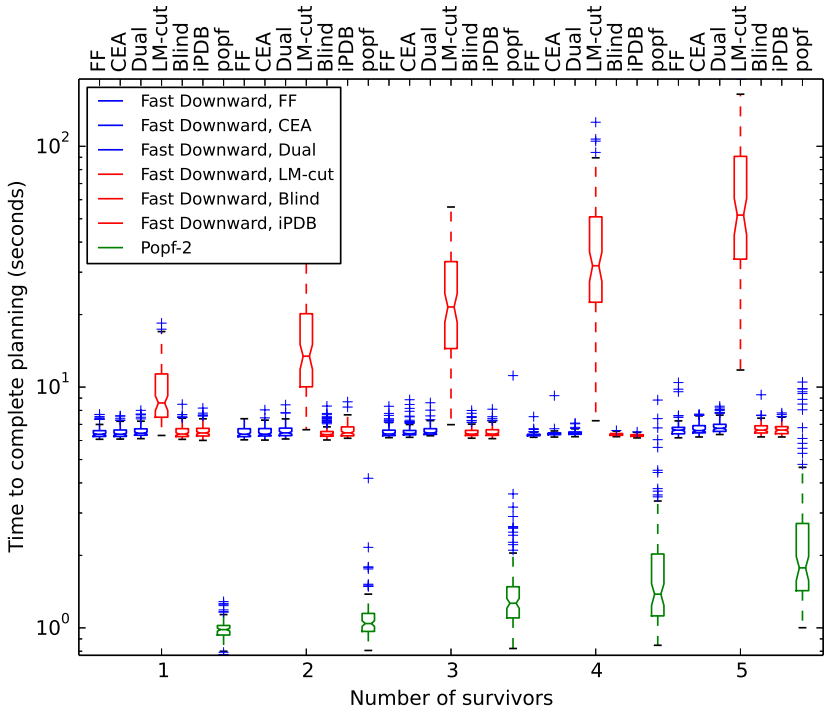


Figure 4.15: Time to complete a planning run for the motion planning domain for m survivors to rescue. Total number of trials performed, $n = 3479$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

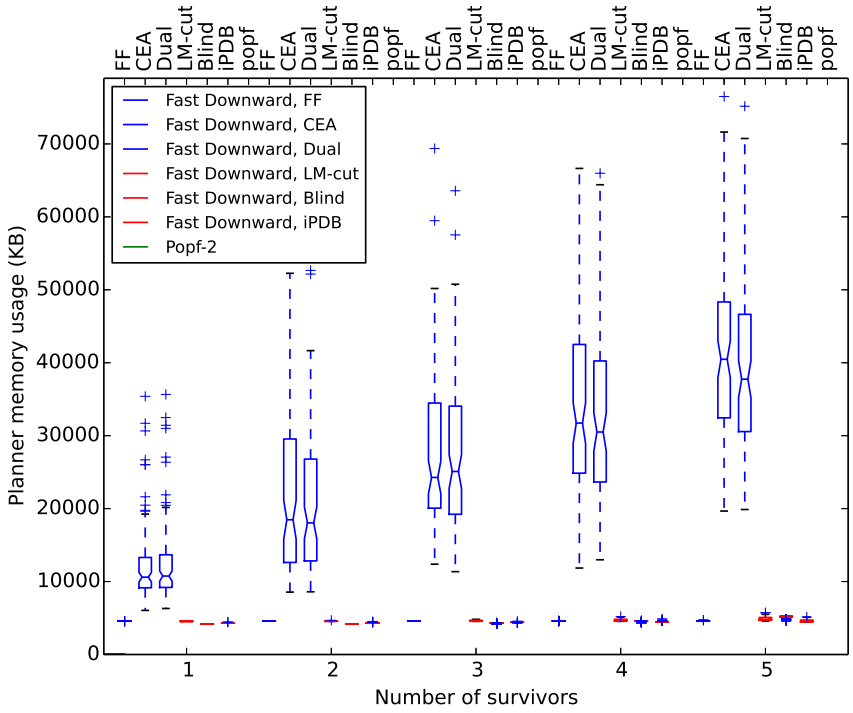


Figure 4.16: Memory used in search for m survivors to rescue. Total number of trials performed, $n = 3479$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

4.2.5 Ordering Multiple Actions

In the previous experiment, each subject was visited by the vehicle. This demonstrated scheduling and pathfinding, but not the satisfaction of prerequisites. In a search and rescue scenario, it is likely that the payload of the rescue platform would be limited. The ICARUS project demonstrated the deployment of a lifeboat carrying ASV from a surface platform, but it is possible that such a vehicle could be limited to carrying a single lifeboat, resulting in a requirement for replenishment between rescues [Machado et al., 2014]. This limitation could require several visits to a supply ship to replenish supplies between runs.

To test the ability of the planner to schedule actions, the domain was extended to replace the earlier *get* action with *collect* and *deploy* actions representing the handling of lifeboats. In this modified domain, a collect action requires the vehicle to be in the same node as the supply ship, and results in the ASV carrying a lifeboat. The deploy action requires the vehicle to be in the same node as a subject with a lifeboat that can be deployed. The PDDL code for these actions can be found in Appendix C in Figures C.3 and C.4.

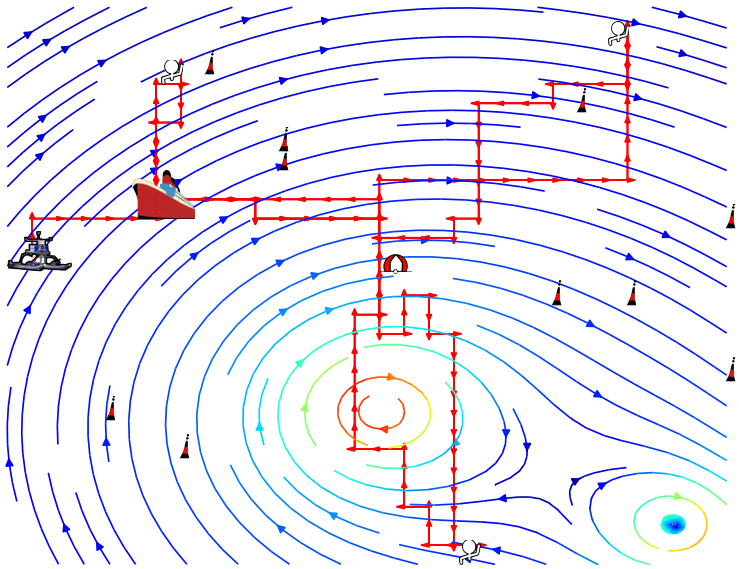
To investigate scaling with the number of actions, the size of the environment was fixed

at $n = 30$ and planning runs were executed with a variable number of supply ships and survivors to be rescued. An example of such a set of planning runs can be seen in Figures 4.17 and 4.18. The times and costs for the runs were tabulated and the results can be seen in Tables 4.8 and 4.9 for one supply ship, and Tables 4.10 and 4.11 for two supply ships. This information is also graphed in Figures 4.19 and 4.20 for one supply ship, and Figures 4.22 and 4.23 for two supply ships.

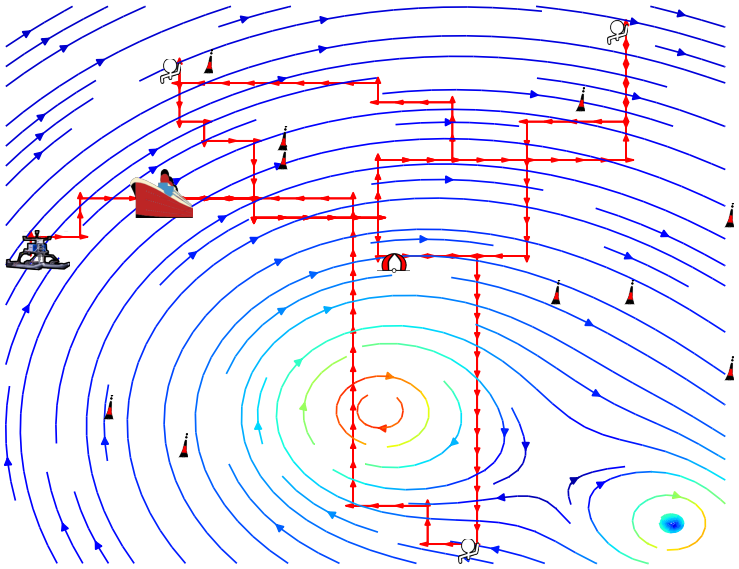
The Popf-2 planner produced similar execution cost plans to the Lazy Greedy search, but unlike previous testing domains demonstrated longer planning times than Fast Downward. From the trend when $survivors \leq 3$ it is possible it would have produced plans in less time than A* with LM-cut, but due to the exclusion of this heuristic from the more complex case, it cannot be confirmed.

As with the previous domain, the Lazy Greedy searches that used the CEA heuristic demonstrated much greater memory usage than the FF heuristic as seen in Figures 4.21 and 4.24. Unlike the previous domain, CEA demonstrated significantly shorter plans than FF when $survivors = 5$. Since Lazy Greedy search does not consider the current plan cost when evaluating the next state to expand, the CEA heuristic must select future states that lead to lower overall cost paths than FF.

In this experiment, A* search based planning continued to outperform Lazy Greedy search and the Popf-2 planner, however the duration required for generating plans with the LM-cut heuristic were large enough that its runs were limited to $survivors = [1, 3]$. The iPDB and Blind heuristics performed well, producing the lowest cost plans while still maintaining minimal memory and time footprints. Notably the mean times for all numbers of survivors with the latter heuristics showed small variation. This shows that there is potential for further increases in mission complexity with the combination of A* search and these heuristics.

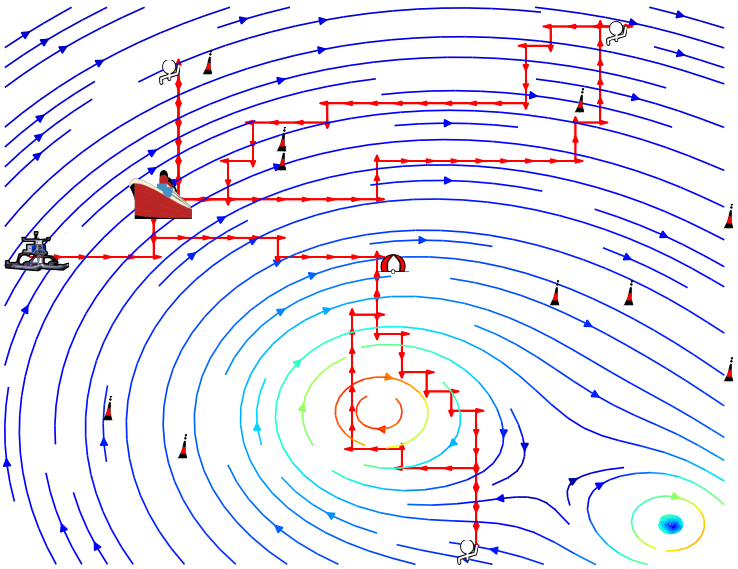


(a)

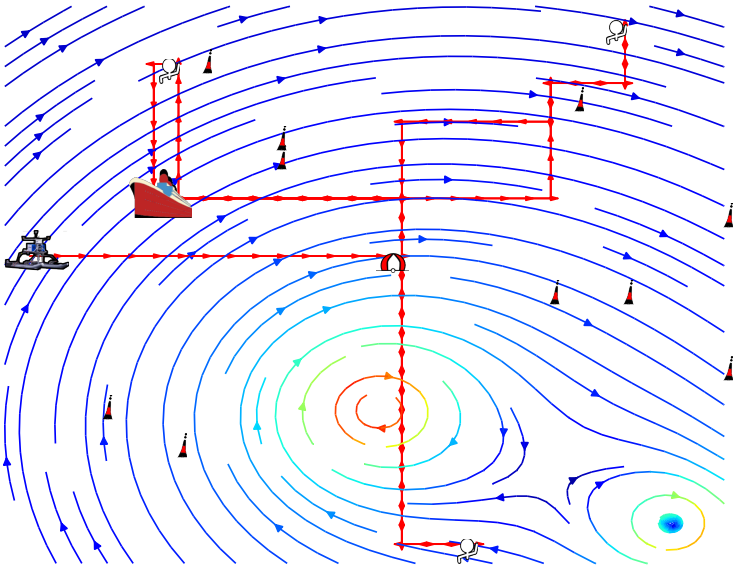


(b)

Figure 4.17: Sample motion plans for domains including move, collect, and deploy actions using Fast Downward and Lazy Greedy Search, and the Popf-2 planner. These runs included three survivors for rescue and one supply ship. Arrows on the red vehicle track indicate the direction of motion. (a) Fast Downward with Fast Forward (FF) (b) Fast Downward with Context-Enhanced Additive (CEA). The plot for the Dual heuristic can be seen in Appendix D in Figure D.8.



(a)



(b)

Figure 4.18: Sample motion plans for domains including move, collect, and deploy actions using Fast Downward, A* search and the Popf-2 Planner. These runs included three survivors for rescue and one supply ship. Arrows on the red vehicle track indicate the direction of motion. (a) Fast Downward with Landmark-Cut (LM-cut) (b) Popf-2 planner. The plot for the Blind and Pattern Database (iPDB) heuristics are similar to LM-cut and can be seen in Appendix D, with Blind in Figure D.9 (a) iPDB in Figure D.9 (b).

Table 4.8: Average times for planning with move, collect, and deploy actions. A single supply ship is present. Total number of trials performed, $n = 3253$

	Number of survivors requiring rescue				
	1	2	3	4	5
Number of trials	99	98	98	97	100
Fast Downward, FF	6.65	6.63	6.59	6.73	6.64
Fast Downward, CEA	6.64	6.72	6.72	6.77	6.78
Fast Downward, Dual	6.64	6.74	6.82	6.93	7.04
Fast Downward, LM-cut	18.68	52.34	150.15		
Fast Downward, Blind	6.67	6.6	6.71	6.79	6.86
Fast Downward, iPDB	6.7	6.67	6.72	6.84	6.92
Popf-2	1.27	3.74	10.66	25.25	53.91

Table 4.9: Average costs for plan execution with move, collect, and deploy actions. A single supply ship is present. Total number of trials performed, $n = 3253$

	Number of survivors requiring rescue				
	1	2	3	4	5
Number of trials	99	98	98	97	100
Fast Downward, FF	617.12	1087.28	1594.8	2083.4	2711.87
Fast Downward, CEA	610.67	1054.74	1478.31	1941.1	2467.88
Fast Downward, Dual	613.23	1059.69	1500.03	1938.58	2495.48
Fast Downward, LM-cut	533.69	850.97	1192.66		
Fast Downward, Blind	533.29	850.47	1192.34	1484.23	1938.53
Fast Downward, iPDB	533.69	850.97	1192.66	1484.62	1938.92
Popf-2	618.68	1027.28	1466.4	1868.45	2383.28

Table 4.10: Average times for planning with move, collect, and deploy actions. Two supply ships are present. Total number of trials performed, $n = 3273$.

	Number of survivors requiring rescue				
	1	2	3	4	5
Number of trials	100	98	99	99	100
Fast Downward, FF	6.53	6.41	6.35	6.51	6.68
Fast Downward, CEA	6.54	6.47	6.49	6.59	6.89
Fast Downward, Dual	6.52	6.43	6.5	6.73	6.95
Fast Downward, LM-cut	17.03	46.02	118.78		
Fast Downward, Blind	6.51	6.43	6.4	6.56	6.83
Fast Downward, iPDB	6.57	6.43	6.45	6.68	6.96
Popf-2	1.1	2.39	5.64	13.26	27.58

Table 4.11: Average costs for plan execution with move, collect, and deploy actions. Two supply ships are present. Total number of trials performed, $n = 3273$.

	Number of survivors requiring rescue				
	1	2	3	4	5
Number of trials	100	98	99	99	100
Fast Downward, FF	553.19	1015.0	1360.02	1887.38	2397.35
Fast Downward, CEA	559.43	966.0	1328.06	1685.04	2174.22
Fast Downward, Dual	561.1	953.1	1278.05	1706.57	2175.07
Fast Downward, LM-cut	474.15	734.06	939.62		
Fast Downward, Blind	473.77	733.63	939.27	1205.1	1461.54
Fast Downward, iPDB	474.15	734.07	939.62	1205.51	1462.13
Popf-2	541.3	907.0	1255.4	1658.48	2045.96

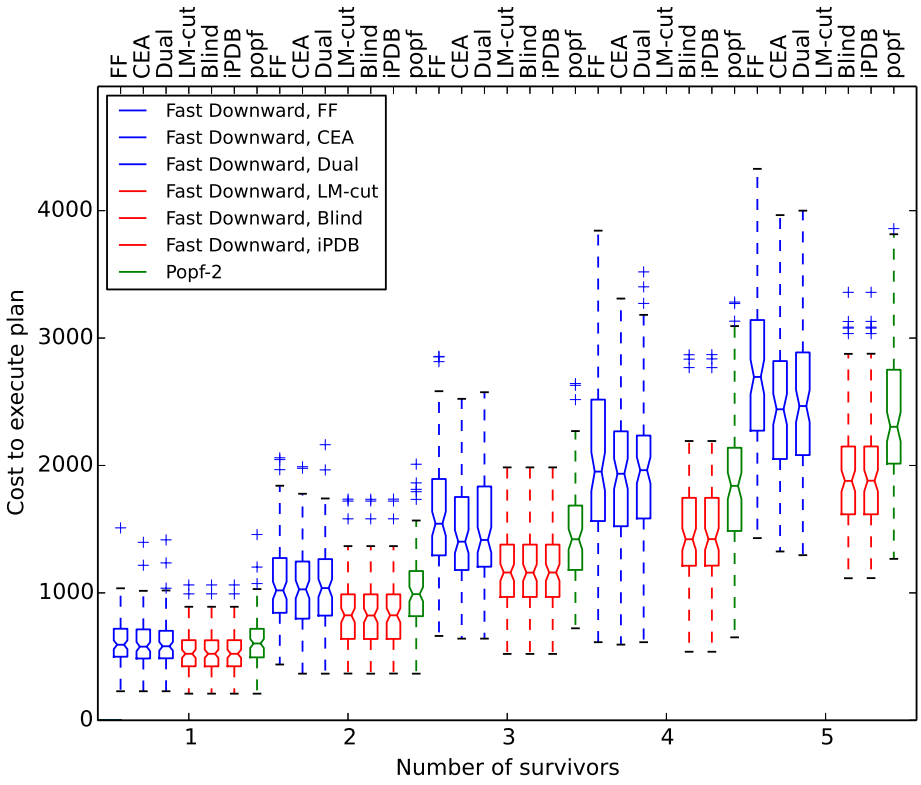


Figure 4.19: Planner cost scaling for collect and deploy actions with one supply ship and m survivors. Landmark-Cut (LM-cut) was only tested for $m \leq 3$. Total number of trials performed, $n = 3253$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

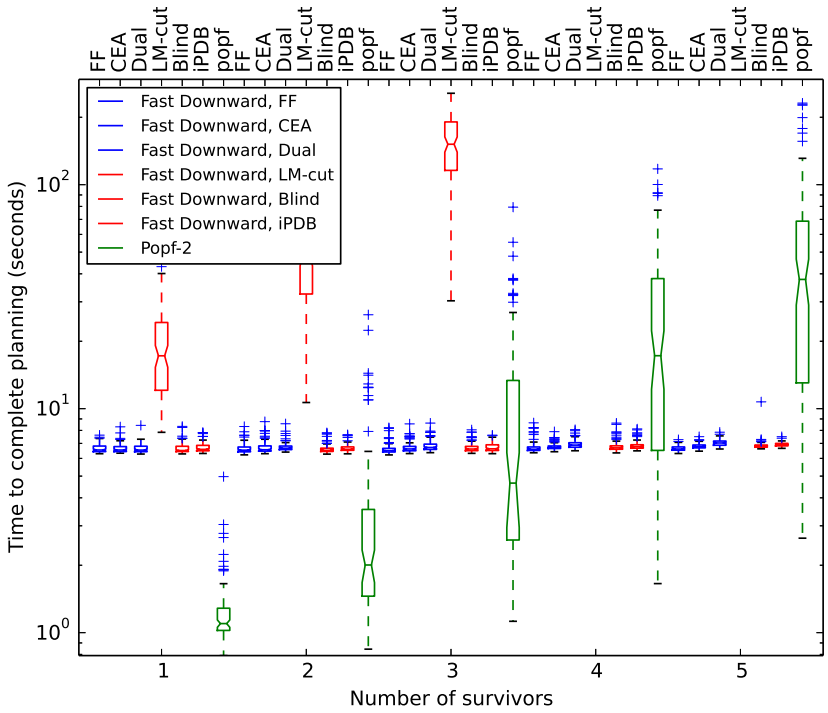


Figure 4.20: Time required to plan for collect and deploy actions with one supply ship and m survivors. Landmark-Cut (LM-cut) was only tested for $m \leq 3$. Total number of trials performed, $n = 3253$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

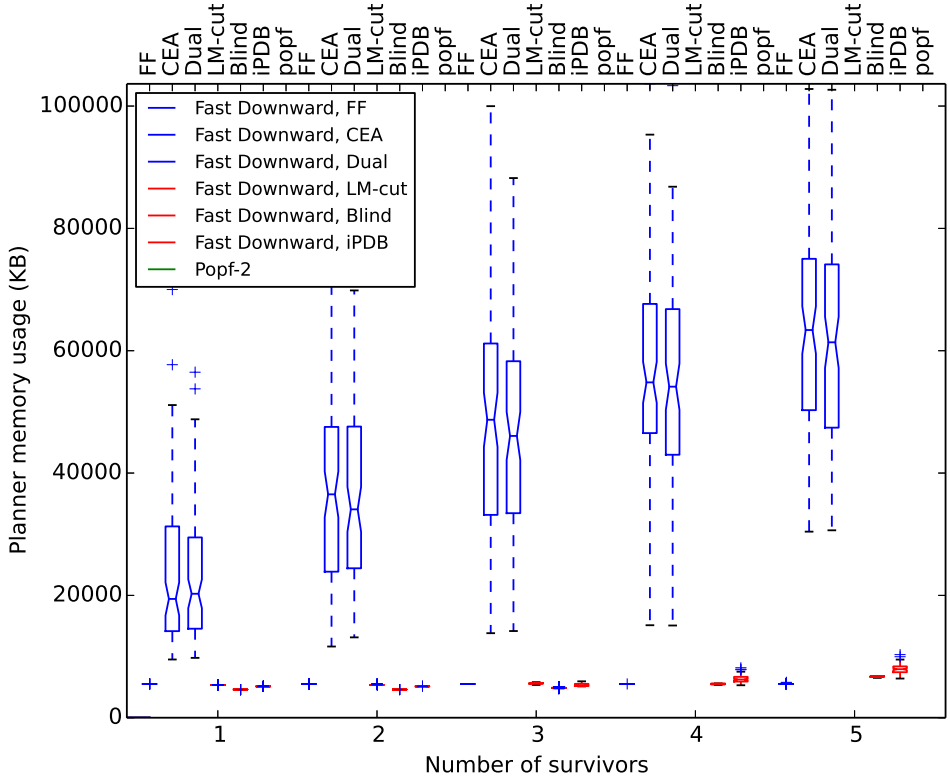


Figure 4.21: Memory required for search with one supply ship. Landmark-Cut (LM-cut) was only tested for $m \leq 3$. Total number of trials performed, $n = 2755$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

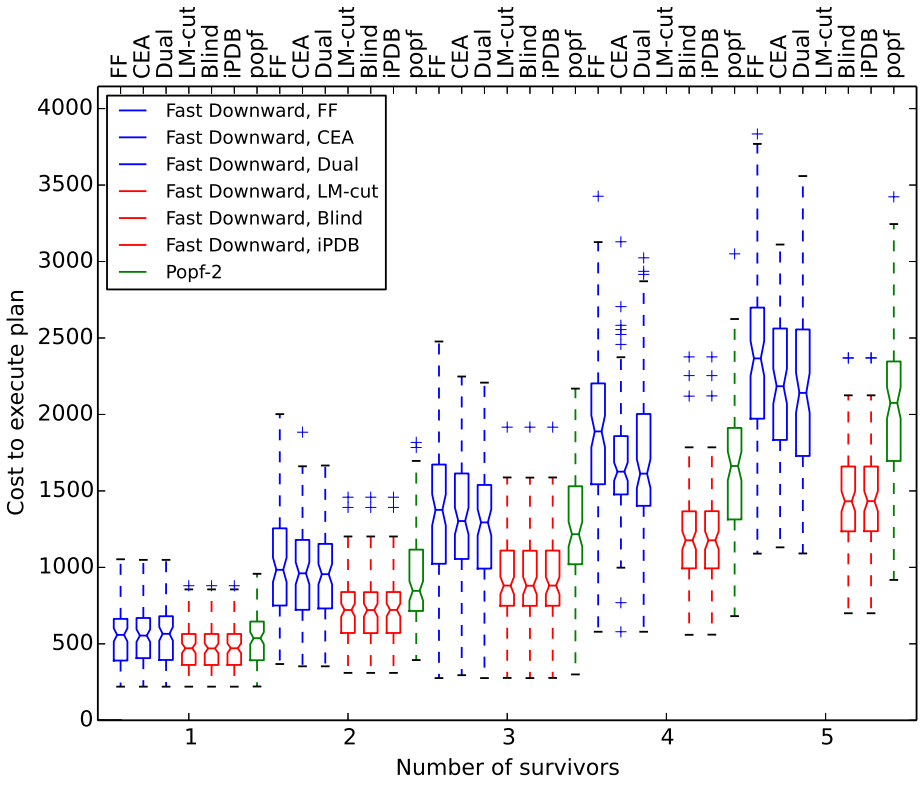


Figure 4.22: Plan execution costs with collect and deploy actions for two supply ships and m survivors. Landmark-Cut (LM-cut) was only tested for $m \leq 3$. Total number of trials performed, $n = 3273$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

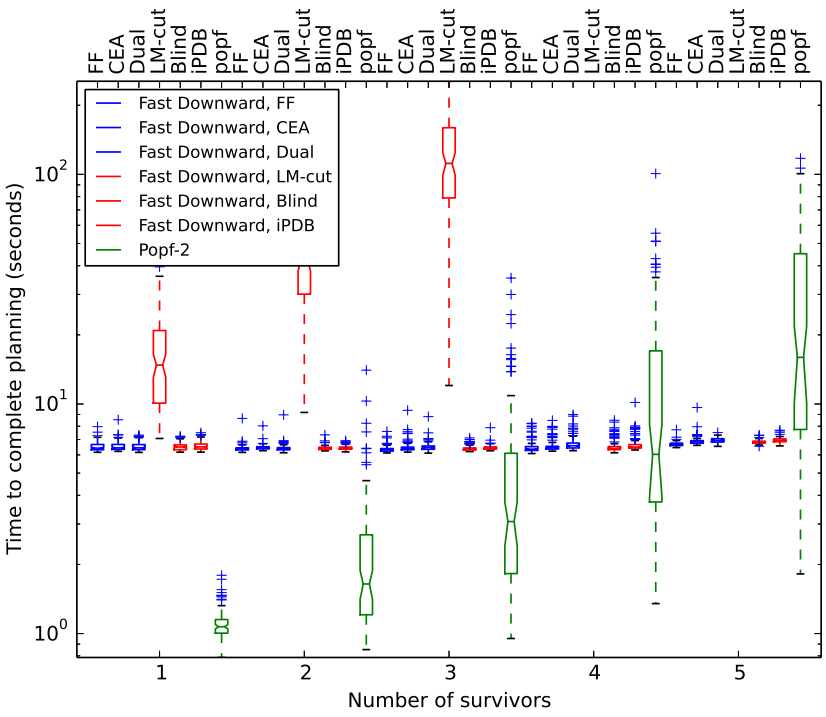


Figure 4.23: Time required to plan for collect and deploy actions with two supply ships and m survivors. Landmark-Cut (LM-cut) was only fully tested for $m \leq 3$. Total number of trials performed, $n = 3273$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

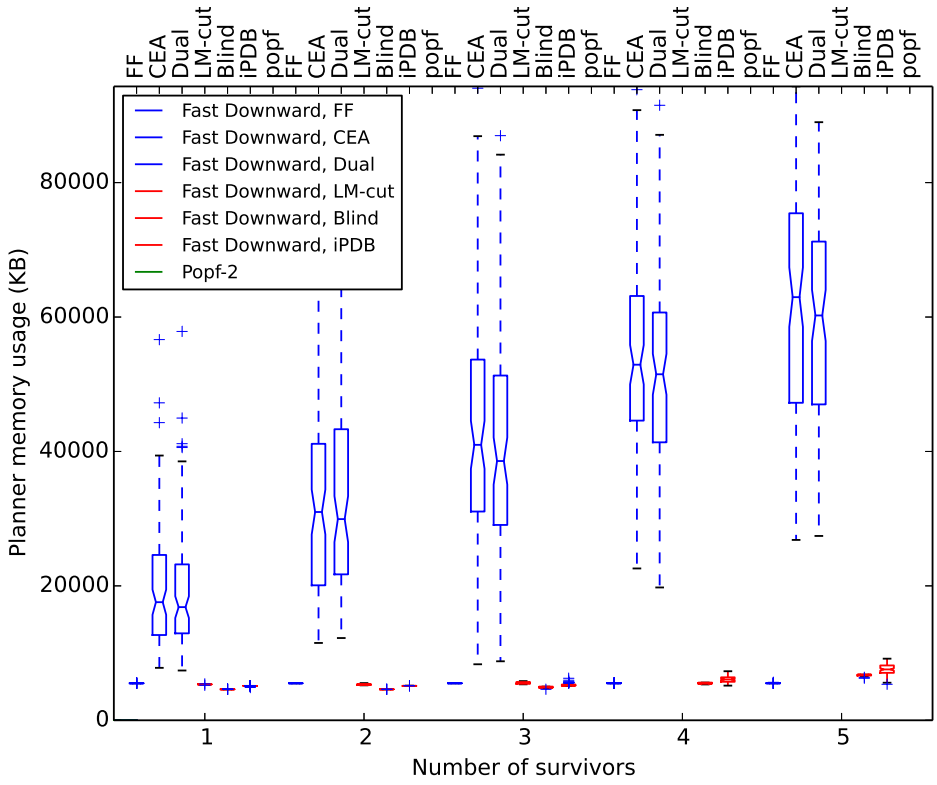


Figure 4.24: Memory required for search for collect and deploy actions with two supply ships and m survivors. Landmark-Cut (LM-cut) was only fully tested for $m \leq 3$. Total number of trials performed, $n = 2777$. Boxplot centreline indicates median, extents show upper and lower quartiles. Non-overlapping notches indicate statistically significant differences between medians.

4.2.6 Selection of Planner for Spatial Tasks

These benchmarks have shown that the Fast Downward planner can plan and order actions to both satisfy requirements and reduce plan costs. The A* search based systems using the LM-cut, iPDB and blind heuristics produced the lowest cost plans, between 15% and 30% less than the Greedy Fast Downward and Popf-2 planners in the survivor rescue scenarios. In these scenarios, The LM-cut system demonstrated rapidly increasing planning times in more complex tasks making it unsuitable for more complex tasks. A* with LM-cut or iPDB should thus produce plans that are efficient while still executing rapidly enough to be responsive to changes in the environment.

Not all planning runs were successful, with a failure rate in the order of $< 1\%$. As outlined in Section 4.2.2, all planning runs in a trial were discarded if any planning system failed to produce a result. This failure rate is within an acceptable range, but a practical robot planning system will have to monitor the output of the planner to ensure the generation of a valid plan before execution.

4.3 Conclusion

A number of planning systems have been tested for their applicability to the mission planning task in a spatial environment as outlined in Table 4.12. These experiments have shown that a domain independent planner can be used to find a low-cost path through a spatial environment to execute a task, while also satisfying mission constraints and goals.

From these tests, the Fast Downward planner with A* search and the Blind or iPDB heuristics have been identified as the most suitable for such spatial planning tasks. These produced the shortest plans while maintaining an execution time that scaled primarily with the number of spatial nodes, rather than the number of actions they were required to schedule.

With 900 spatial nodes, the Fast Downward planner averaged between 6 and 7 seconds to generate a plan when not using the LM-cut heuristic. These short planning times would allow these planners to be run off-line, using the current estimate of the environment and

producing a plan to reach the goal state.

900 spatial nodes is a very small for a robotic spatial environment. For example, the willow_garage example map from the turtlebot_navigation package [Open Source Robotics Foundation, 2014d] has $344, 128$ grid locations. In Section 4.2.3, an increase in nodes from 900 to 2500 resulted in an approximately tenfold increase in planning time, while attempting to plan with 3600 nodes caused a failure of the planner. Use of the Fast Downward planner with a full-resolution map similar to the turtlebot_navigation example would thus not be possible.

The next chapter will combine the Fast Downward planner with the belief compression systems investigated in Chapter 3 to perform planning with a real-world maritime vehicle.

Table 4.12: Summary of effectiveness search and heuristic types used with the Fast Downward planner

Search	Heuristic	Summary
Lazy Greedy	Fast Forward	Produced longest plans, low memory usage
Lazy Greedy	Context-Enhanced Additive Heuristic	Long plans, high memory usage
Lazy Greedy	Combined Fast Forward and Context-Enhanced Additive Heuristic	Long plans, high memory usage
A*	Landmark-Cut	Efficient plans, rapidly increasing planning time
A*	Blind	Efficient plans, low memory usage
A*	Pattern Database	Efficient plans, low memory usage
Popf-2		Moderate efficiency plans, short planning time in simple environments

Chapter 5

Topological Mission Planning

5.1 Introduction

The previous chapters evaluated methods for belief compression of spatial environments, and the generation of vehicle plans' action costs. This chapter will examine the implementation of a planning system for TopCat, a 5m Autonomous Surface Vessel (ASV) based on the Wave Adaptive Modular Vessel (WAM-V) hull. As with the previous chapter, the planner will be evaluated with a search and rescue task based on project ICARUS [De Cubber et al., 2013].

A number of simulations were performed with tasks of increasing complexity starting from simply visiting several positions, and increasing in complexity until the mission required the execution of multiple actions.

5.2 Planning with Spatial Constraints

In a symbolic planing system, as described in Appendix B, the planner chooses actions based on their effects to achieve a goal. These action effects are applied to the state simultaneously, as such all actions are *atomic* and instantaneous. In a real-world system, it is expected that many of these actions would require a non-trivial period to execute while the robot's state evolves from initial to final condition. To interface between these domains,

a system is required to translate between the discrete actions of the planning system and the continuous real-world that the robot's hardware is present in. This system is typically referred to as an *executive*.

As covered in Section 1.1, the approach used to integrate the STanford Research Institute Planning System (STRIPS) with Shakey was to generate a plan using STRIPS, and then dispatch actions sequentially to lower level FORTRAN programs for execution. Due to limitations in memory space, the LISP based planner was swapped out of memory before execution [Nilsson et al., 1968]. Thus, the planning layer was not operational during the operation of the executive. This model, where the planning, executive and low-level reactive systems are separated is referred to as the three layer model of planning, a conceptual diagram of which can be seen in Figure 5.1(a). Despite significant increases in computing power, systems such as ROSPLAN, covered in Section 1.5.4.2, still use this model.

As shown in Table 1.1, each action that could be executed by Shakey had a singular parameter specifying the object that they would work upon. In Shakey's workspace, shown in Figure 1.2, with its six rooms and seven doors, the number of objects to be manipulated was limited. As investigated in Chapter 4, domain independent symbolic planning systems can only operate on small spatial environments. For a planner to scale to the more complex environment expected of a field robot, a compressed spatial model is required. The topological compression techniques covered in Chapter 3 provide a possible method for reducing the complexity of an irregular natural environment to a level that is possible to be used directly by a symbolic planner, allowing the model of planning shown in Figure 5.1(b). In this model, spatial data is used to inform both the planning and executive layers.

The high-level planning in Chapter 3 differs from that used by Shakey in that the planning units are constraints, specifying the set of trajectories to be traversed rather than the specific path to be traversed. This chapter will investigate two different approaches to the integration of a symbolic planning system with spatial data for robotic planning.

5.3 Robotic Planning and the Travelling Salesperson Problem

Since many of the robot's actions require it to be in a specific spatial position, for example adjacent to a survivor requiring rescue, then the overall lowest plan cost will be the arrangement of these actions that minimises the travel cost. In this configuration, the problem represents a form of *Travelling Salesperson Problem* (TSP) [Russell and Norvig, 2010], visiting all the nodes in a graph with the minimum possible cost. Solutions to the TSP problem are NP-Hard but the limitation of using TSP solvers in real-world robotic applications appears to be the calculation of the edge costs of the TSP graph. In a planar problem with no obstacles, the travel cost between TSP nodes is equal to the Euclidian distance. However, once obstacles are introduced, then a closed form solution to the edge costs no longer exists. Systems such as ROSPLAN, and Englot and Hover's ship inspection system use RRT algorithms to calculate the path length [Englot and Hover, 2013]. However, this operation is sufficiently expensive that calculating the costs from every observation point to every other observation point, a total of $n(n-1)/2$ operations for n nodes, that the generation of a full TSP matrix is impractical for large values of n. In particular, Englot and Hover's system uses Euclidean distances for initial estimates, and only calculates true distances for edges that are expected to be on the TSP tour.

For a robotic system, there may be additional restrictions on the vehicles actions including launch and retrieval positions, limitations on vehicle range, and load capacity. Furthermore, the solution to the TSP is a *cycle*, a closed loop of edges. Thus, an optimal TSP solution may not be optimal for a mission that requires the terminal position to differ from the initial. Solving the ordering of actions with a symbolic planning system would allow these constraints to be considered in the construction of an optimal plan.

5.4 Symbolic Planning for Ground Vehicles

To test the applicability of planning with belief compression to robotic mission planning, an initial prototype using the Robotics Operating System (ROS) existing support for ground vehicles was developed. The move_base navigation stack [Open Source Robotics Foundation, 2014b] uses a three-layer architecture comprising a *reactive*, an *executive* and a *deliberative* layer [Russell and Norvig, 2010] as shown in Fig 5.1a. In the stack these three layers correspond to move_base's local planner, move_base's global planner, and the executive that dispatches goals to the navigation system. The reactive local planner uses sensor data from the environment to build short range, short duration maps of local obstacles, and creates trajectories for the robot system based on these maps and the path provided by the higher level systems. The executive layer corresponds to the global planner which uses a static or slowly updating map to produce a path from the robot's current pose to the goal pose. Finally, the deliberative layer, which may be a planning system, script or immediate commands from a human operator, dispatches goal poses to the executive layer. In this model each layer sends information to the one beneath it, but only considers the success or failure of the lower level planning system.

By using the spatial information available to the system in the goal planning stage, a modified planning system can be created where the deliberative layer has sufficient information to not only plan goals in an order that attempts to minimise the effort required to visit them, but as shown in Fig 5.1(b), also performs part of the planning function formerly performed by the executive. Since the deliberative layer has provided a constrained set of homotopic trajectories, the executive system can restrict its search to a smaller area, allowing it to reduce the number of nodes searched. This comes at the cost of potentially choosing less efficient trajectories, but as found in Chapter 3, the increase in path length is rarely significant.

The robot's goal is set using a PDDL task containing two new keywords *traversable* and *locatable* as shown in Table 5.1. These two keywords cover two basic sorts of spatial information - individual objects that may be agents, and a spatial environment represented as a graph structure that can be traversed. The *locatable* keyword also allows initialisation of all such created objects using simple PDDL statements. A program was developed

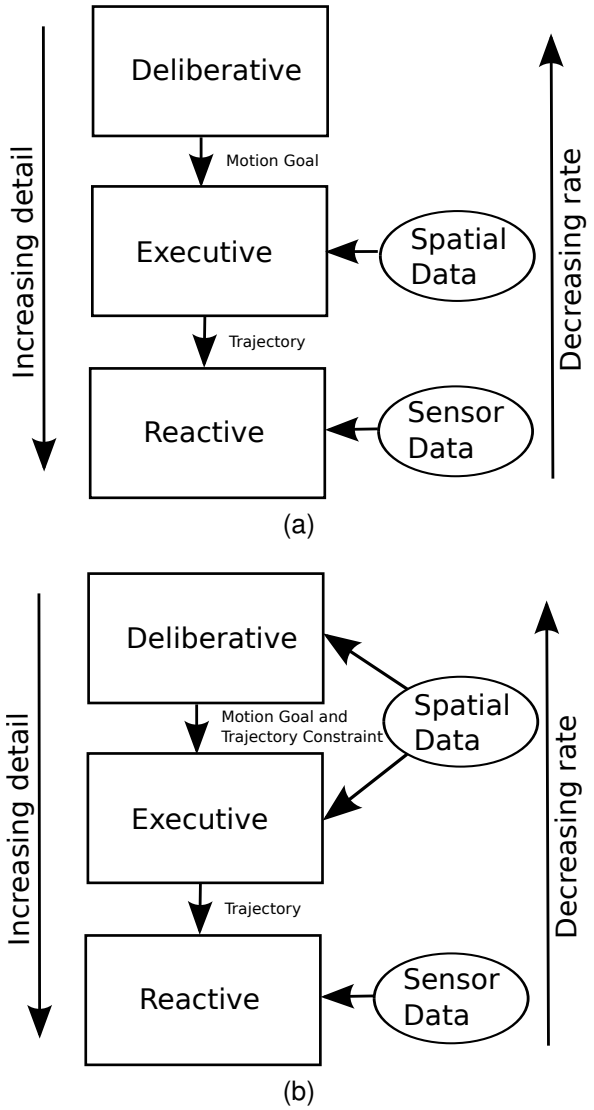


Figure 5.1: (a) Three layer planning model as implemented in the Robotics Operating System (ROS). (b) Three layer planning model modified to support a spatially aware scheduler.

that would read the PDDL files and combined with data generated by the robots navigation system would expand these keywords to produce a planning graph of the topological environment as shown in Fig 5.2(e). Action costs for traversing branches are automatically generated based on branch length, allowing PDDL solvers capable of optimising plan metrics to attempt to minimise the energy expended when performing a task. To differentiate between objects that are static and those that can perform actions, the test domain includes the predicate *canact* for all agents.

To test this strategy, a simulated environment was created with the Stage robotic simulator [Vaughan, 2008] containing a robot with a lidar sensor and a number of goal objects. This robot was interfaced with the ROS standard navigation stack [Open Source Robotics Foundation, 2014b]. The navigation system was configured to use a map generated by the planning system that contained only the area of the current cells in which the vehicle was operating, as seen in Figure 5.3. This restriction of data to lower level systems can provide an advantage when navigating in complex environments. ROS uses a sampling planner for its low level trajectory generation that scores possible trajectories using three criteria - the

Table 5.1: New PDDL keywords

Keyword	Description
traversable <map_name>	Define a graph based on a ROS map that can be used with movement commands. map_name is used to find the correct ROS service to call to obtain a map.
nodename <node_names>	specifies the type that will be used for node objects. Defaults to ”node”
edgename <edge_name> <predicate>	specifies the type used for edge objects and the predicate that will be used to show connectivity between nodes. Defaults to ”edge” and ”at”
costname <cost_name>	specifies the traversal cost of an edge. If no instances of this keyword exists, no costs will be created.
locatable <type> <name> [<predicate> ...]	Define an entity that may appear zero or more times within a planning system. An PDDL entity is created for ROS transform that matches the given name with the specified type and initialised with predicates as specified on the list.
locatablepredicate <predicate> <node_type>	allows the creation of a custom predicate associating the entity with a node. Defaults to ”at” and ”node”.

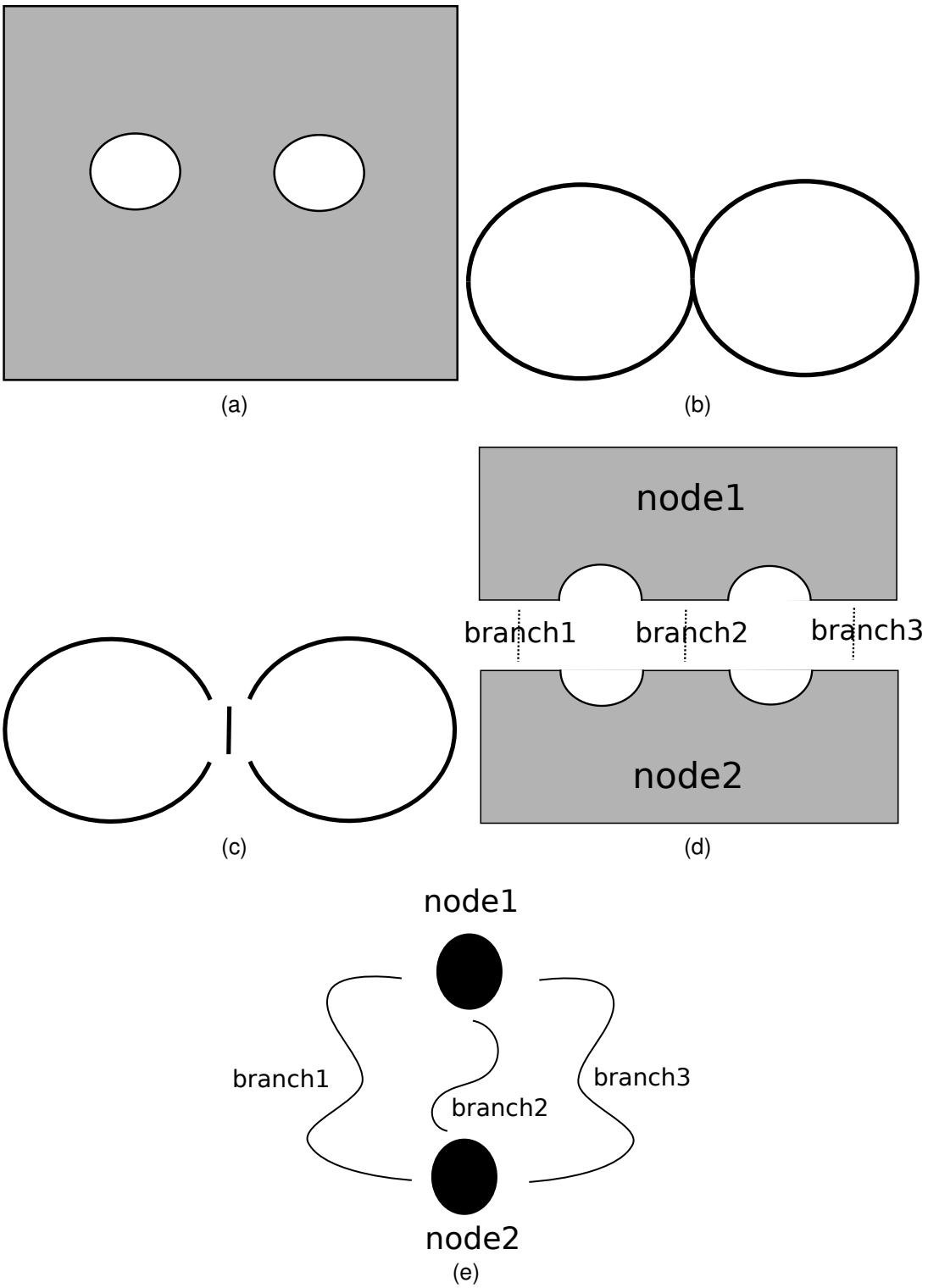


Figure 5.2: Stages in the generation of a planning graph from a volume using a skeleton. (a) Original volume. (b) Topological Skeleton derived from the volume. (c) Skeleton broken into segments. (d) Reconstruction of volume based on the location of nodes. (e) Planning graph derived from skeleton and reconstruction.

achievement of the goal, the avoidance of obstacles, and following the assigned path. In some environments, the long range goal may be on the far side of a concave obstacle from the robot's pose. If the local planner prefers trajectories that head towards its goal, it may cause the robot to be trapped within such a local concavity. As implemented, the piecemeal presentation of map and goal reduces the likelihood of such an occurrence.

Once the plan has been generated, it must be implemented by the robotic platform. The output of a PDDL planning run is a list of actions to be performed that will allow the goal to be achieved. These actions can be mapped to a ROS message that can communicate the action names and ground predicates, and the result can be used with the actionlib library to invoke execution on domain specific control and planning tasks.

This model assumes that each action is atomic, with the planner invoking each action sequentially until they are complete. To simulate the robot's ability to interact with the world, three python scripts allowing the robot to move around its environment and interact with objects were created. These scripts were exposed to the planning system using the actionlib library. PDDL actions named identically to the available scripted actions allow the planning system to generate plans.

This section showed that a deliberative planning system could be integrated with the existing ROS move_base guidance system to guide a simulated ground robot. However, the guidance and control of a maritime robot differs significantly. In particular, a maritime vehicle lacks the connection to a solid surface that allows a kinematic solution to both navigation and guidance that a wheeled or tracked robot has. To allow guidance and control of the maritime vehicle, an

Table 5.2: Actions available to the simulated robot

Action	Description	Precondition
move	Move the robot from one cell to another	Robot must be in an adjacent cell
approach	Move the robot adjacent to an object	Robot and observation point must be in the same cell
observe	Examine an object	Robot must be adjacent to an object
leave	Move the robot away from an object	Robot must be adjacent to an object

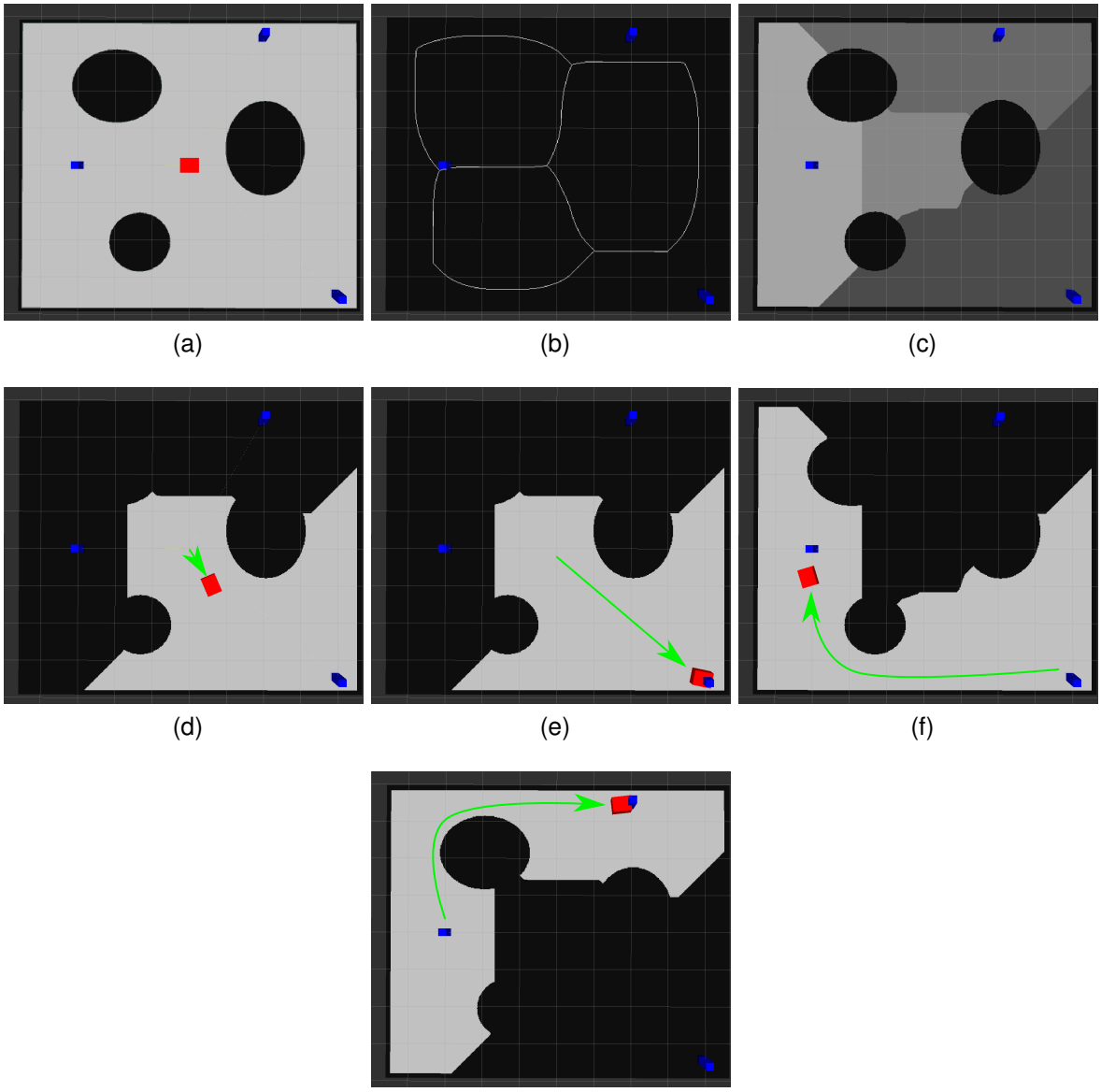


Figure 5.3: A simulated environment with a robot (red cube), three obstacles (black ovals), and three goals (blue pillars). Green arrows indicate the paths of the robot. (a) Intial configuration. (b) Skeleton. (c) Segmentation. (d) Planner has created a plan and dispatched the robot to the first goal. The map is restricted to the zones required for traversal. (e) The robot reaches the first goal. (f) The robot reaches the second goal. (g) The robot reaches the third goal.

alternate planning and executive is required. The next section will cover the implementation of such an alternate system for the TopCat ASV.

5.5 Planning for Maritime Vehicles

To demonstrate maritime planning, a new planning and navigation system was developed for the TopCat vehicle. This system incorporating a reactive layer comprising a model predictive control system, an executive layer including an object tracking system, a path generation system, and a deliberative layer incorporating a symbolic planning based mission planner. A block diagram outlining these relations can be seen in Figure 5.4. The following sections will cover this system in more detail.

5.5.1 Reactive Layer

The reactive layer receives actions as *transects*, goals that the vehicle is to approach from a particular heading. The use of this primitive is to improve the effectiveness of the vehicle at

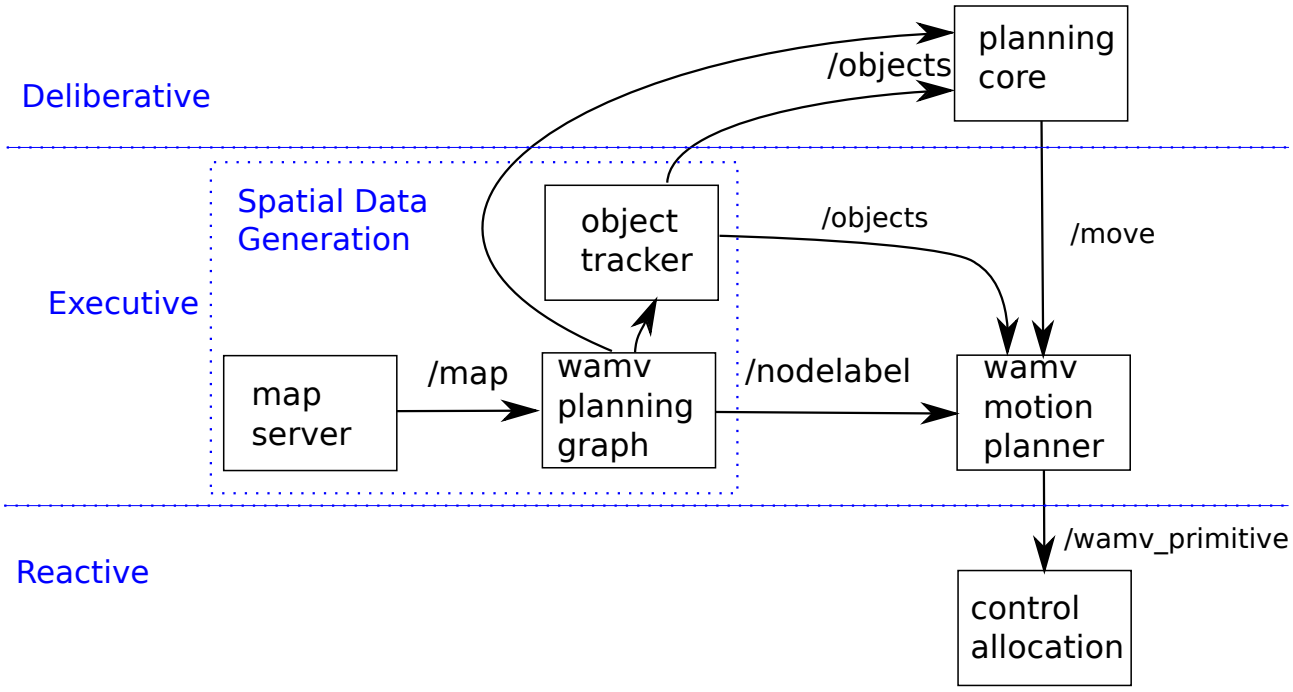


Figure 5.4: Diagram of Autonomous Surface Vessel (ASV) planning and control system. Blue text indicates the corresponding segments of the three-layer model visualised in Figure 5.1(b).

performing scientific tasks that would require the vehicle to repeatably sample a known path. As discussed by Roelfsema et al., simply minimising heading error while travelling to a goal in an environment with water currents will result in a curved trajectory [Roelfsema et al., 2015]. Following such a curved trajectory may result in incorrect sampling of an environment when repeating survey tasks. Details of this node including the method of control can be found in Appendix E.

5.5.2 Executive Layer

The executive layer uses a grid based map to both generate a compressed graph modelling the spatial environment, and also to generate collision free paths. A *map_server* node publishes the current known map of the environment, which is used by the *wamv_planning_graph* node to generate the compressed belief space of the environment, both as predicates and also labelled maps.

The *wamv_motion_planner* uses both map data and the compressed skeleton to construct motion plans. Path planning is performed by first checking for a direct collision free path to the goal. If no such path is available, a path is generated using the same grid search algorithm as used in the path length evaluation in Chapter 3. The output of this search is a set of grid squares to be traversed. These grid squares are converted into a set of transects suitable for the control allocation system by the use of a split and merge algorithm.

The executive layer also contains *object_tracker*, a system for tracking object positions. This is used by both the path generation system and the deliberative planning system. Since an object tracking system based on external data is still under development, this system currently stores static positions for objects other than the ASV.

5.5.3 Deliberative Layer

The deliberative layer comprises the *planning_core* system. This package interfaces between the executive layer and the Fast Downward symbolic planning system. One of the primary design criteria of *planning_core* was maximising the exposure of the planner's state

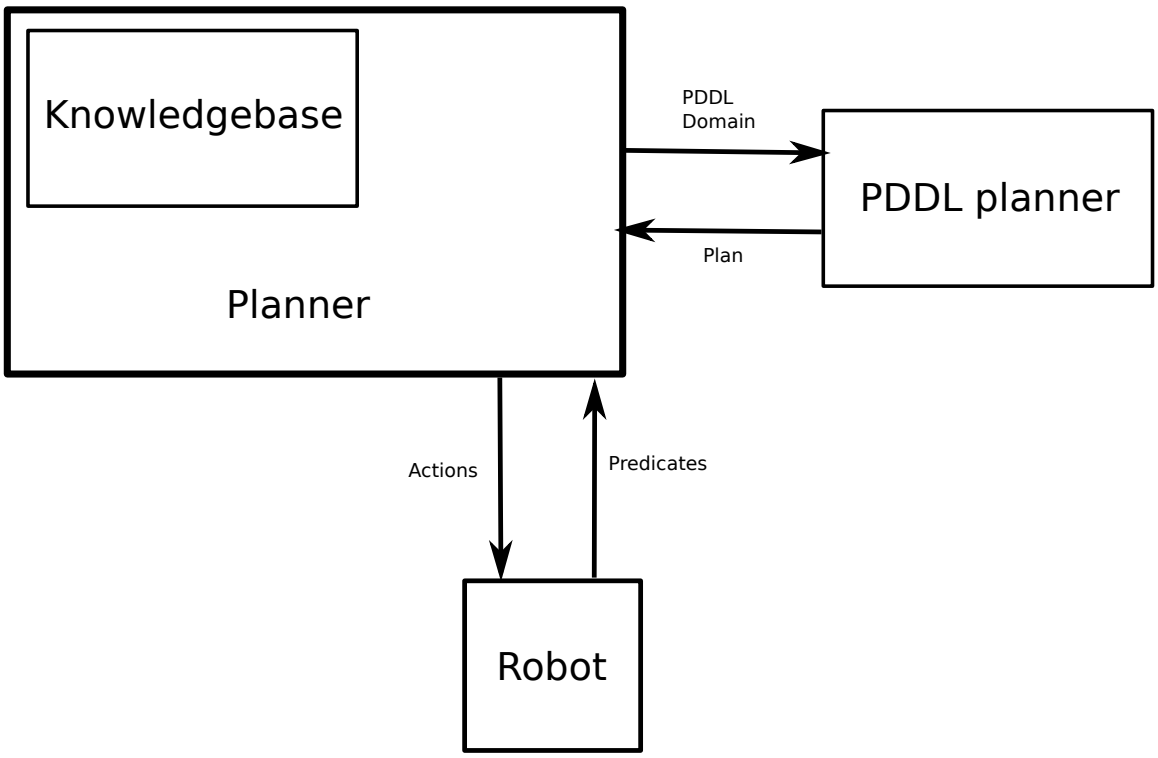


Figure 5.5: Conceptual model of planning core framework

to the ROS system, thus allowing introspection of the planning process. As demonstrated by ROSPLAN, PDDL predicates can be encoded as ROS messages. However, a limitation of the system is that the ROS message definition system does not support recursion, and as such cannot support the Boolean relations that are typically found in a planning domain. While the AND, and NOT relations are implemented, the limitations on recursion and custom message types prevents a simple implementation of an OR type relation. Full support for Boolean preconditions could have been implemented by the addition of another layer of messages and a translation to a Sum-Of-Products form, but would add further complexity to the system. Due to the rarity of OR relations, this has not limited the implementation of robotic control domains. Details on the message types used to encode this information can be found in Appendix C.

Predicates, functions, actions, and the initial state can be populated from PDDL files or using ROS messages. Once loaded, this state can be used to either generate domain and task files, or to manually apply actions to the belief to predict future states. During operation, the state of the environment is communicated to the planning system by ROS topics using *Objects* messages, the format of which is covered in Section C.4.5, while the generated plan is published as *Plan* messages that are described in Section C.4.6.

As discussed in Section 5.2, the classic model of plan execution is to dispatch actions sequentially to lower level domain specific planners. This is the approach used by executives from Shakey to ROSPLAN. However, since *planning_core* is intended to inter-operate with the topological belief compression system discussed in earlier chapters, the nodes of the spatial graph encode not movements, but constraints. Each node specified by the move action stating which homotopic segment should be used when searching for an optimal trajectory. In this model a sequence of node traversals followed by a move to a waypoint must be interpreted as a move to the waypoint within the constraint specified by the regions associated with the nodes. The algorithm used to perform this action grouping is shown in Algorithm 8.

5.6 Planning for an Autonomous Surface Vessel (ASV) with planning_core

As covered in Appendix A, field testing was performed of the reactive and executive layers of the planning system. Unfortunately, due to logistical reasons it was not possible to do an all-up field test of the planning system. However, as covered in Appendix F, a simulated ASV, shown in Figure 5.6, was available that emulated the behaviour of the real vehicle. This simulated vehicle was combined with a model of the West Lakes environment developed from spatial data as shown in Figure 5.7.

To provide the required spatial data needed to inform the deliberative and executive layers, a number of data sources such as OpenStreetMap imagery [Open Street Map, 2015b] and previous vehicle runs were used. Notably, a number of buoys exist in West Lakes that are not shown in standard maps. These include two sets of small marker buoys used to mark competition lanes, and a larger buoy used to mark the presence of overhead power lines. Using TopCat's navigation RADAR, a map of the boating lake and northern portion of the eastern branch of West Lakes was created as seen in Figure 5.8. This map data was used to exclude areas around these buoys. The traversable environment of West Lakes was described using a vector spatial layer, a rendering of which can be seen in Figure 5.9.

Algorithm 8 Dispatch move actions as a batch from the plan A

```
function DISPATCHACTIONS(A)
    dispatch_action ← ∅
    action_index ← 0
    while isMove(A[action_index]) do
        dispatch_action ← dispatch_action ∪ A[action_index]
        action_index ← action_index + 1
    end while
    if isSpatial(A[action_index]) then
        dispatch_action ← dispatch_action ∪ A[action_index]
    end if
return dispatch_action
end function
```

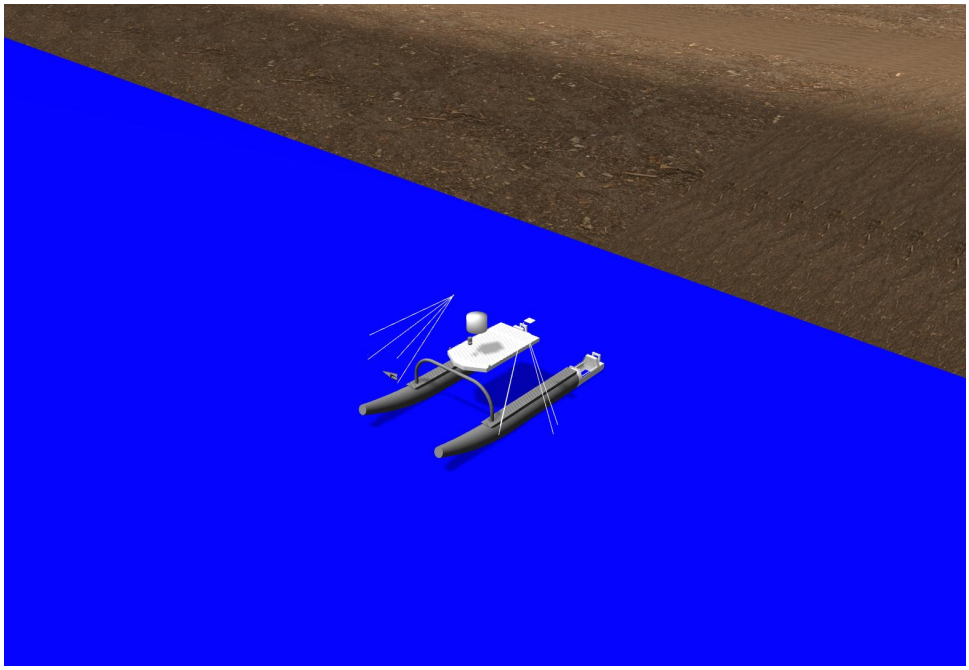


Figure 5.6: Top Cat vehicle in simulation. Computer Aided Design (CAD) models developed by Tenzin Crouch and Jesse Stewart

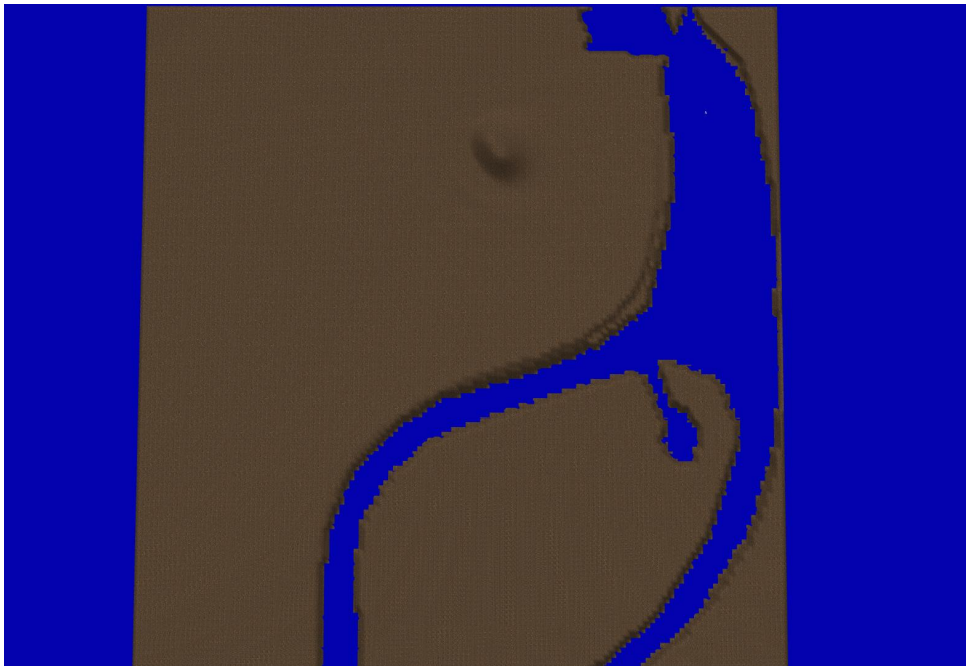


Figure 5.7: West Lakes simulation environment. Heightmap derived from 10m Digital Elevation Model [Keane, 2016]

Figure 5.8: RAdio Direction And Ranging (RADAR) map of West Lakes. White areas are accumulated strong RADAR returns. Backing image ©OpenStreetMap contributors.

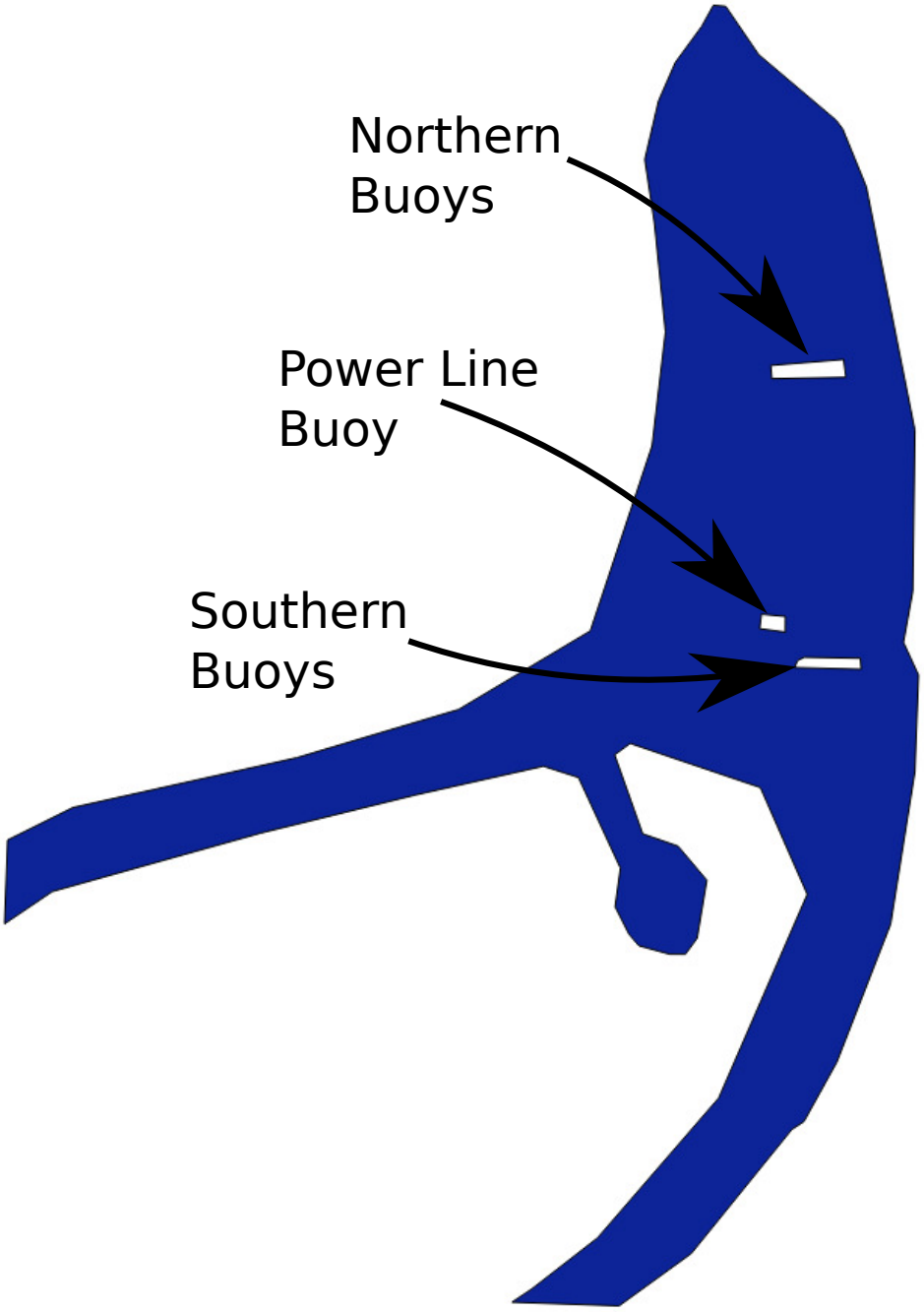


Figure 5.9: Vector map of the traversable area of Northern West Lakes as rendered using the QGIS package. Annotations show the exclusions around the buoys mapped in Figure 5.8. More information on QGIS and other Geographical Information Systems can be found in Appendix G

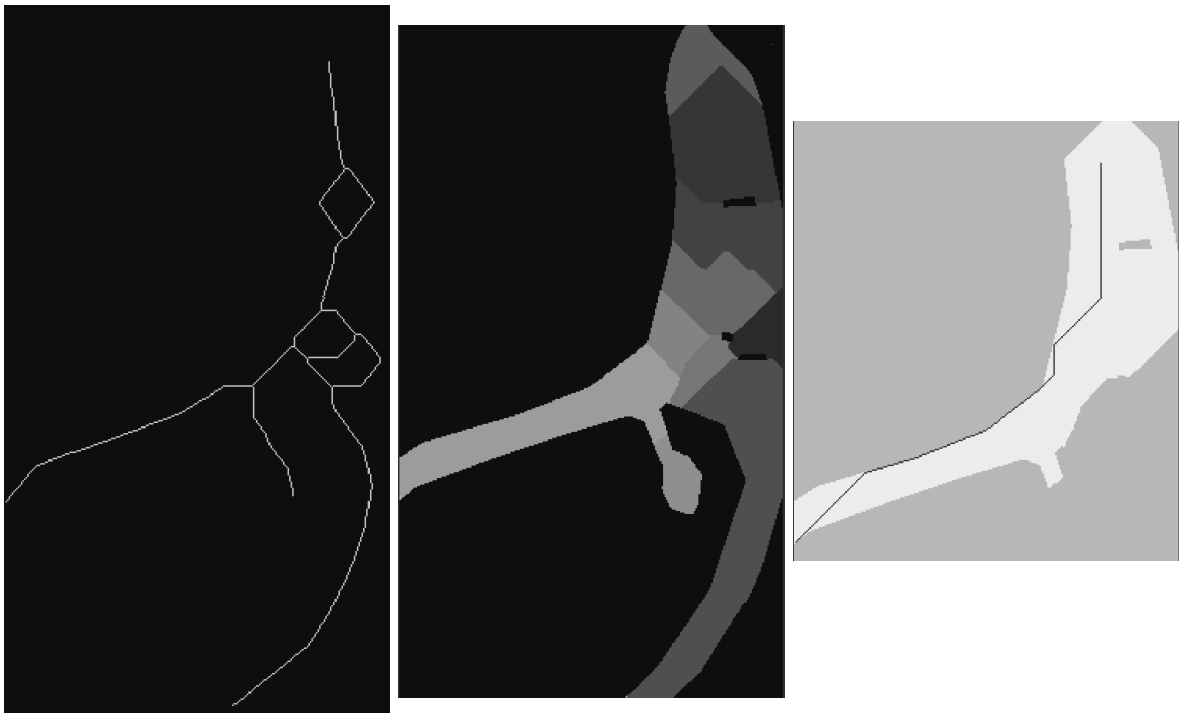
To demonstrate the integration of the planner with the motion planning system, a simple simulation was performed. The vehicle was initialised in the north of the boating lake area shown in Figure 5.9. The planned goal was to navigate down to the southwestern extent.

The planning process begins with the generation of a skeleton of the spatial environment. This is performed by the *wamv_planning_graph* node, using the Palágyi and Kuba [Palágyi and Kuba, 1999] algorithm and watershed reconstruction. The resultant graph is published on a ROS topic as an *Objects* message the details of which can be found in Section C.4.5. A visualisation of the skeleton can be seen in Figure 5.10a, while the reconstructed segments can be seen in Figure 5.10b.

Once the skeleton and node information is available, the object tracker node can use it's spatial information to generate it own list of objects with predicate information. This information is also published as an *Objects* message, an example of which is shown in Figure C.12. Using this information combined with PDDL code encoding the *move* action loaded from a text file, The planning core system was able to successfully generate a plan that dispatched the vehicle to the goal node. The generated plan is shown in Table 5.3. This plan required the traversal of several goal nodes, as such the movement commands were appended together into a single action and dispatched to the wamv_motion_planner as shown in Table 5.4. This action is dispatched using the actionlib library [Open Source Robotics Foundation, 2014a] to the *wamv_motion_planner* node using the *PDDLAction* action shown in Figure C.15. This node produces a path through the specified segments as shown in Figure 5.10c, this path is converted to line segments, and the result dispatched to the *control_alloc* node which then drives the vehicle to the goal.

Table 5.3: Plan generated by Fast Downward to move from Node 11 to Node1.

move wamv node11 node10 bridge9 (55)
move wamv node10 node7 bridge10 (50)
move wamv node7 node5 bridge5 (27)
move wamv node5 node3 bridge4 (34)
move wamv node3 node1 bridge1 (186)



(a) (b) (c)

Figure 5.10: Three stages in the construction of a motion path from a set of movement actions for the map shown in Figure 5.9. (a) Palágyi and Kuba skeleton of the North end of West Lakes. (b) Reconstructed segments of the North end of West Lakes. (c) Constrained map and vehicle path generated by the wamv_motion_planner node.

Table 5.4: Action dispatched by Planning Core based on plan

sending goal: ['wamv', 'node11', 'node10', 'node7', 'node5', 'node3', 'node1']

5.7 Scheduling of Rescue Tasks with planning_core

The *planning_core* planning system has demonstrated the ability to interface with the wamv_motion_planner system used for generating motion plans for the vehicle. The intention of this design was to allow the use of symbolic data with spatial information to allow the informed planning of tasks. In previous chapters, the Palágyi and Kuba skeleton [Palágyi and Kuba, 1999] was shown to efficiently compress the spatial belief space of a robot, while the Fast Downward planner [Helmert, 2006] with asymmetric cost information and the Blind or iPDB [Haslum et al., 2007] heuristics was shown to produce low cost motion and mission plans. *planning_core* allows these techniques to be combined to produce schedules for the ASV vehicle.

To simulate a task similar to EU-ICARUS, a simulated sinking vessel, and a pair of rescue vessels were added as shown in Figure 5.11. These vessels block traversal of the vehicle, increasing the complexity of the environment. As with the simulations in the previous chapter, it is assumed that another vehicle has already performed the search task, locating and localising the survivor's positions. In this simulation, nine survivors were randomly placed in the traversable area, represented as fixed objects within the simulation. The maritime vehicle was also randomly placed, and given the task of visiting each survivor and performing a *rescue* operation. The planning system uses the same PDDL based environment developed for the rescue task in Chapter 4.

5.7.1 Method

As with Chapter 4, a number of survivors were spread in the environment and the ASV was tasked with visiting each survivor with the least cost. The control system was configured to produce a steady velocity of $1ms^{-1}$, thus a lowered action cost should result in a reduced distance covered by the vehicle, and a similarly reduced mission time.

To provide a baseline, a simple Greedy planner was developed. This planner finds a set of actions that satisfies the goal by selecting the action a from the set of actions A that has the lowest cost at each step, similar to the Blind heuristic covered in Section 4.2.1.7. The

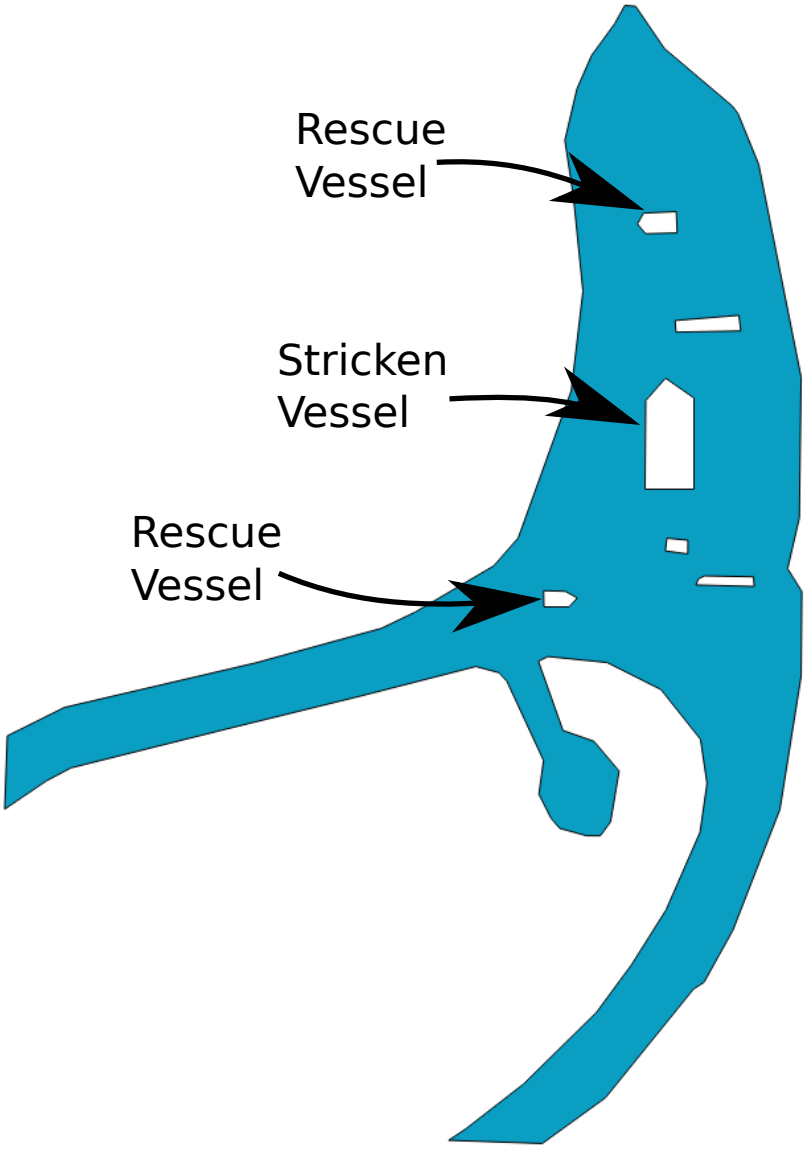


Figure 5.11: Vector map of the traversable area of Northern West Lakes as rendered using the QGIS package. In this figure, exclusion areas have been added for rescue scenario. More information on QGIS and other Geographical Information Systems can be found in Appendix G.

selected action is executed, and then removed from the set of available actions. The process repeats until there are no longer any actions to execute. Unlike the Symbolic planner that calculates the cost based on the distance between nodes, the Greedy planner uses the Cartesian distance between the current position of the ASV and the goals as shown in Equation 5.1.

$$a = \min_{a \in A} |\overrightarrow{X_{wamv} X_a}| \tag{5.1}$$

Since the Greedy Planner only considers the current action, the overall mission cost will not be minimised since the planner does not consider the global cost of its plan. Thus, the algorithm is *locally optimal*, but the complete plan is not guaranteed to be *globally optimal*. Johnson and McGeosh, who refer to this algorithm as *Nearest Neighbour*[9], provide a bound of the ratio of optimal to maximum path length based on the number of locations to visit, as shown in Equation 5.2 [Johnson and McGeoch, 1997]. However, this bound assumes that the *triangle inequality* to be true, which does not hold for environments that contain obstacles.

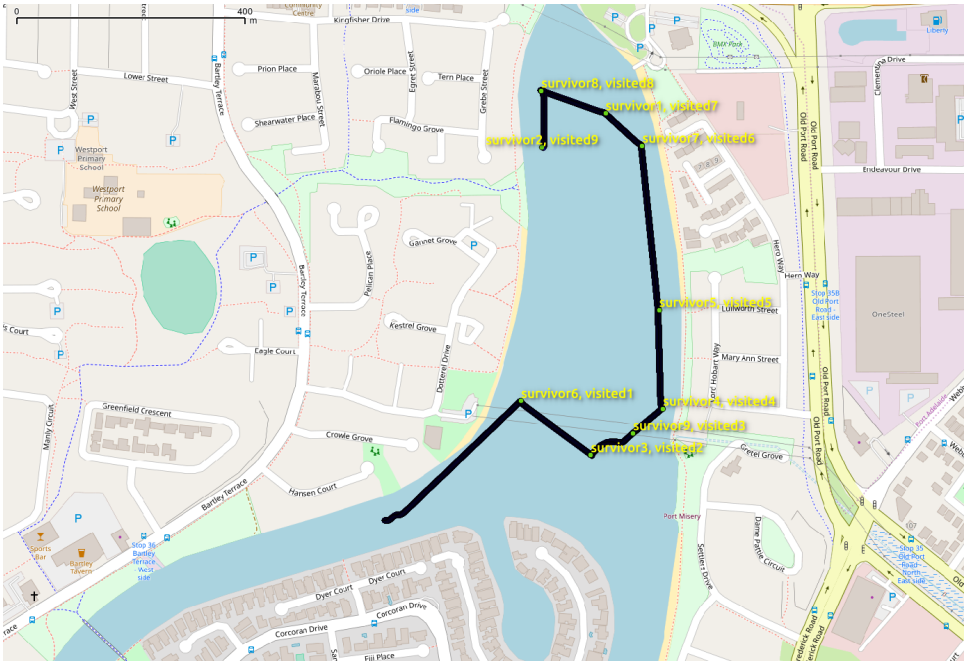
$$d_{nearestneighbour} \geq (0.5 * ((\log n) + 1)) * d_{optimal} \tag{5.2}$$

5.7.2 Results

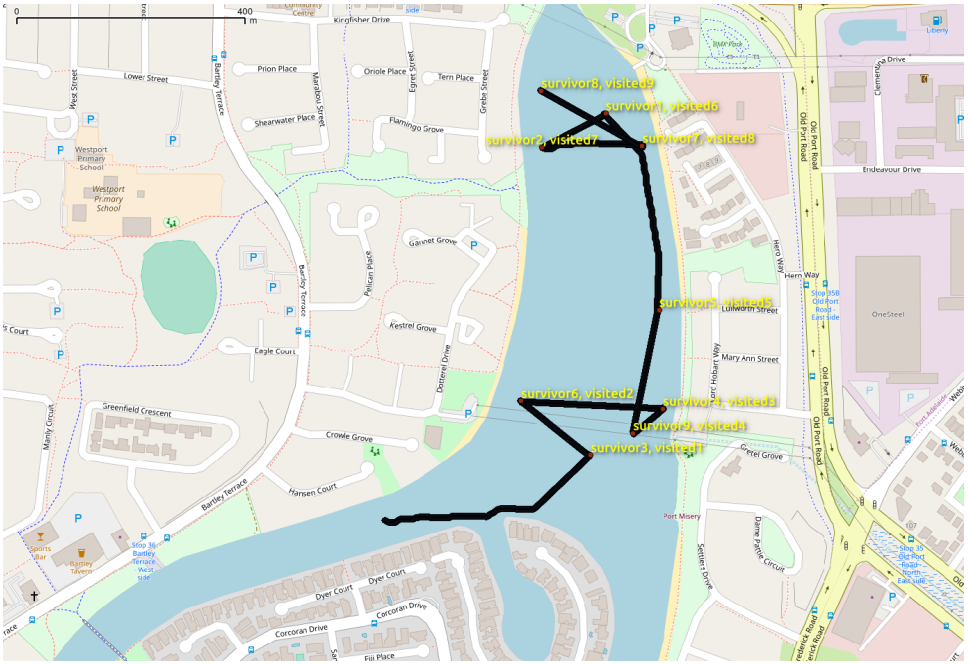
To provide a sample of execution times and costs, the simulation was repeated five times with the position of the survivors randomised after both planners had executed. A summary of the results can be seen in Table 5.5. Visualisations of the vehicle track and action ordering for the fourth test run can be seen in Figure 5.12. The tracks of the remaining runs can be seen in Appendix D, in Figures D.10 to D.13.

Examination of the results shows that runs 1, 3, and 5 show similar performance for the Greedy and Symbolic Planner algorithms, while on runs 2 and 4 the distance travelled by the ASV under Symbolic planner control exceeded the distance travelled under Greedy planner

[9] Johnson and McGeosh use Greedy to describe an algorithm that finds the closest pairs in a tour first



(a)



(b)

Figure 5.12: Vehicle tracks for the fourth task configuration (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. Backing image is ©OpenStreetMap contributors.

Table 5.5: Time used and distance covered by TopCat vehicle when executing simulated rescue

Run	Greedy		Symbolic		Difference	
	Distance	Time	Distance	Time	Distance	Time
1	2016m	2016s	2061m	2091s	2%	4%
2	2658m	2513s	3461m	3402s	30%	35%
3	2704m	2724s	2768m	2782s	2%	2%
4	1154m	1157s	1650m	1712s	43%	48%
5	2020m	1990s	2149m	2182s	6%	10%

control by 30% and 42% respectively.

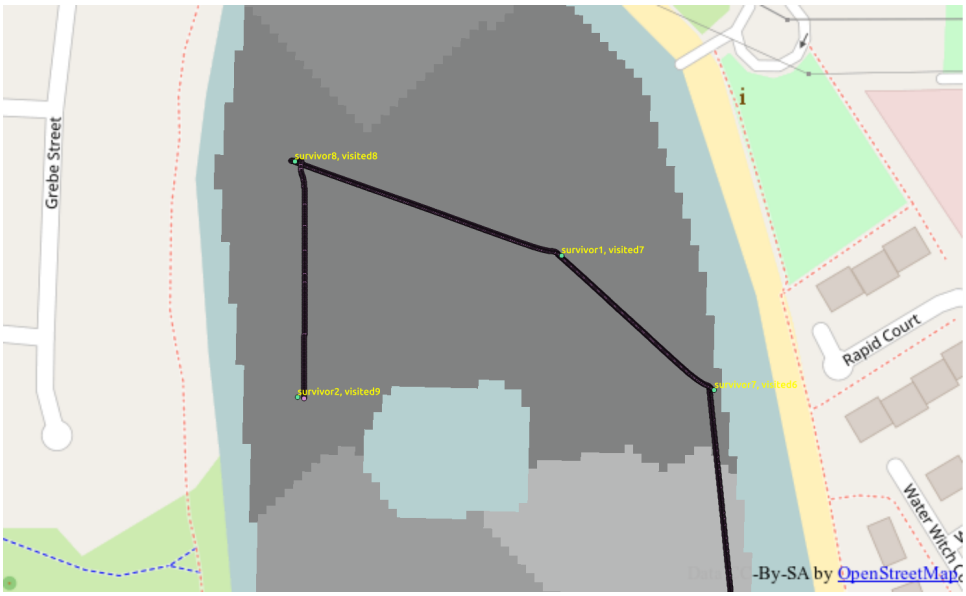
Examination of Run 4 shows that both the Greedy algorithm shown in Figure 5.12(a), and the Symbolic Planner shown in Figure 5.12(b), travelled anti-clockwise around the operating area, but the ordering of actions differs between algorithms. In particular, Survivors 3 and 6, 4 and 9 are rescued in reverse order. As shown in Figure 5.13, Survivors 1, 2, 7 and 8 share the same node. Since the Symbolic planning system treats all such nodes as single locations, no optimisation of the ordering of rescue actions can take place.

These results show that while the topological thinning can effectively compress a problem for effective planning, this plan may still not be optimal. In particular, short range spatial relationships between actions may result in the sub-optimal scheduling of the actions to be performed.

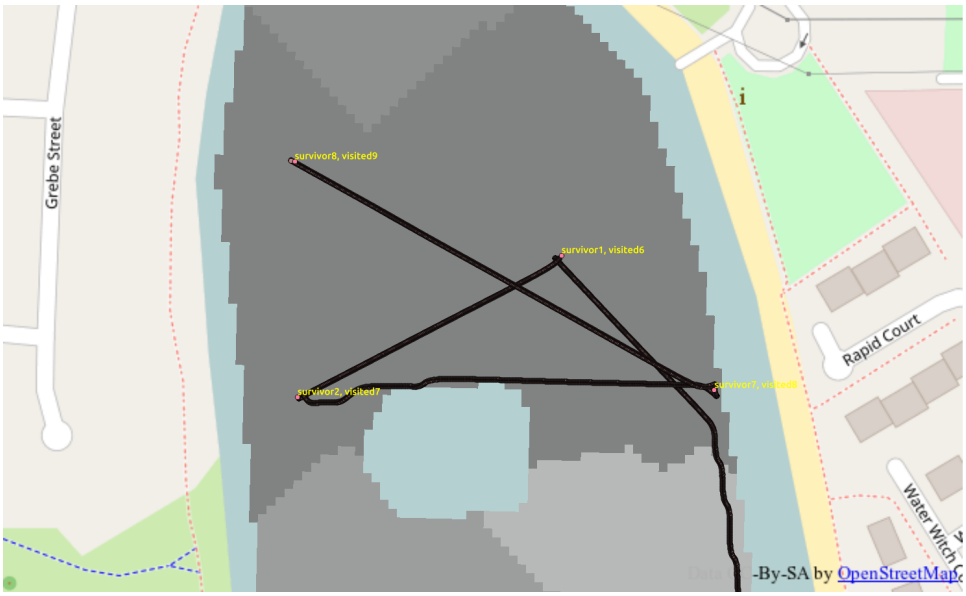
5.7.3 Conclusion

This section has demonstrated that the *planning_core* system can be used to schedule a number of actions in a spatial environment. However, the execution time and distance were inferior to performing the same task with a simple Greedy planner.

The high-level scheduling system was capable of producing low-cost plans, but since the internal arrangement of each node was not efficiently laid out, the overall result was poor. However, a concept that is used in the solution of the TSP, is *plan refinement* - taking a plan that is sub-optimal and re-ordering it to improve its effectiveness. The next experiment will attempt to use such a plan refinement strategy to improve the solution.



(a)



(b)

Figure 5.13: Magnification of Figure 5.12 showing the northern boating lake for the fourth task configuration. Shading shows the segmentation generated by the node reconstruction. (a) Vehicle track with Greedy Planner (b) Vehicle track with Symbolic planner. Backing image is ©OpenStreetMap contributors.

5.8 Planning with Refinement

In the previous section, the efficiency of deliberately planned missions using topological belief compression was compared against the best first method using geometric information. While the planner was capable of global optimisation, the simplification made to its model to allow efficient search removed data that was relevant to the local optimisation problem.

5.8.1 Method

One method for the solution to the TSP is the reduction of the global problem to a set of local optimisation problems. For example, Karp used a system of spatial subdivision to reduce the overall problem to a set of tractable sub-problems [Karp, 1977]. Similarly, the plan produced by the Symbolic Planner could likewise be used to identify sub-tours which could then be optimised by a lower-level planning system, a form of *plan refinement*. Since the Symbolic planning system is optimising globally, the final plan could be improved by refining the initial plan by local optimisation.

To allow the handling of sub-tour optimisation, the execution model was updated to dispatch sequential rescue actions as a single unit to the lower level planner. This planner would then use the Greedy planner from Equation 5.1 to choose the order to execute rescue actions within the space provided by the spatial constraints. The resulting system will be referred to as the Symbolic with Refinement planner. An updated model of the planning system can be seen in Figure 5.14. Note that in this model, the Greedy planner system does not receive information on the general spatial environment; it only optimises the current set of rescue tasks within the provided spatial constraints.

5.8.2 Results

The previous experiment was repeated with this updated model, with the results being shown in Table 5.6. Visualisations of the vehicle track for trial 3 can be seen in Figure 5.15. The remaining tracks can be seen in Appendix D, in Figures D.14 to D.17.

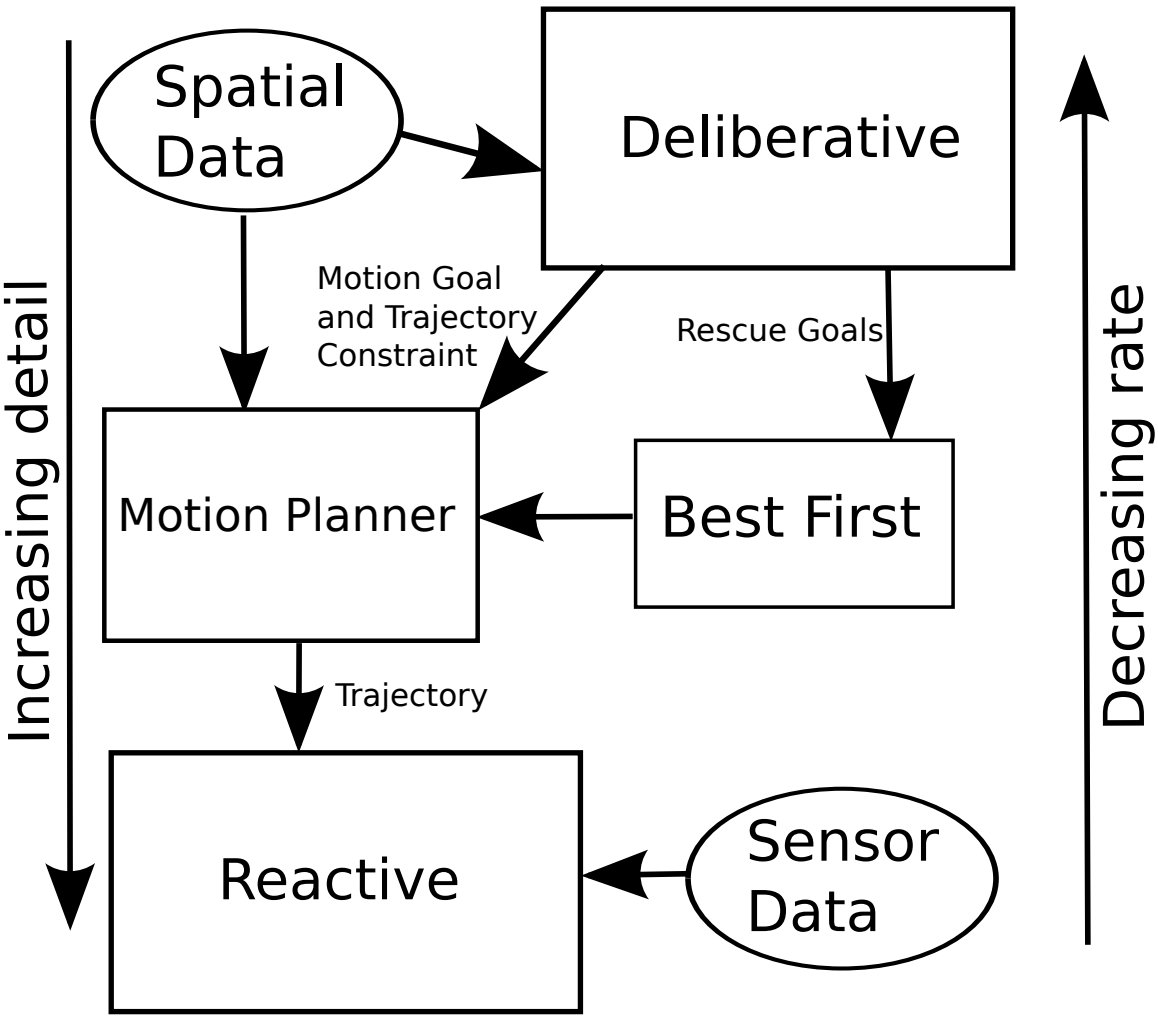
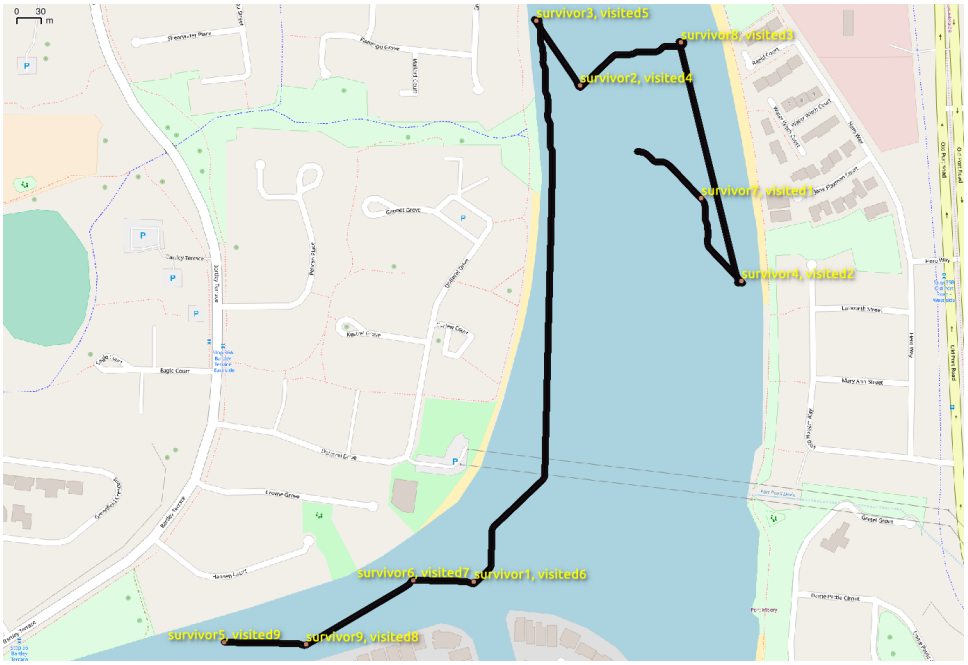


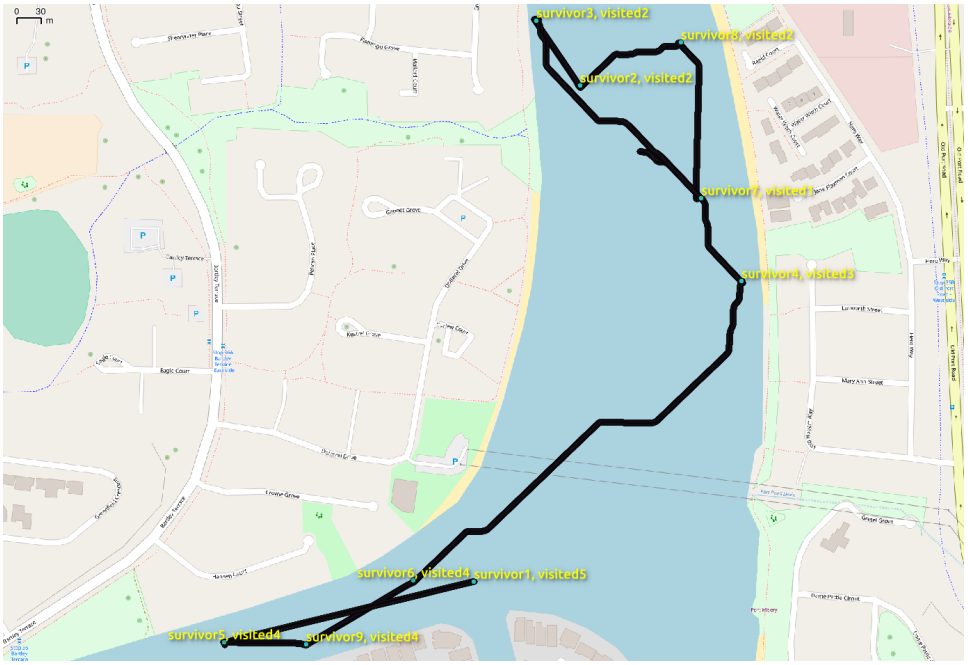
Figure 5.14: Three-layer model of autonomy for Symbolic With Refinement planner

Table 5.6: Comparison of time used and distance covered by TopCat vehicle when executing simulated rescue using Greedy planner vs Symbolic With Refinement planner.

Run	Greedy		Symbolic With Refinement		Difference	
	distance	time	distance	time	distance	time
1	2168m	2078s	2054m	1971s	-5%	-5%
2	2789m	2683s	2461m	2501s	-12%	-7%
3	1500m	1459s	1753m	1762s	17%	20%
4	1583m	1624s	1584m	1631s	0%	0%
5	1685m	1655s	1657m	1671s	-2%	1%



(a)



(b)

Figure 5.15: Vehicle tracks for the third task configuration. (a) Vehicle track with Greedy Planner (b) Vehicle track with Symbolic With Refinement planner. Backing image is ©OpenStreetMap contributors.

Examination of the results shows that except for run 3, most of the runs were similar in execution time to the purely Greedy planned system. Run 3 took approximately 20% longer to execute. Examination of Figure 5.15 shows that between survivors 4 and 1, a significant amount of backtracking was performed. This may have contributed to the increased cost of the planned mission.

In this experiment, the task of rescuing survivors did not require the consideration of preconditions, thus the problem closely matched the TSP. In a problem where some actions have preconditions requiring satisification, a planning system that considers the global problem should provide an improved solution over only local optimisation.

5.9 Scheduling of Rescue Tasks with Preconditions

The previous section showed that a domain independent planner could be used to schedule visits to multiple goals. However one of the strengths of a domain independent planner is the ability to satisfy preconditions of actions. Chapter 4 showed that a planner could combine preconditions with a spatial domain to generate paths while minimising costs. By combining spatial compression with the planning domains demonstrated in Section 4.2.5, complex plans can be scheduled.

In this experiment, a further goal is added. At the end of the mission, the maritime vehicle must dock at the north end of the boating lake to deliver the survivors to awaiting facilities.

5.9.1 Method

In the planning system, a new action *dock* was added. This took an entity with the grounded predicate *isdock* as a parameter. An object with this predicate called *wamvdock* was placed at the north end of the boating lake. This action was forced to be last by deleting the *canact* predicate of the ASV vehicle when it is executed. Since *canact* is a precondition of all other actions, this dock task must be performed last. PDDL code encoding these predicates and action can be found in Appendix C.

The Greedy Planner was also modified, by adding a move command to the *wamvdock* object at the end of its tour of the survivors. Once again, five runs were executed with the position of the survivors and vehicle randomised between runs. Results of these runs are tabulated in Table 5.7, while visualisations of the vehicle track for the first and third trials can be seen in Figures 5.16 and 5.17. The remaining trials can be seen in Appendix D in Figures D.18 to D.20.

5.9.2 Results

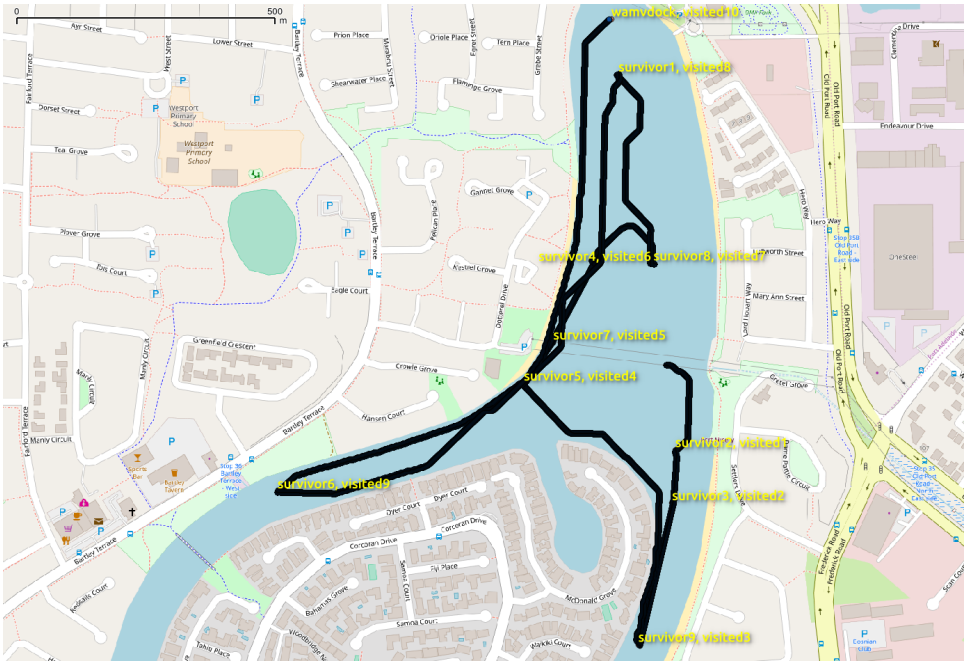
In this configuration, the Symbolic planner system outperformed the Greedy planner significantly in four out of the five runs, producing travel distance between 25% and 40% shorter than the missions planned with the Greedy planner. Since the Greedy planner did not consider the distance from the last point to the dock when ordering actions, four out of the five missions required a significant travel distance from the tours end to the dock as shown in Figure 5.16. The exception is the third mission shown in Figure 5.17, which visits the survivors in the same order as the planned mission. Since the Greedy planner does not consider the global cost of the generated path, this ordering of actions is only coincidental.

5.10 Large-Scale Testing of Scheduling with Preconditions

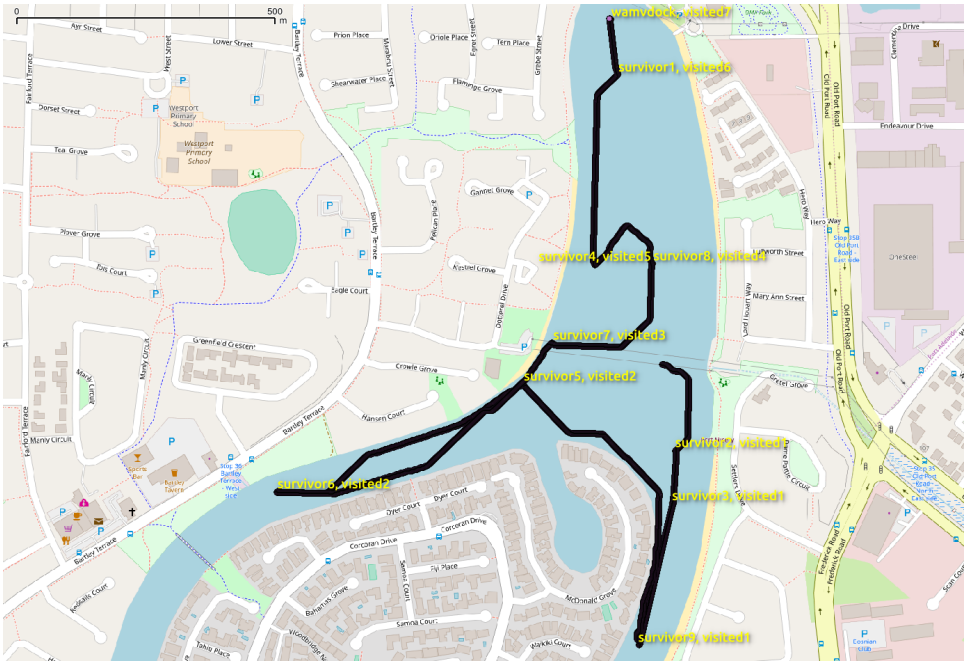
The previous section showed that the Symbolic Planner system could outperform the Greedy Planner system when preconditions were in place on the mission. However, the limited trials prevent this from being a truly rigorous test. To produce a rigorous result, a larger scale test

Table 5.7: Time used and distance covered by TopCat vehicle when executing simulated rescue.

Run	Greedy		Planned		Difference	
	distance	time	distance	time	distance	time
1	3819m	3602s	2849m	2812s	-25%	-22%
2	2641m	2576s	1814m	1882s	-31%	-27%
3	2256m	2226s	2258m	2272s	0%	2%
4	3739m	3696s	2256m	2282s	-40%	-38%
5	3982m	3761s	2458m	2472s	-38%	-34%



(a)



(b)

Figure 5.16: Vehicle tracks for the first task configuration. Vehicle must dock at the end of mission (a) Vehicle track with Greedy Planner (b) Vehicle track with Symbolic Planner. Backing image is ©OpenStreetMap contributors.

(a)

(b)

Figure 5.17: Vehicle tracks for the third task configuration. Vehicle must dock at the end of mission (a) Vehicle track with Greedy Planner (b) Vehicle track with Symbolic Planner. Backing image is ©OpenStreetMap contributors.

is required, but the simulation environment used to test the planning system precludes such large scale testing since it is both resource intensive and the missions are long in duration. Since the previous runs had shown that the vehicle responded reliably to guidance from the executive layer of the planning system, the reactive layer and simulator were removed and replaced with a program that read the motion goal position, and updated the pose to match. The simplified system neglects the time and distance required for heading changes but, since the distances are large compared to the turning circle of the vehicle, the recorded distances will be an accurate proxy of the actual vehicle performance.

This simplified system allowed a much larger set of trials to be run. The trials were executed with $survivors = 5, 10, 20$, and three conditions, one with the vehicle being randomly placed as in Section 5.8, and a second where the vehicle was placed randomly, but required to dock at the north end of the boating lake, and a third where the vehicle both launched and recovered from the north end of the boating lake. This combination of conditions demonstrates the creation of plans both without and without constraints.

The data was recorded and a summary can be seen in Table 5.8, while graphs of the distribution of normalised differences in path length can be seen in Figures 5.18, 5.19 and 5.20 for the case where the vehicle was launched randomly and not required to dock, Figure 5.21 where the vehicle was launched randomly and required to dock at the end of it's run, and Figures 5.22, 5.24 and 5.25 for launch and retrieval both occuring at the northern end of the boating lake. Figure 5.23 shows detail of the histogram of Figure 5.22 centered on the origin.

In the unconstrained planning case, the mean of the mission length for planned missions were shorter than the Greedy missions, by between 1 to 4% which are less than one standard

Table 5.8: Statistical comparison of mission lengths for Symbolic Planner vs Greedy Planner.

	No Constraint		Dock Constraint		Launch and Dock Constraint	
Survivors	mean	standard deviation	mean	standard deviation	mean	standard deviation
5	0.013	0.119			0.006	0.031
10	0.023	0.13			0.035	0.062
20	0.035	0.09	0.158	0.115	0.066	0.071

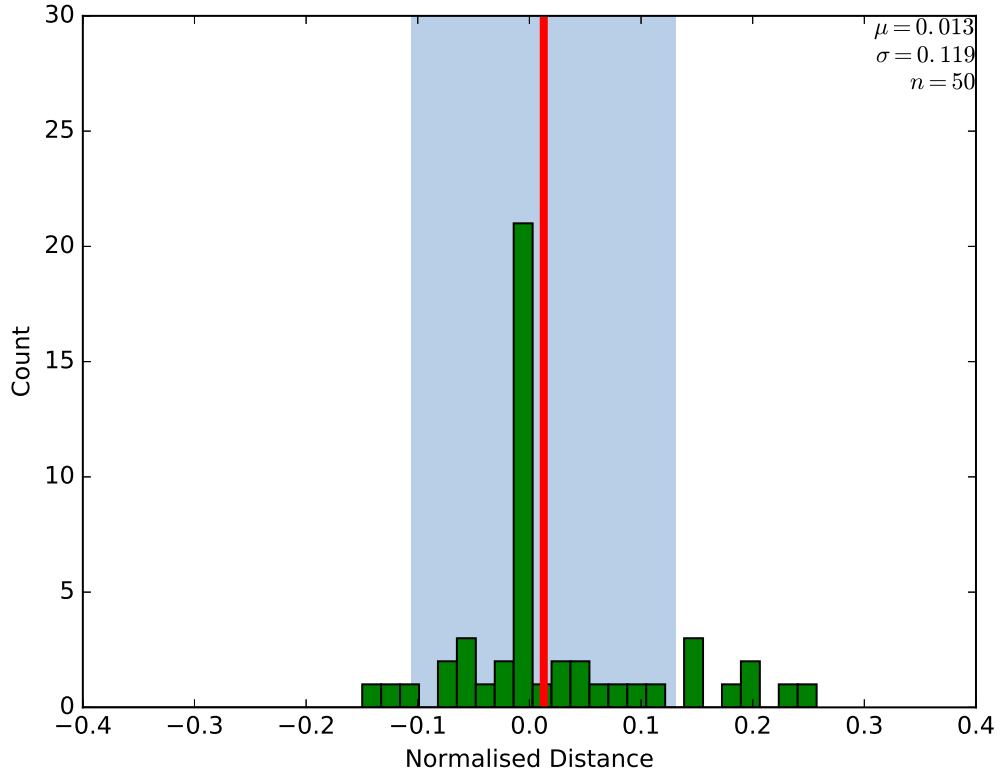


Figure 5.18: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 5$. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

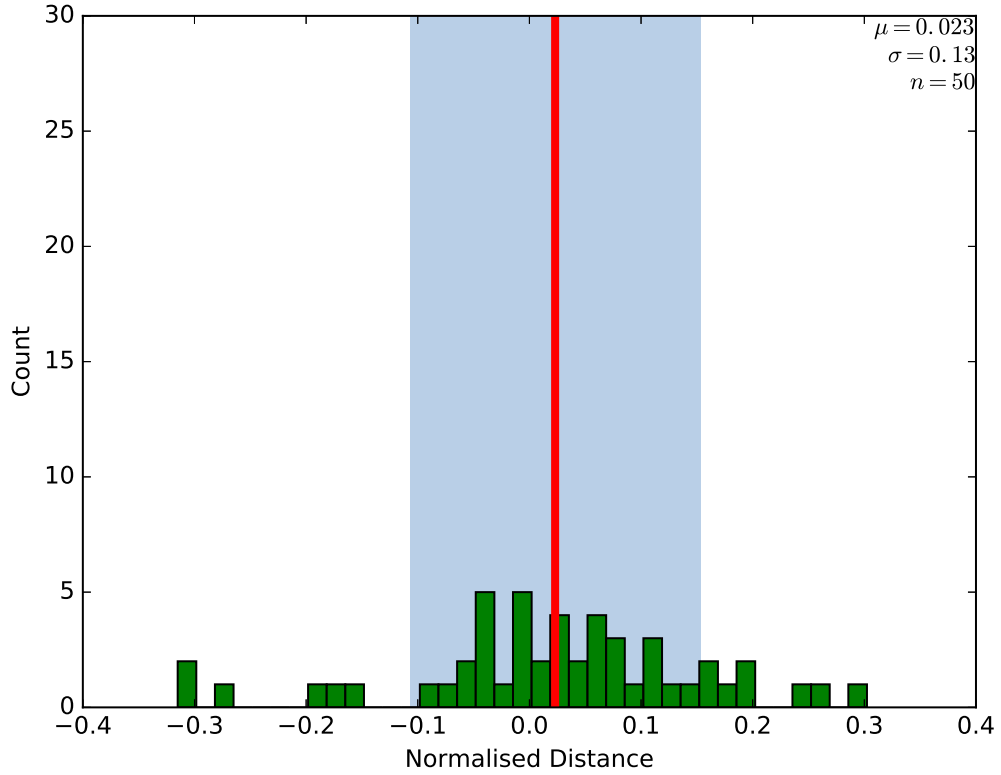


Figure 5.19: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 10$. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

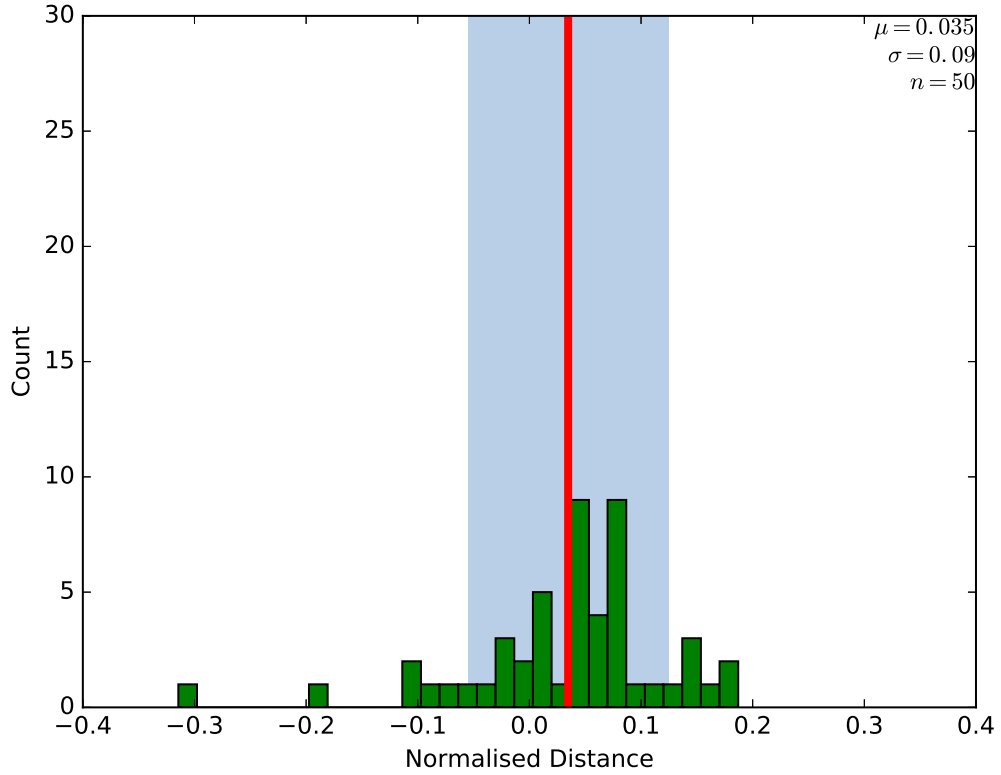


Figure 5.20: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 20$. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

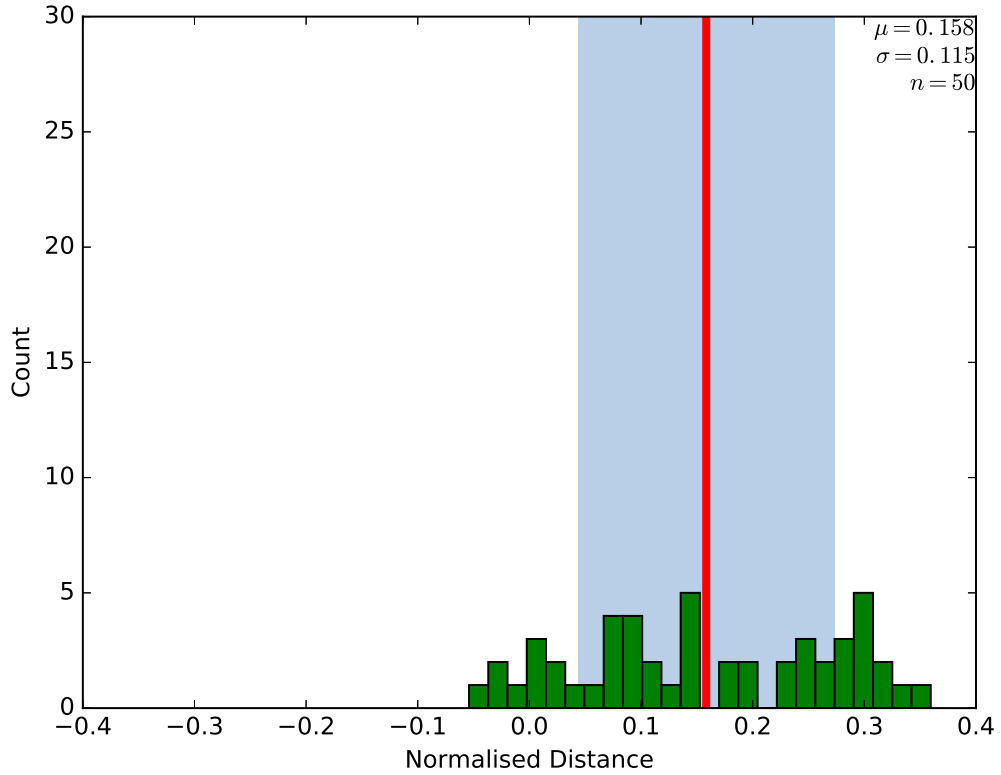


Figure 5.21: Histogram of normalised difference between mission length for the Greedy planner and symbolic planner with $survivors = 20$. Mission began at a random location, but was constrained to end at the north end of the boating lake. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

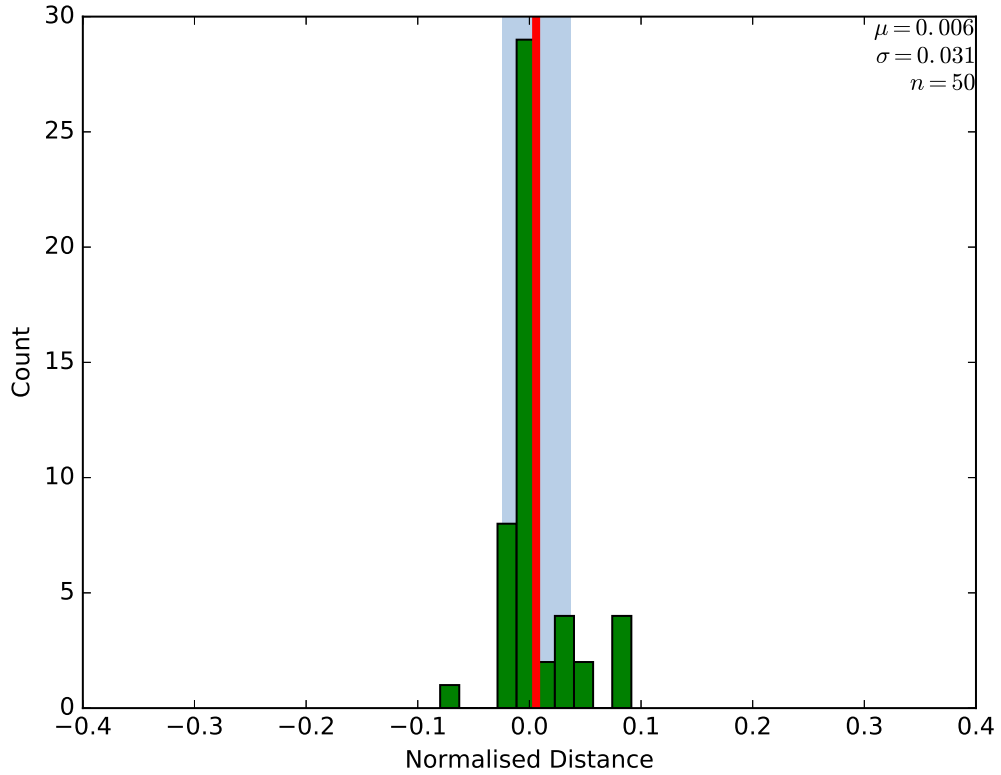


Figure 5.22: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 5$. Mission was constrained to begin and end at the north end of the boating lake. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

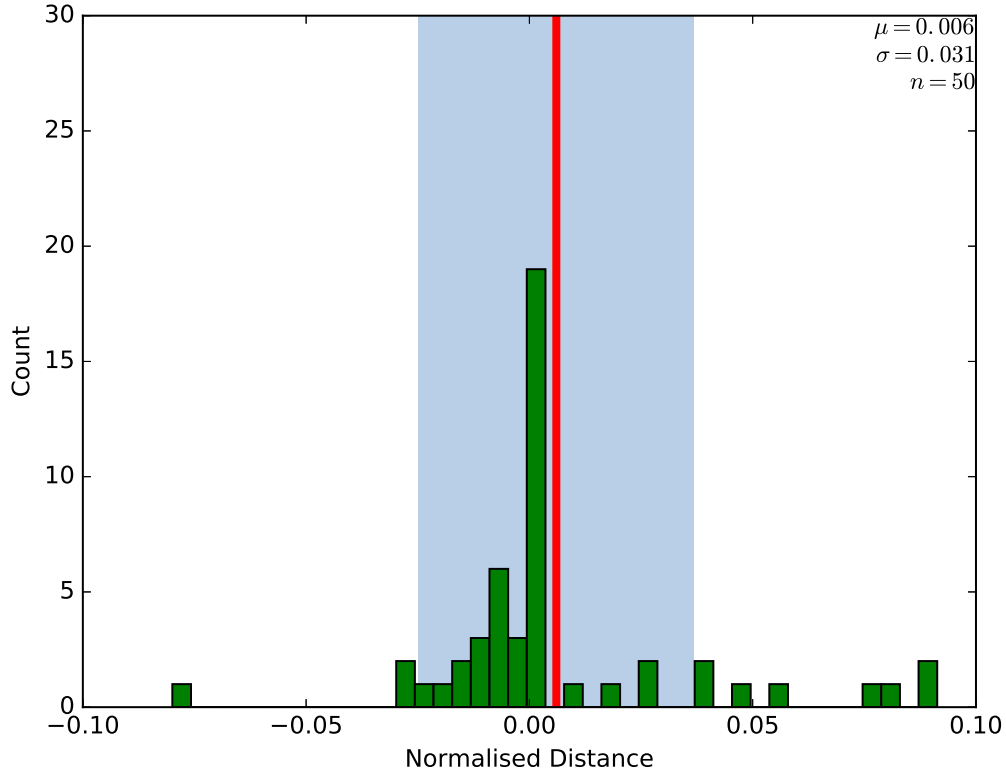


Figure 5.23: Detail of histogram shown in Figure 5.22. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

deviation.

For the case where the vehicle was launched and retrieved from the northern dock, the mean difference between the mission generation systems was larger than the unconstrained case, but once again was not larger than one standard deviation.

Examination of the histograms of the normalised plan lengths shows that in most cases, the distributions appear broadly unimodal, with outliers in the range of $\pm 30\%$ for the unconstrained case and $[-10\%, +30\%]$ for launch and retrieval in the northern boating lake. While the normalised mean differences between planning types were small, examination of the histograms showed differences with outliers. Figure 5.18, representing the unconstrained case with $survivors = 5$, shows outliers in the range of approximately $[-30\%, 30\%]$. The corresponding Figure 5.22 for the launch and dock constraint shows outliers in the range of $[-10\%, 10\%]$. Similar trends of reduced variability exist for the cases of $survivors = [10, 20]$.

The difference between these cases is that in the case where the vehicle was launched and docked in the boating lake, plans are constrained in their initial and final positions. Since the

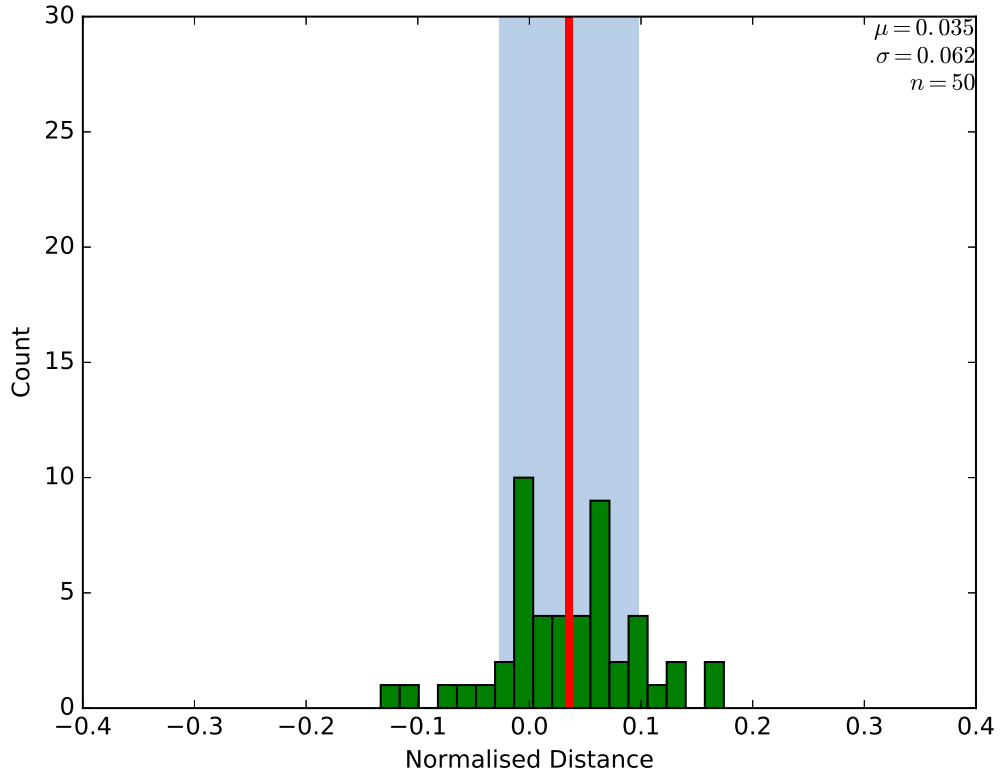


Figure 5.24: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 10$. Mission was constrained to begin and end at the north end of the boating lake. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

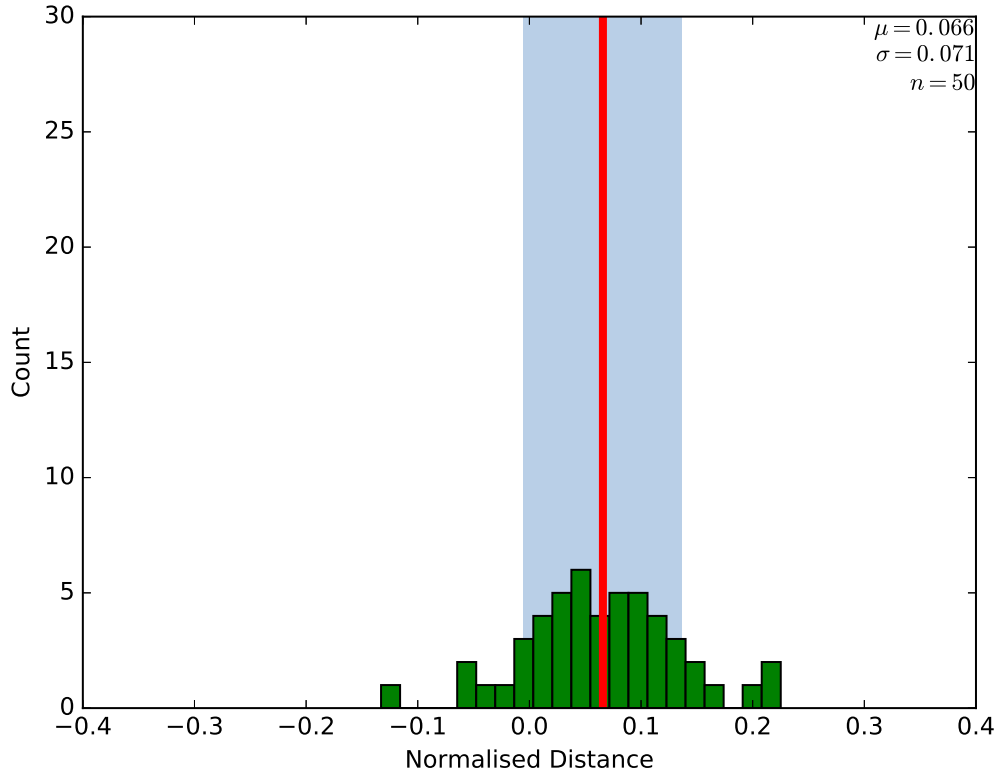


Figure 5.25: Histogram of normalised difference between mission length for the Greedy Planner and Symbolic Planner with $survivors = 20$. Mission was constrained to begin and end at the north end of the boating lake. Red line indicates the mean, while the shaded area is $\pm 1\sigma$. Positive values indicate Greedy Planner mission length exceeds Symbolic Planner mission length.

Symbolic Planner based system considers the global problem when generating each action, the ordering of actions considers the final position constraint of the vehicle.

The third case was when the initial position of the ASV was randomised, but the final position was constrained to finish at the northern boating dock. This configuration differs from the dock and launch constraint in that the path of the robot does not form a tour. As such, like in Figure 5.16, simply ordering goals to minimise tour length can result in a longer trip to return to the dock. In this configuration, the Symbolic Planner based system produced plans that were on average 15.8% shorter than the Greedy planner based system. This difference corresponded to $\approx 1.4\sigma$.

5.11 Conclusion

This chapter has covered a pair planning of planning systems developed for the task of planning field robotic systems including both ground and maritime vehicles. The Symbolic planning system was tested extensively and was found to be outperformed by a simple Greedy planner at short distances. By the application of plan refinement, and in situations where preconditions are significant, the plans produced by the combination of Symbolic Planning, and topological compression resulted in plans 15.8% shorter than using a locally optimising system.

These planning systems have demonstrated that the result of combining a domain independent planning system with a compressed spatial model to generate plans informed by spatial environment. With the extension of these systems with plan refinement, the results are also superior to locally optimising systems. A brief table outlining the properties of these systems can be seen in Table 5.9.

The flexibility of domain-independent planning with these superior path planning results allows significant scope for the implementation of future highly-complex planning systems using numerical predicates to track values such as fuel levels and vehicle carrying capacity allowing these values to be combined to generate optimal mission plans that incorporate vehicle constraints in addition to spatial information.

Table 5.9: Summary of Planner Properties

Type	Greedy	Symbolic	Symbolic With Refinement
Type	Select nearest action	Solve task using domain independent planning system	Solve task using domain independent planning system. Re-order actions using Greedy Planner
Supports preconditions	No	Yes	Yes
Considers spatial environment	No	Yes	Globally

Chapter 6

Conclusion

6.1 Summary

Planning in spatial environments is a non-trivial problem. The number of states to be explored prevents simply combining mission planning with spatial data. The method used by the ROSPLAN system for performing this task is to factorise the planning problem by considering spatial information only as a set of costs to travel between waypoints [Cashmore et al., 2015], but this can be limited. In particular, producing movement costs between n waypoints requires $(n(n-1))/2$ paths to be explored. Depending on the method, size, and length of the paths this may be a highly expensive operation. This is the reasoning behind the Lazy Tour coverage planning system - only generating detailed paths for those that are expected to be in the final coverage plan [Saha et al., 2006].

This thesis proposes an alternate method for performing high-level mission planning by combining mission and spatial data using belief compression for processing by a domain-independent symbolic planning system. As shown in Chapter 2 these topological methods of belief compression allow the data required to produce a plan to be reduced by assuming that if a homotopic space can be identified, then an optimal trajectory can be found within the space. As such, detailed path generation is only required for paths that will be utilised by the vehicle.

This assumption reduces the planning problem to the selection of a suitable set of homo-

topic spaces to traverse. Thus, topological information can be used by the high level planner to set broad spatial constraints for the low-level planning system. This allows the high level planning system to restrict the scope of its search, potentially improving its own efficiency. Chapter 3 showed that this assumption could be applied to both simulated and real-world map data while producing models of the environment that allowed effective choice of segments to traverse for short path plans. When performing the high level path planning task with spatial data derived from Apra Harbour, the Palágyi and Kuba skeletonisation algorithm and watershed reconstruction were found to produce high-level paths that contained the optimal trajectory between $60\% - 100\%$ of the time with the mean length of non-optimal paths being 1.5% longer than optimal.

A number of planning systems were examined in Chapter 4, including multiple search and heuristics to identify their applicability to the task of planning in a spatial environment, and their scalability. In particular, the A* search algorithm with the Blind and iPDB heuristics was found to be efficient at the planning task with multiple preconditions while not demonstrating any appreciable slowdown in the tested spatial environments with increasing task complexity.

Chapter 5 combined belief compression and a domain-independent symbolic planner to generate mission plans for a simulated maritime vehicle. This method was shown to be superior to a greedy planning system in a maritime rescue task where preconditions exist due to it's ability to optimise the global plan. In the case where an ASV was randomly placed, required to collect survivors and then dock, the Symbolic With Refinement planner outperformed a greedy planning system by an average of 15%.

6.1.1 Additional Deliverables

As part of the development of this thesis, a number of other useful systems were developed including systems for performing various skeletonisation and geometric tasks, libraries for handling PDDL code, a path generation system as demonstrated in Appendix A, and an alternate method for control allocation using model predictive control as described in Appendix E. These systems have been used for measuring the bathymetry of a lake environment, demonstrating their suitability for maritime survey tasks. Derivatives of these

systems and other concepts developed during the production of this thesis have found use in the development of the TopCat ASV, and projected future autonomous vehicles.

In addition to the robot planning architecture, a simulation system was also developed based on the Gazebo robotics simulator as outlined in Appendix F. This simulator was validated by repeating and comparing the results of field trials. This system was used for the development and testing of the high-level planning systems.

6.2 Future Work

6.2.1 On-line Planning

One of the important points to come out of the creation of this thesis is that similar to the travelling salesperson problem (TSP), mission plans can benefit from a process of plan refinement. This requires the maintenance of state that would be discarded in a standard off-line planning model. Without the ability to evaluate whether the addition or reordering of planned actions will violate plan constraints, refinement can only be performed at the most trivial of levels.

6.2.2 State Update

An improved model of the planning system would allow publication of states during a planning run. As observed in Chapter 5, it is possible to encode standard textual and numeric grounded predicates into Robotics Operating System (ROS) message types. This means that it should be possible to extend ROS' tool based model to the level of the mission planning system. By modifying the planning system to be a set of independent tools that interoperate, introspection of the system can be improved by leveraging the existing tools for state recording and review.

6.2.3 D* Search

A limitation of informed search algorithms such as A* is that they require the entirety of planning graph to remain in memory as they cannot prune their search space [Russell and Norvig, 2010]. However this can be turned to an advantage by the use of search algorithms such as D* that allow updates to the planners state as the robots environment changes [Stentz, 1994]. Such a system would not require the entire search to be repeated for each change to the environment.

6.2.4 Trusted Autonomy

With the current interest in trusted autonomy, improving introspection can improve trust by both communicating the intentions of the planning system and the reasoning behind its actions. Such visibility and accountability of the planning system should allow human operators to build trust by their own review of the planning systems proposed actions.

6.2.5 International Regulations for Preventing Collisions at Sea (COLREGS)

A further area of interest is the design of suitable belief compression systems. The topological based methods tested in Chapter 3 were initially envisioned for an Autonomous Underwater Vehicle in a volumetric environment. There are similar requirements for planning tasks for maritime surface operations. A system based on the use of Delaunay Triangulation was developed for Flinders University's entry to the Maritime RobotX Competition [Webb et al., 2016]. This system was further expanded into a proposed system for planning in constrained environments such as marinas [Wheare et al., 2018]. Further development of this system should allow it to support International Regulations for Preventing Collisions at Sea (COLREGS) navigation tasks.

6.3 Environmental monitoring

The TopCat Autonomous Surface Vessel (ASV) was designed to be capable of performing environmental monitoring tasks. TopCat's shallow draft allows it to operate in areas that are closer inshore than what a true ocean-going survey vessel would require. This potentially allows precision monitoring of inshore waters. This was demonstrated in Appendix E which included a demonstration of bathymetry measurement using the TopCat vehicle.

This survey was limited in that an ASV is limited to observations that can be performed from the surface. Roelfsema et al. expressed interest in the data gathering abilities of an Autonomous Underwater Vehicle (AUV) able to dive beneath the limit achievable by a human swimmer [Roelfsema et al., 2015]. Compared to an underwater vehicle, an ASV has greater payload and access to precision navigation systems, but lacks the ability to approach the underwater area of interest. A possible solution is the development of a hybrid vehicle consisting of an ASV platform with a tethered Remotely Operated Vehicle (ROV). Development of such hybrid systems are being performed by the EU Trident project [Sanz et al., 2013], and the University of Florida's Anglerfish [Gray and Schwartz, 2016]. Such hybrid vehicles have potential to perform many of the tasks that are currently done by shallow-water AUVs. Interest in this area does appear to be increasing, with the upcoming 2018 Maritime RobotX challenge also proposing a task to recover a set of underwater rings. This will likely require the use of a tethered ROV [Maritime RobotX Challenge, 2018].

Future development of the ASVs planning system would allow the automatic generation of observation plans, with support for ship tracking and COLREGS path planning potentially allowing survey tasks to be completed in complex and busy environments such as harbours.

6.3.1 Beneficiaries

The systems designed for this thesis would benefit applications requiring efficient plan generation where the problems of spatial and mission planning are not easily separable. This could provide a potential improvement in performance for applications in the following areas;

6.3.1.1 Research

As shown in Appendix A, the TopCat platform can perform environmental monitoring tasks. By the use of a spatially-aware planning system similar to that covered in this thesis, future systems could perform such tasks in environments that are complex such as harbours and near structures without requiring extensive pre-planning by a human operator. This would allow more missions to be performed for a given number of staff potentially improving productivity.

6.3.1.2 Academic

The use of topology to construct graphs representing a compressed space has potential applications beyond the production of graphs for plan generation. In much the same way that heuristics are relaxed planning problems, a topological graph can be considered a relaxation of a path plan. As such, a topological skeleton could have applications to guiding solutions in other spatial planning problems.

6.3.1.3 Industry

Similar to the use as a research platform, improvements in planning could lead to more effective performance of monitoring tasks around complex structures such as tailing dams, aquaculture nets, and oil rigs. These structures require inspection for damage and corrosion. Particularly in areas with complex spatial environments, inspection plans could benefit from the use of topological information.

6.4 Conclusion

In conclusion, this thesis has shown that deliberative domain-independent planners can be used with spatial data if it is first compressed. This system is capable of outperforming a locally optimising planning system while executing rapidly enough to allow rapid re-planning

due to environmental changes, and supporting the future implementation of more complex robotic planning domains.

Appendix A

Experimental Validation of TopCat Planning Architecture

To validate that the path planning system developed for the TopCat Autonomous Surface Vessel (ASV), a task based on both environmental monitoring was developed. This would demonstrate both the ability of the planning system to generate trajectories, and for TopCat to reliably and repeatably follow them.

A.1 Environmental Monitoring

As discussed in the introduction, one of the areas of interest for field robotics is the carriage of sensors for environmental monitoring. Amongst the areas of research being undertaken by the School of The Environment at Flinders University, is work on the monitoring of coastal geomorphology and evolution. Earlier studies such as the one undertaken by da Silva et al. had used fishing vehicles to carry depth sounding equipment [da Silva et al., 2012]. TopCat, with its shallow draft and ability to beach without damage, could potentially make an excellent sensor platform for capture of bathymetric data in shallow inshore waters. This comes at the cost of having limited directional control due to its large windage and shallow draft. To demonstrate that the control system was capable of reliably executing the repeatable transects required for the generation of time series data, an experiment was performed

at West Lakes to demonstrate control of the platform and integration with the bathymetry equipment.

A.2 Aims

To demonstrate the ability of the TopCat ASV as an environmental mapping tool, the following aims were developed;

- Demonstrate guidance of the TopCat vehicle to waypoints using the planning system
- Demonstrate repeatable traversal of transects.
- Capture data from the CeeScope bathymetry device for building bathymetry maps of a maritime environment.

A.3 Method

The area selected for this experiment was the north end of West Lakes boating lake - an artificial Saltwater environment developed as part of the West Lakes housing development in the 1970s. Powered vehicles are prohibited to operate on West Lakes, but a permit was negotiated for the operation of TopCat for research purposes. Imagery of West Lakes can be seen in Figure A.1.

Depth information was captured using a CeeScope echo sounder. The control and data logger was mounted on the sensor tray, while the SONAR transducer was bolted to TopCats deployer mechanism. The CeeScope device integrates position estimation, depth sounding and data logging into a single package. To enable it to log precision depth information without the provision of external tide data, a Real-Time Kinematic (RTK) Global Position Satellite (GPS) based state estimation system was configured using an R10 [Trimble, 2014] and a BX982 GPS [Trimble, 2016]. State estimates from the system were communicated to the sensor using the NMEA protocol. After conclusion of the experiment, the depth data and GPS fixes were downloaded from the CeeScope sensor.

Twelve waypoints were generated in the Boating Lake. The vehicle was given six transects to traverse, each between a pair of waypoints as shown in Figure A.2. Transects were executed by use of an actionlib [Open Source Robotics Foundation, 2014a] goal sent to the wamv_motion_planner node. Each of these goals used two named waypoints as the start and finish of a transect.

The planner automatically created a goal dispatching the vehicle to the start waypoint. The motion planning node then dispatched these goals to the contol allocation node. The program was executed five times and the resulting vehicle track plotted. A map showing the vehicle track can be seen in Figure A.3. All runs were executed under autonomous control. The first run was aborted due to the presence of a kayak resulting in the deviation between waypoints search2 and search3. The remaining runs were completed without incident.

More details on the performance of the vehicle guidance and control system can be found in Appendix E.

A.4 Results

The Ceescope depth data was post-processed to produce the depth data shown in Figure A.4. These data are consistent between runs, but not corrected for antenna offset error.

Inspection of the vehicle track shows that the vehicle performed the majority of its inspection task correctly, however the approach to waypoints search5, and search9 shows the vehicle converging onto the search transect only after passing the waypoint. This is due to the method of implementation of the control system, as the vehicle uses a line-of-sight mechanism for converging onto the line of transect, there is no constraint on this occurring before the start waypoint. Waypoints search3, search7, and search11 do not exhibit this behaviour since the vehicle ran long at their preceding waypoints.

Using these data, a contour bathymetry map of the boating lake was constructed using QGIS [QGIS Development Team., 2018] contour plugin [Roubeyrie and Crook, 2018]. The result of this is the map shown in Figure A.5. While no ground truth was available to confirm the measurements, the readings taken by the system appear consistent.

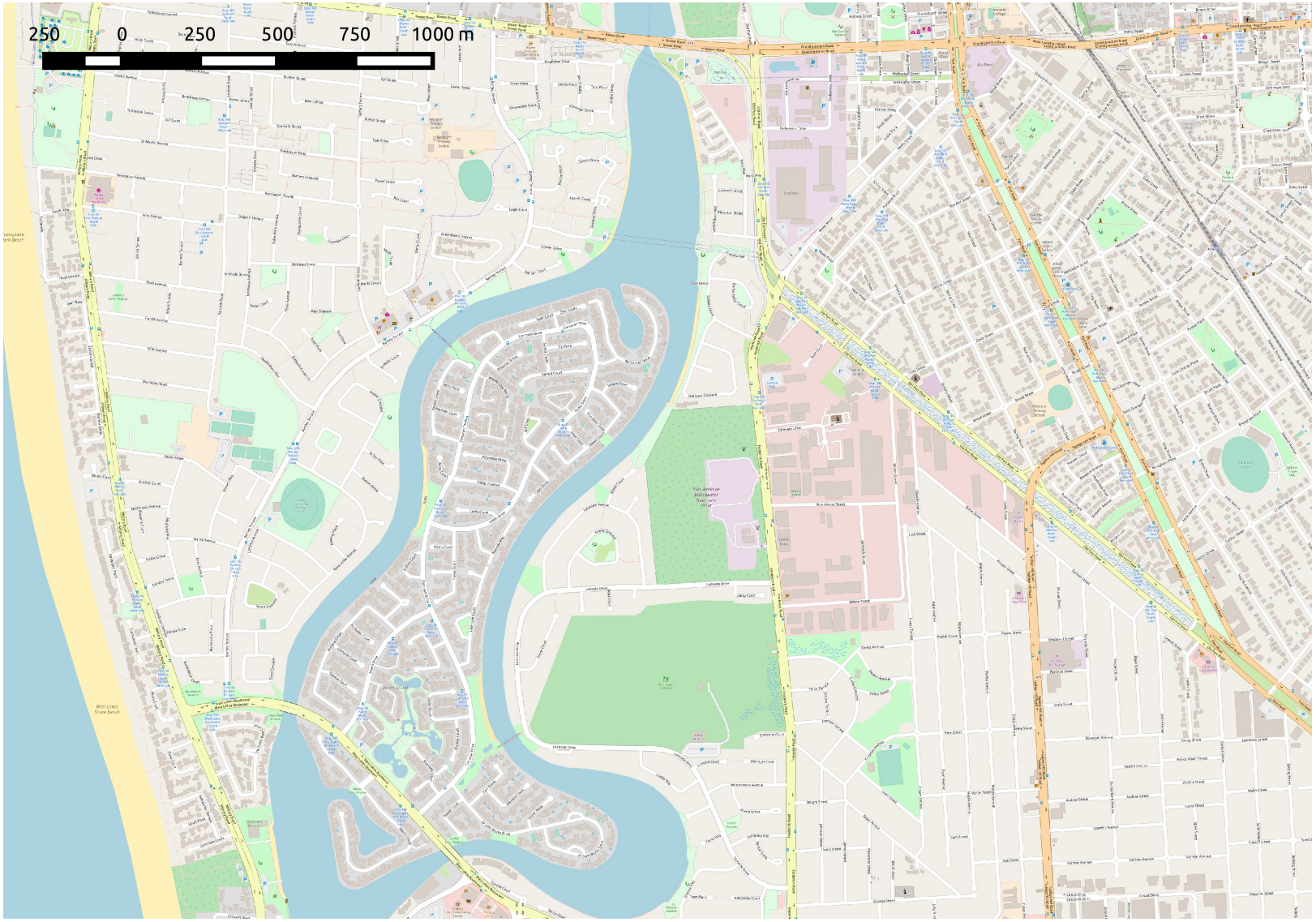


Figure A.1: Map of area surrounding West Lakes. Map ©OpenStreetMap contributors.

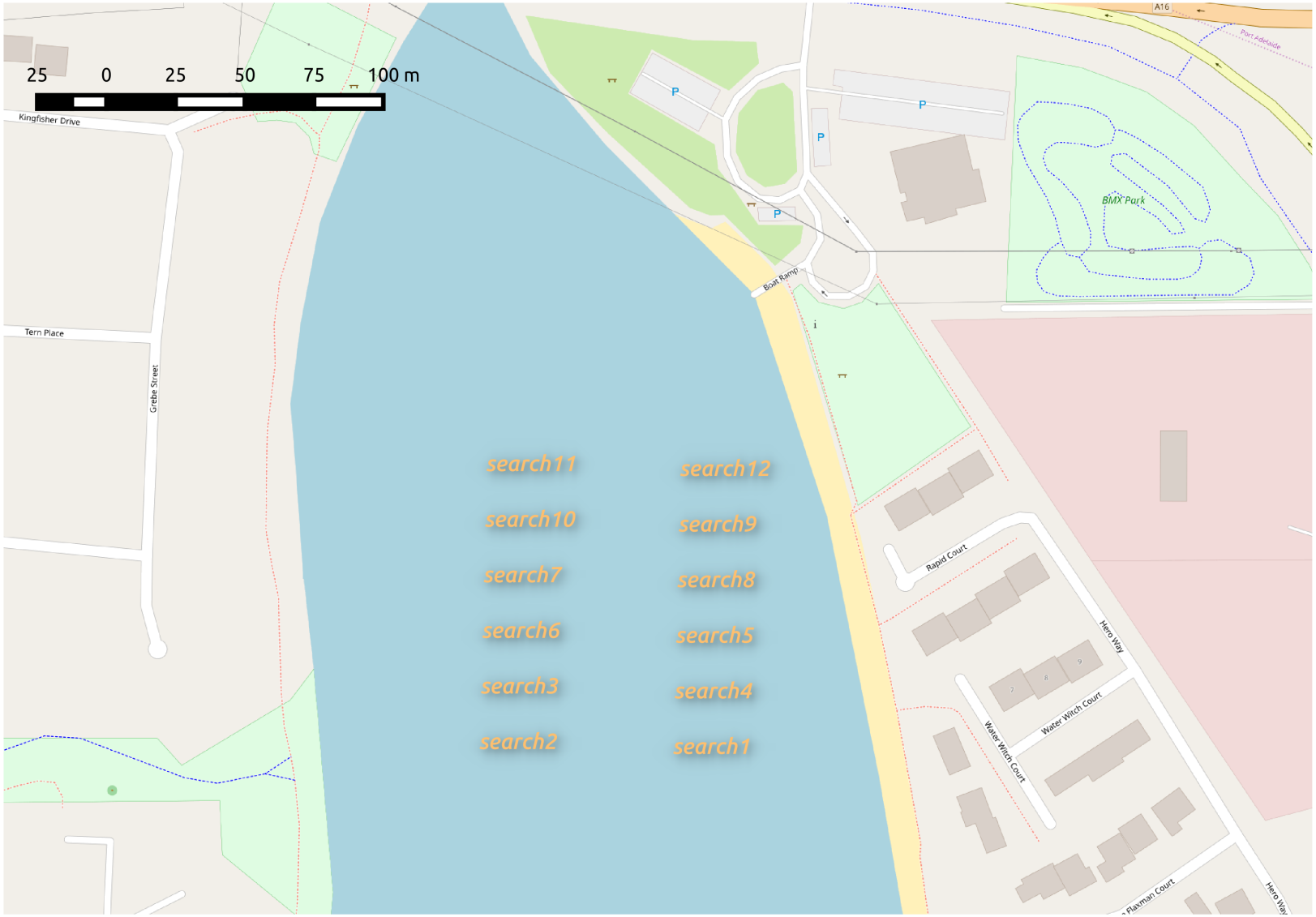


Figure A.2: Map of West Lakes with search waypoints marked. Backing map ©OpenStreetMap contributors.

Figure A.3: Map of West Lakes with search waypoints and vehicle track marked. Backing map ©OpenStreetMap contributors.

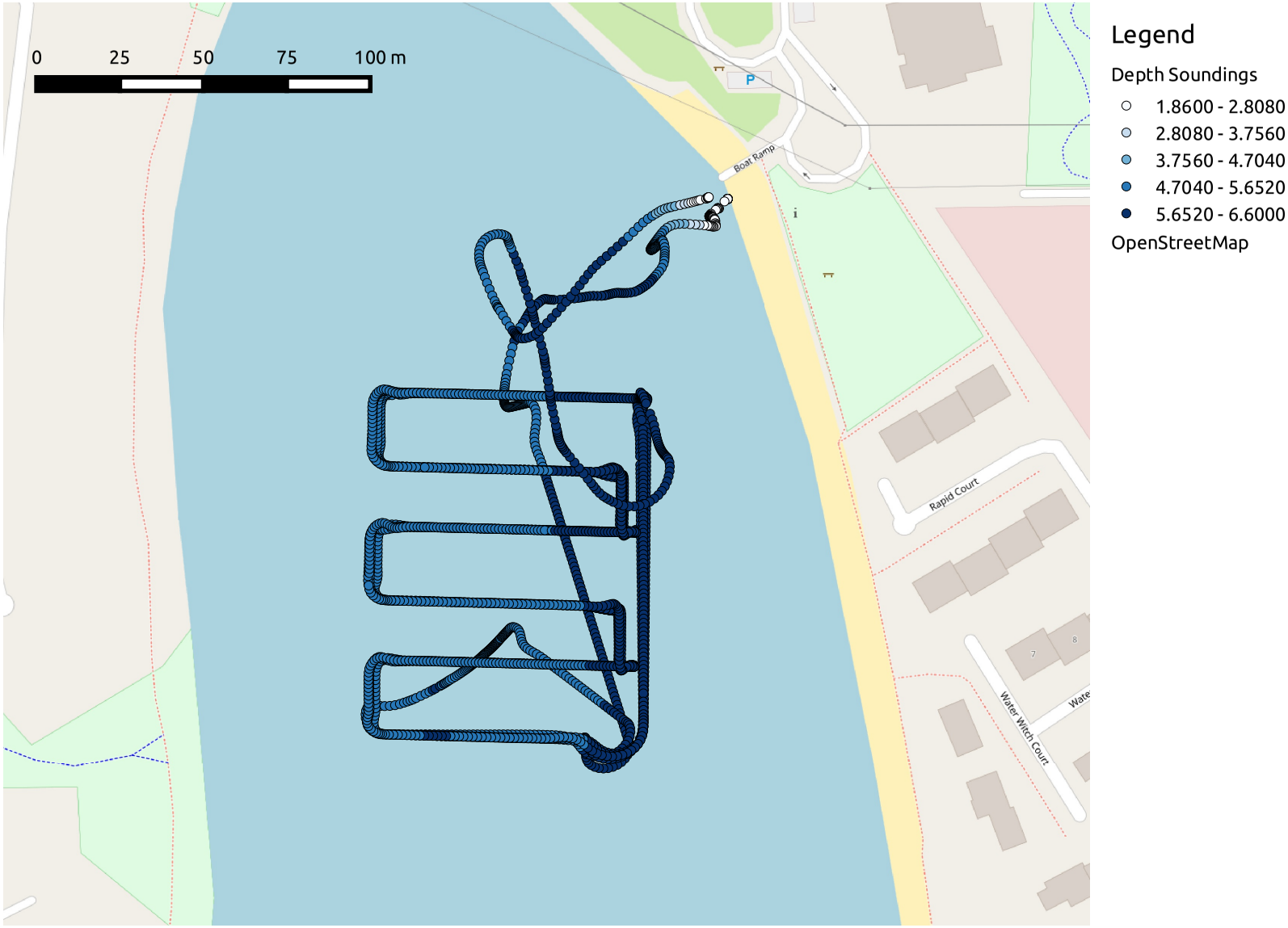


Figure A.4: Depth data captured by CeeScope sensor during West Lakes depth sounding mission. All depths in metres. Backing map ©OpenStreetMap contributors.

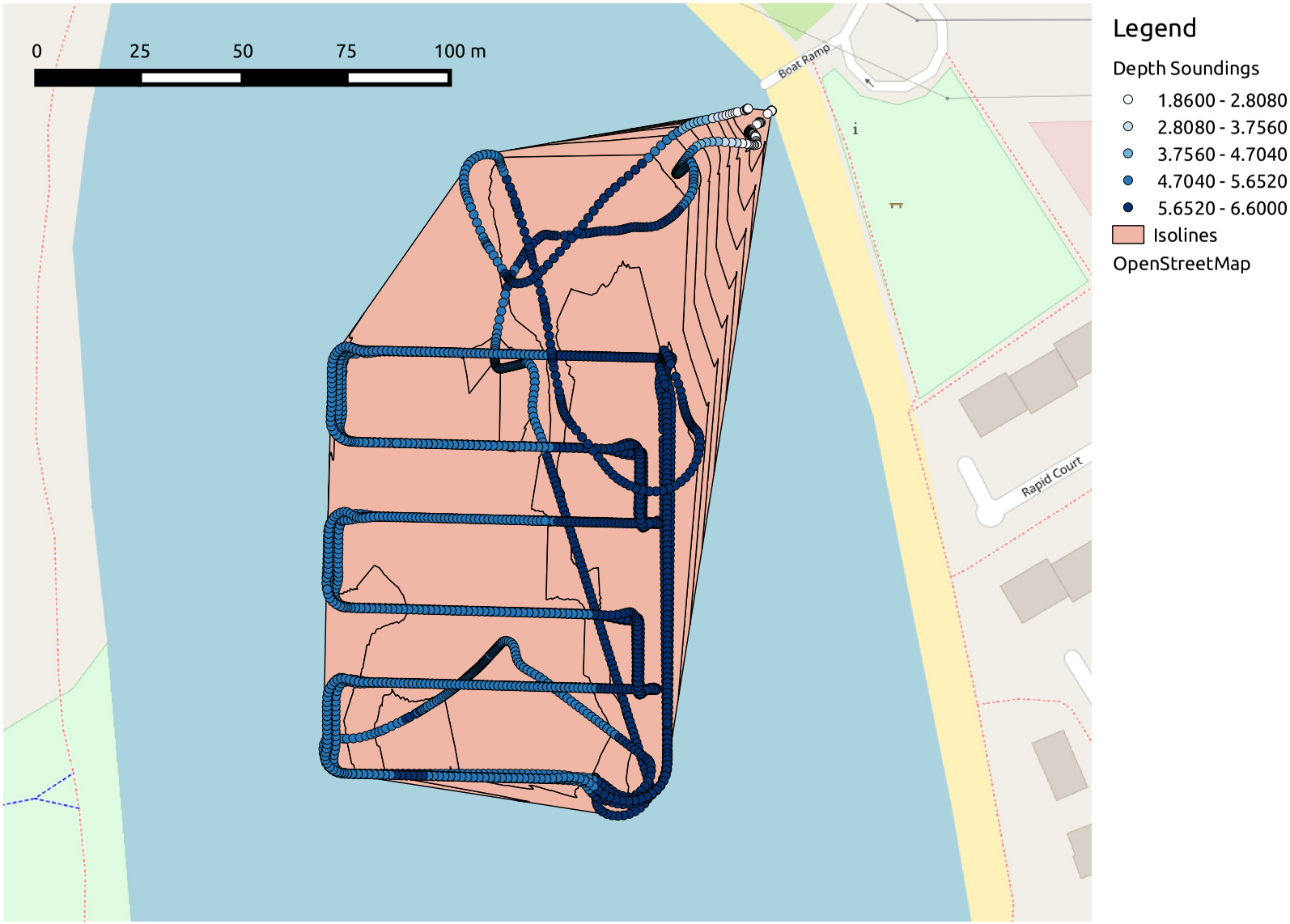


Figure A.5: Isolines constructed from the depth data captured during West Lakes depth sounding mission. All depths in metres. Backing map ©OpenStreetMap contributors.

A.5 Conclusion

The vehicle was capable of repeatable traversal of transects with a lateral error in the order of half a metre. This capability was used for the capture of depth data from a prepared plan which was then used to construct a bathmetry map of the test area. Construction of this map shows that the TopCat ASV can perform environmental monitoring tasks. Further improvements to the handling of transects and motion primitives would allow the system even greater precision when performing such tasks.

Appendix B

An Introduction to the Operation of Symbolic Planners

Since the use of the Problem Domain Description Language (PDDL) and symbolic planning is important to this thesis, a short introduction describing its operation has been added.

An early problem demonstrating the advantages of searching for plans is the monkey and banana problem proposed by McCarthy [McCarthy, 1963]. This problem uses three objects, a monkey, a box and a bunch of bananas to illustrate the requirement for building multi-step planners.

In this problem, the monkey is an agent that is capable of executing the plans calculated by the planning system. A graphical representation of the monkey can be seen in Figure B.1 along with the Problem Domain Description Language (PDDL) code required to initialise a planning domain. PDDL provides a planner independent method for defining such planning tasks.

In a symbolic planning system the state of a planning operation is defined as a set containing the initial state, set of available actions, current plan, and the goal state. Bylander represents these as the tuple shown in Equation B.1 [Bylander, 1994]:

$$[P, O, I, G] \tag{B.1}$$

Where;

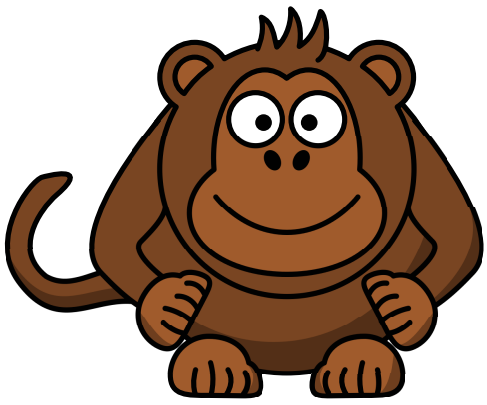
- P - is the set of ground atoms describing the current world state.
- O - is the set of *operators* that can transform the atoms.
- I - is the initial state.
- G - is the goal state.

In operation, the planner searches through actions adding those that transform the goal into the initial state to the plan. Figure B.1 shows the PDDL code defining the monkey's existence as an object has been added. No initial actions or grounded predicates are defined, so the initial state is empty as shown in Equation B.2.

$$[P, O, I, G] = [\emptyset, \emptyset, \emptyset, \emptyset] \tag{B.2}$$

An action consists of a set of preconditions, a list of predicates to be added to the state and another list of predicates to be deleted the *add list* and *delete list*.

When actions are added, they define alterations to the predicates of the system. If another entity is added that represents a bunch of bananas the monkey wishes to possess, this can be represented by the addition of a predicate called *has* representing possession and an



(define (domain monkeybananas)
(:requirements :strips :typing :fluents)
(:types entity)
(:objects monkey - entity))

Figure B.1: A monkey representing an agent in a planning system. Clipart from openclipart.org

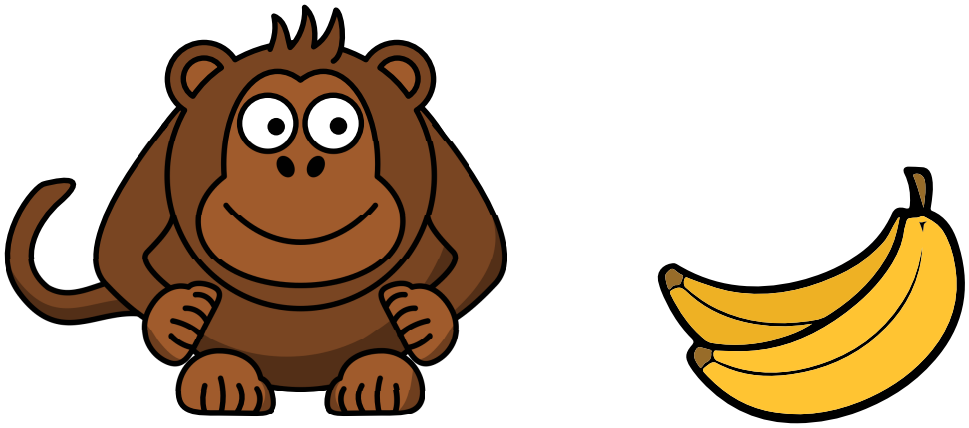
action *get* that allows the predicate to be set. Predicates start off as potential relationships between *free variables* representing types of objects. When a type in a predicate is replaced by a specific object, it is said to be *grounded*. The goal defines the grounded predicates that the system is directed to achieve. In this system, the state is now as shown in Equation B.3;

$$[P, O, I, G] = [\emptyset, \text{(get (:effect has o1 o2))}, \emptyset, \text{(has monkey banana)}] \tag{B.3}$$

which can then be used to generate the plan as shown in Equation B.4;

$$\text{plan} = \text{(get monkey banana)} \tag{B.4}$$

In addition to transforming the predicates of a system, actions may also have preconditions that are required to be true before they can be performed. If the bananas are suspended from a branch, the monkey may require a method to reach them - the banana is initialised



(:predicates (has o1 o2 - entity))
(:action get (:effect has o1 o2))
(:objects monkey banana - entity)
(:goal (has monkey banana))

Figure B.2: The monkey can reach its goal by executing the get action. Clipart from openclipart.org

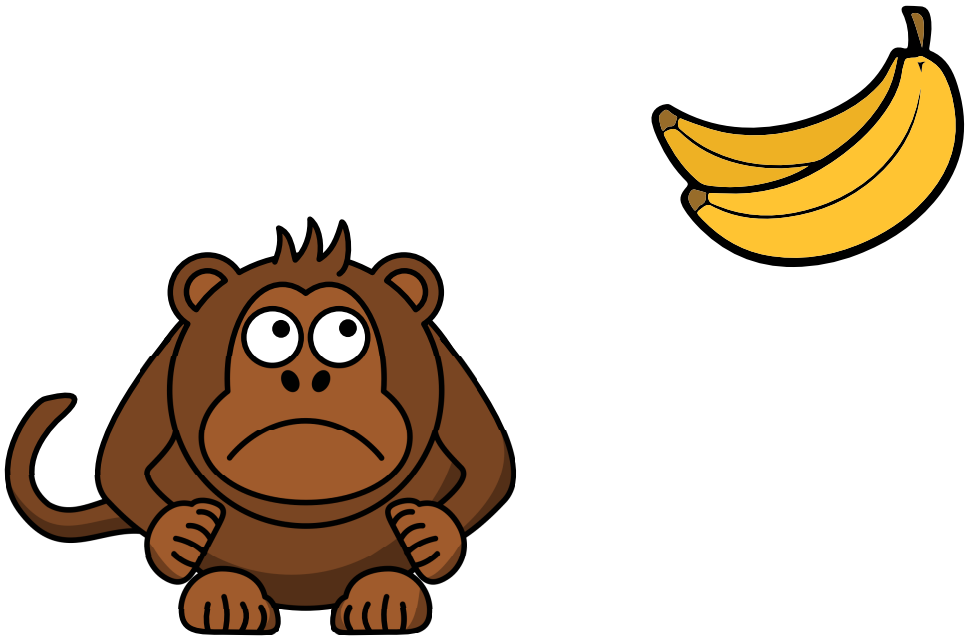
with the predicate *high* which is also added to the preconditions of the get action.

$$
\begin{aligned}
[P, O, I, G] = &[[\text{(high banana)}], \\
&[\text{(get (and (high o1)(high o2))(:effect has o1 o2))}], \\
&[\text{(high banana)}], \\
&[\text{(has monkey banana)}]]
\end{aligned}
\tag{B.5}
$$

Since the monkey has no method of achieving the grounded predicate (high monkey), this configuration of the problem is insoluble as shown in Equation B.6.

$$\text{plan} \equiv \emptyset \tag{B.6}$$

For the monkey to reach the bananas, a new action needs to be added. *Climb* allows the monkey to climb an object that has the predicate *climbable*, gaining the predicate *high* and thus satisfying the preconditions of the get action. A new object, ladder, is added and



(:predicates (high o - entity))
(:action get (:precondition (and (high o1)(high o2)) (:effect has o1 o2))
(:init (high banana))
(:goal (has monkey banana))

Figure B.3: The prerequisites of the get action prevent the monkey from obtaining the bunch of bananas.

initialised with the climbable predicate as shown in Figure B.4, this gives the system shown in Equation B.7.

$$\begin{aligned}[P, O, I, G] =&[[\text{(high banana), (climbable ladder)}],\\ &[\text{(get (and (high o1)(high o2))(:effect has o1 o2))},\\ &\text{(climb (:precondition (climbable o1)) (:effect high o2))}],\\ &[\text{(high banana), (climbable ladder)}],\\ &[\text{(has monkey banana)}]]\end{aligned} \tag{B.7}$$

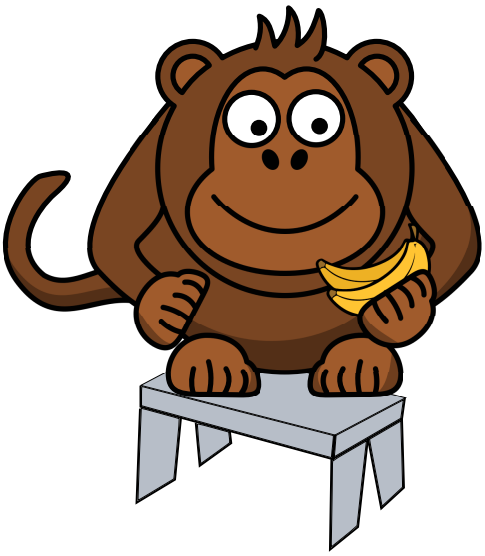
In this configuration, the monkey cannot immediately reach its goal - it must first enter an intermediary state as shown in Equation B.8.

$$\begin{aligned}\text{plan} &= [\text{(climb ladder)}]\\ P &= [\text{(high banana), (climbable ladder), (high monkey)}]\end{aligned} \tag{B.8}$$

Once in this intermediate state the preconditions of the get action are satisfied and the plan can be completed as shown in Equation B.9.

$$\text{plan} = [\text{(climb ladder), (get bananas)}] \tag{B.9}$$

This domain assumes that the objects are all at a single location. If the entities are at seperate locations as shown in Figure B.5, a new set of actions is required. Support for multiple locations can be provided by adding a new predicate *at* that designates an entity as being within an instance of another new type, a *node*. The monkey can move between locations which are *accessible* or move the ladder and itself by using *push*, which requires the ladder and monkey to be at the same location. This is performed by adding the following predicates and actions to the system as shown in Equation B.10



(:predicates (climbable o - entity))
(:action climb (:precondition (climbable o1)) (:effect high o2))
(:objects monkey banana ladder- entity)
(:init (climbable ladder))

Figure B.4: The addition of a ladder object and the climb action allows the monkey to achieve its goal.

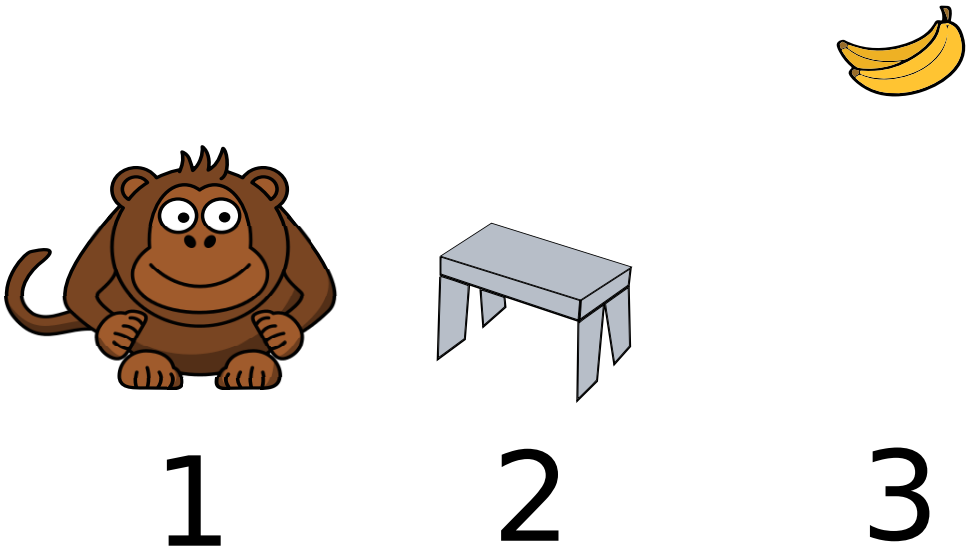


Figure B.5: Each entity now has a location

$$
\begin{aligned}
[P, O] = &[[\text{(at location1 banana), (at location2 monkey), ,} \\
&\text{(accessible location1 location2),(accessible location2 location3)}] \\
&[\text{(move (and (at o2 o1)(accessible o2 o3)),} \\
&\text{(:effect (and (at o3 o1)(not at o2 o1)))}]]
\end{aligned}
\tag{B.10}
$$

Notably this action is the first to use the delete list. When the monkey leaves a location, this is performed by deleting the predicate showing a relationship between the entity and that location. Since, by the closed world assumption, any relationship that is not listed as being true, is assumed to be false the monkey will no longer be associated with its former location.

If the ladder is not at the required location a similar action can move it to the bananas as shown in Equation B.11;

$$
\begin{aligned}
[P, O] = &[[\text{(at location3 ladder)}] \\
&[\text{(push (and (at o3 o1)(at o3 o2)(accessible o3 o4)),} \\
&\text{(:effect (and (at o4 o1)(not at o3 o1)(at o4 o2)(not at o3 o2)))}]]
\end{aligned}
\tag{B.11}
$$

The spatial relationships of objects can thus be encoded into a planning domain and a plan of action derived that performs a sequence of actions in this environment. Were the problem presented as a simple motion-planning task, no solution would be possible since the monkey must transform its environment before a trajectory can be generated that takes it to its goal.

In this case, if each action is assigned a cost $f(a)$, then the solution space can be searched to find the plan A in the set of all possible plans $A_{1..n}$ that satisfies the relationship in Equation B.12.

$$
\min(\Sigma f(a), a \in A_k), A_k \in A_{1..n} \tag{B.12}
$$

Since this plan is the lowest cost set of actions, it can be said to be *optimal*.

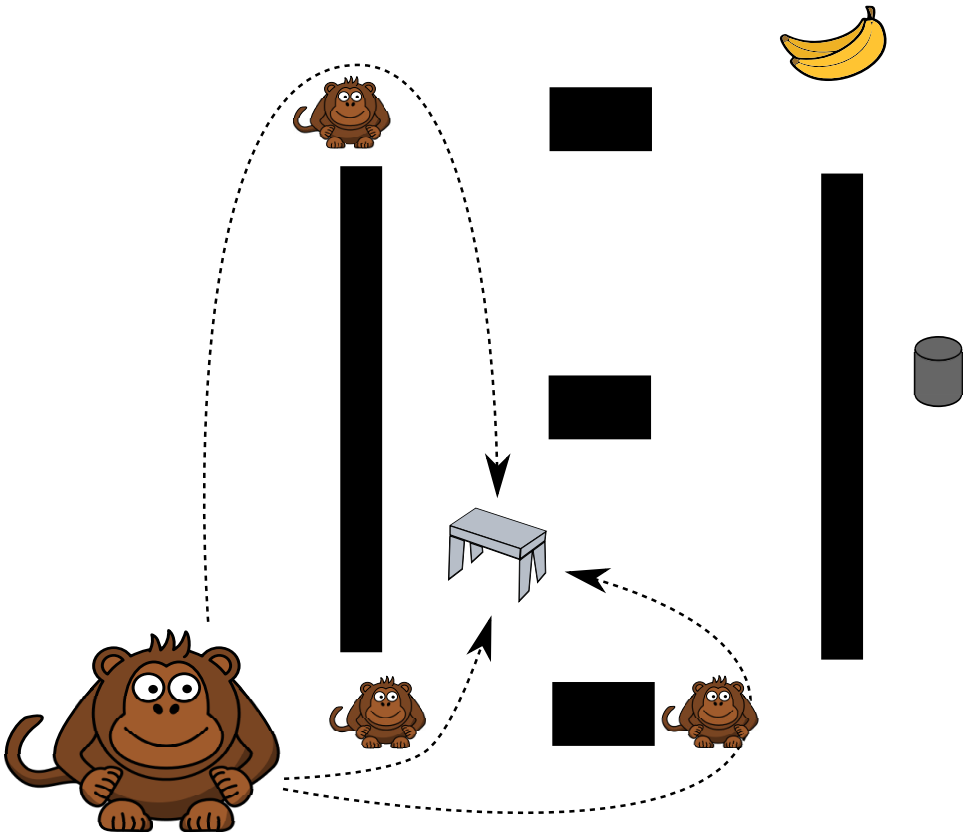


Figure B.6: In a more complex environment, an agent may have multiple paths to a goal

B.1 Conclusion

This chapter has provided a brief overview of the operation of the method of operation of a symbolic planning system. Although the problem is quite different to that of planning for field robotics, the spatial relation predicates and get action are identical to those that were used in the body of this thesis.

Appendix C

Problem Domain Description Language Keywords and Handling

This thesis involves the use of several domain-independent symbolic planning systems with software to interact with a field robots systems. This process involves transforming information from the different internal states used by these systems. This chapter will provide an overview of the languages, code and message types used to perform this information transfer.

The systems developed for this thesis use the Problem Domain Description Language (PDDL) to specify the state of the robots environment and it's possible actions to the tested symbolic planning systems. This appendix will provide a brief list of Problem Domain Description Language (PDDL) keywords and their definitions. Later sections will cover the particular PDDL code used to define the domains used, and finally the custom Robotics Operating System (ROS) messages used to communicate between the parts of the planning system.

C.1 Problem Domain Description Language (PDDL) Statements

The layout of a PDDL file resembles the *Lisp* language in having a recursive structure delimited using brackets. As such, the contents of a file can be represented as a tree structure. A PDDL planner requires two files to generate a plan, a *domain* file that sets the types, predicates and actions that are used by the domain, and a *task* file that instantiates the grounded predicates describing the initial and goal states.

For planning competitions, a single domain file will be used with a number of task files of increasing complexity. For example, the *visit-all* domain from the 2011 IPC competition [Lipovetzky, 2010] has twenty task files in it's sequential optimal repository [Lipovetzky, 2011].

C.1.1 Common

Both the domain and task file have a set of common text tags representing Boolean relationships

- *and* - boolean and operation. This operator will evaluate to true if all of it's subtags are true.
- *or* - boolean or operation. This operator will evaluate to true if any of it's subtags are true.
- *not* - boolean not operation. This operator will evaluate to true if it's subtag is false.

C.1.2 Domain File Only

- *requirements* - lists the capabilities of the solver that are required for the domain to be correctly processed.
- *types* - the object types that will appear in the domain

- *predicates* - a list of the relationships between keywords and objects that can be used in the symbolic planning system.
- *functions* - used in domains that support numerical fluents to specify the numerical relationships that are supported.
- *actions* - The use of actions that can be used by the planner to transform the state. Each action contains a parameter, precondition and effect keyword.

C.1.3 Task File Only

- *domain* - specifies the name of the domain that this task file is associated with.
- *objects* - the list of objects that are instantiated for the task.
- *init* - a set of grounded predicates that specifies the initial state of the domain.
- *goal* - a set of grounded predicates that specifies the final state that the planner is to achieve.

C.2 pddl_libs, a Library for the Manipulation of Problem Domain Description Language Statements

To allow the integration of standalone PDDL based symbolic planning systems with a robot executive, a method for constructing PDDL files based on the robot's current state was required. To fulfil this role, a library called pddl_libs was developed

At the core of the pddl_libs library is a tree-based data structure similar that allows storage of data in a similar layout to the file structure. Loading of PDDL files into this structure uses a lexical analyser based on the ply library [Beazley, 2016]. PDDL statements in this structure can be searched for, modified or added to. Additional functions include;

- Searching for branches of the PDDL based on tokens
- Helper functions to assist in the addition of facts, actions, and goals

- The creation of domain and task files from the current state of the tree
- An interface to allow automated execution of PDDL domains using the Fast Downward [Helmert, 2006] and Popf2 [Coles et al., 2011] planners, and the parsing of their output files.

For the planning system in Section 5.4 that is based on expanding PDDL statements, an important ability is to substitute statements. This allows new PDDL statements to be generated based on a combination of found statements and the robots belief. The original statements are hidden preventing them from confusing the PDDL planner. This allows the PDDL language to be extended with a statements that are specialised for the task of spatial planning.

C.3 Problem Domain Description Language (PDDL) Code Used For Maritime Planning

Chapters 4 and 5 have used PDDL code to encode domains to be solved by the domain independent planning systems. This section will detail the predicates and actions used to define these domains.

C.3.1 Objects

Using PDDLs *typing* requirement, code can be written that supports object type checking. This allows the detection of possible incorrect code. The object types used by the maritime planning systems are shown in Table C.1.

Table C.1: Types used in the Problem Domain Description Language (PDDL) code used by the maritime planning system.

Type	description
entity	An entity that can act or be acted upon
node	A node of the spatial graph
edge	an edge that connects graph nodes together

C.3.2 Predicates

The PDDL domain indicates the state of the object by the use of logical statements called *predicates*. A summary of the predicates used by the planning system can be seen in Table C.2.

C.3.3 Functions

Functions in a PDDL domain operate in a similar manner to predicates, in that they are representations of the state of a system. However, where predicates are either present or absent, functions represent numerical values. A summary of the functions used in the PDDL domains can be seen in Table C.3

Table C.2: Predicates used in the Problem Domain Description Language (PDDL) code used by the maritime planning system.

Predicate	Description
at ?o - entity ?n - node	The entity is at a node.
accessible ?e - edge ?n1 ?n2 - node	A node is accessible from another node via an edge.
canact ?o - entity	Indicates that the entity can perform actions.
has ?o1 ?o2 - entity	The entity o1 has possession of o2.
haslifeboat ?o - entity	The entity currently has a lifeboat that can be used to rescue a survivor.
depot ?o - entity	The entity is a depot that can provide replacement lifeboats.
rescued ?o - entity	The entity has been rescued.
isdock ?o - entity	The entity is a dock.
docked ?o - entity	The entity is docked.

Table C.3: Predicates used in the Problem Domain Description Language (PDDL) code used by the maritime planning system.

Function	Description
total-cost	The total cost of executing a plan
energyrequired ?e - edge ?n1 ?n2 - node	The energy required to travel from a node to another node via an edge.

C.3.4 Actions

To allow the correct selection of actions, the preqrequisites and effects of these actions need to be encoded in the PDDL domain.

C.3.4.1 Move

The most common of these actions is *move*, which allows an entity to change the *node* that it is *at*. A listing of the code of this action can be seen in Figure C.1. The cost of performing a *move* action can vary. This information is encoded by the numerical predicate *energyrequired*. When the move action is performed, the *total-cost* value is increased by the action cost specified by the *energyrequired* function.

C.3.4.2 Get

The demonstration task shown in Section 4.2.4 used a domain independent planner to choose the order of rescuing a number of survivors. This required the addition of a new action called *get*, the PDDL code for which can be seen in Figure C.2. This only requires an entity that *canact* to be *at* the same *node* as the *entity* to be rescued.

```
(:action move
:parameters (?o - entity ?n1 ?n2 - node ?e - edge )
:precondition (and (at ?o ?n1 )
(accessible ?e ?n1 ?n2 )
(canact ?o ) )
:effect (and (not (at ?o ?n1 ))
(at ?o ?n2 )
(increase (total-cost) (energyrequired ?e ?n1 ?n2))))
```

Figure C.1: Problem Domain Description Language (PDDL) code encoding the *move* action.

```
(:action get
:parameters (?o1 ?o2 - entity ?n- node)
:precondition (and (at ?o1 ?n ) (at ?o2 ?n )
(canact ?o1 ))
:effect (has ?o1 ?o2 ))
```

Figure C.2: Problem Domain Description Language (PDDL) code encoding the *get* action.

C.3.4.3 Rescue

Later versions of the domain would use a *rescue* action that required the presence of a lifeboat as represented by the *haslifeboat* predicate. This version of the action can be seen in Figure C.3. The rescue of a survivor with a lifeboat expends that lifeboat. To replenish the lifeboat supply, the vehicle can use the *collect* action while *at* the same *node* as an *entity* with the *depot* predicate. Code implementing this action can be seen in Figure C.4

C.3.4.4 Dock

The vehicle may be required to dock at the end of its run. The *dock* action combined with the *isdock* predicate allow an entity that can act to be given the *docked* predicate when *at* the same node. Removal of the *canact* predicate ensures that this is the last action that the vehicle can perform. Code for the *dock* action can be seen in Figure C.5.

```
(:action rescue :parameters (?o1 - entity ?n - node ?o2 - entity )
:precondition (and (at ?o1 ?n )
(at ?o2 ?n )
(canact ?o1 )
(haslifeboat ?o1))
:effect (and (rescued ?o2)
(not(haslifeboat ?o1))))
```

Figure C.3: Problem Domain Description Language (PDDL) code encoding the *rescue* action with a lifeboat.

```
(:action collect
:parameters (?o1 - entity ?n - node ?o2 - entity )
:precondition (and (at ?o1 ?n )
(at ?o2 ?n )
(canact ?o1 )
(depot ?o2 ))
:effect (haslifeboat ?o1))
```

Figure C.4: Problem Domain Description Language (PDDL) code encoding the *collect* action that replenishes the lifeboat supply.

```
(:action dock
:parameters (?o1 - entity ?n - node ?o2 - entity )
:precondition (and (at ?o1 ?n )
(at ?o2 ?n )
(canact ?o1 )
(isdock ?o2 ))
:effect (and (docked ?o1)
(not (canact ?o1))))
```

Figure C.5: Problem Domain Description Language (PDDL) code encoding the *dock* action that docks the vehicle at the end of a mission.

C.4 Inter-operation of Domain Independent Planning with the Robotics Operating System (ROS)

As outlined in Section 5.5, the custom message encoding system of the Robotics Operating System (ROS) has sufficient expressiveness to encode domain-independent domains within it's structure. This section will provide a brief overview of how this can be performed.

At their most basic level, a PDDL domain and task are tree structures containing a number of statements encoding the desired relationships between actions and predicates. While ROS messages do not support recursion, they do support the inclusion of sub-messages zero or more times. Using this, a message structure can be constructed that encodes the commonly used features of a PDDL domain.

C.4.1 The Predicate Message

The most fundamental of the messages used for domain encoding was the *Predicate* message. This message encodes a single grounded predicate, and is used as the basis of more complex message types. The message encoding is shown in Figure C.6.

```
string name
string[] parameter
int8 relation
int8 TRUE=0
int8 FALSE=1
```

Figure C.6: Custom Robotics Operating System (ROS) message encoding a predicate.

C.4.2 The NumericPredicate Message

In addition to the true and false information provided by the standard predicates a system that encodes numerical values such as action costs will require a method of communicating these values. The *NumericPredicate* message encodes a numerical relationship between a set of symbols representing the function and a numerical value. The custom message encoding is shown in Figure C.7.

C.4.3 The Parameter Message

Predicates specify the facts of a planning system by providing relationships between symbols. To ensure that correct typing is performed when constructing predicates, a method of communicating the object types is required. The *Parameter* message is used to communicate this type information. A listing of the custom ROS message can be seen in Figure C.8.

```
Predicate[] functions
int8 relation
int8 GREATERTHAN=0
int8 LESSTHAN=1
int8 EQUAL=2
int8 ADD=3
int8 SUBTRACT=4
int8 SET=5
int32 value
```

Figure C.7: Custom Robotics Operating System (ROS) message encoding a numeric predicate.

```
string name
string type
```

Figure C.8: Custom Robotics Operating System (ROS) message encoding a parameter.

C.4.4 The Action Message

The *action* message encodes an action. Preconditions and effects are encoded as zero or more *Predicate* and *NumericPredicate* messages.

C.4.5 The Object and Objects Message

The robots control and estimation systems communicate their estimated state of the system with the planning system by the use of *Objects* messages. The definition of the *Object* message can be seen in Figure C.10. The *Objects* message contains zero or more *Object* messages. *Objects* messages are used to communicate both the state of the spatial environment, including the action costs for traversing between nodes as seen in Figure C.11 and the state of objects within the environment as seen in Figure C.12. The use of these messages allow separation between the executive of the robot and the planning system.

```
string name
Parameter[] parameters
Parameter[] numeric_parameters
Predicate[] preconditions
NumericPredicate[] numeric_preconditions
Predicate[] effects
NumericPredicate[] numeric_effects
```

Figure C.9: Custom Robotics Operating System (ROS) message encoding an action.

```
string name
string type
int32 node_id

bool tracked
geometry_msgs/Point position

Predicate[] predicates
NumericPredicate[] numeric_predicates
```

Figure C.10: Custom Robotics Operating System (ROS) message encoding an objects properties.

```
name:  "bridge5"
type:  "edge"
node_id:  4
tracked:  False
position:
x:  -82.6819422992
y:  -966.084036425
z:  0.0
predicates:
-
name:  "accessible"
parameter:  [bridge5, node4, node14]
relation:  0
-
name:  "accessible"
parameter:  [bridge5, node14, node4]
relation:  0
numeric_predicates:
-
functions:
-
name:  "energyrequired"
parameter:  [bridge5, node4, node14]
relation:  0
relation:  2
value:  232
-
functions:
-
name:  "energyrequired"
parameter:  [bridge5, node14, node4]
relation:  0
relation:  2
value:  232
-
```

Figure C.11: Part of the *Objects* message used to communicate state with the planning system. This portion encodes the connectivity between two nodes.

```
name:  "wamv"
type:  "entity"
node_id:  2
tracked:  False
position:
x:  -667.681942299
y:  -783.084036425
z:  0.0
predicates:
-
name:  "free"
parameter:  [wamv]
relation:  0
-
name:  "canact"
parameter:  [wamv]
relation:  0
-
name:  "haslifeboat"
parameter:  [wamv]
relation:  0
numeric_predicates:  []
-
```

Figure C.12: Part of the *object* message used to communicate state with the planning system. This portion encodes the state of the Autonomous Surface Vessel (ASV).

C.4.6 The Plan Message

When the domain-independent planning system has generated a plan for execution, it's state should be communicated for logging and monitoring purposes. The custom ROS message type *Plan* shown in Figure C.13. The *Plan* message comprises zero or more *PlanAction* messages as shown in Figure C.14.

C.4.7 Actionlib Integration

Once the planning system has generated a plan, it needs to be dispatched to the systems that will implement the actions. In this planning system, this is performed using the *actionlib* library [Open Source Robotics Foundation, 2014a] using the *PDDLAction* custom action definition shown in Figure C.15. When it is time for an action to be performed, an *actionlib* server calls the corresponding client with the name of the action to be performed and the corresponding grounded objects in the strings section. By monitoring the state of the objects in the system, this information is sufficient for the client to complete the action. The client responds with success or failure when the action is complete.

```
bool execute
PlanAction[] actions
```

Figure C.13: Custom Robotics Operating System (ROS) action definition used to publish the generated plan.

```
string name
float32 time
string[] values
```

Figure C.14: Custom Robotics Operating System (ROS) action definition used by the planner to publish individual actions.

```
string action
string[] strings
---
bool result
---
int32 status
int32 final_cost
```

Figure C.15: Custom Robotics Operating System (ROS) action definition used by the planner to invoke actions.

C.5 Conclusion

This chapter has provided brief information on the software, PDDL code, and ROS messages that are used to communicate the domain and task information between the executive, planning system and domain-independent planner that comprise the planning system developed for this thesis.

Appendix D

Additional Figures Showing Motion Plans

Due to their size, some figures showing vehicle motion plans were unable to be shown in the body of this thesis. These figures have been reproduced here.

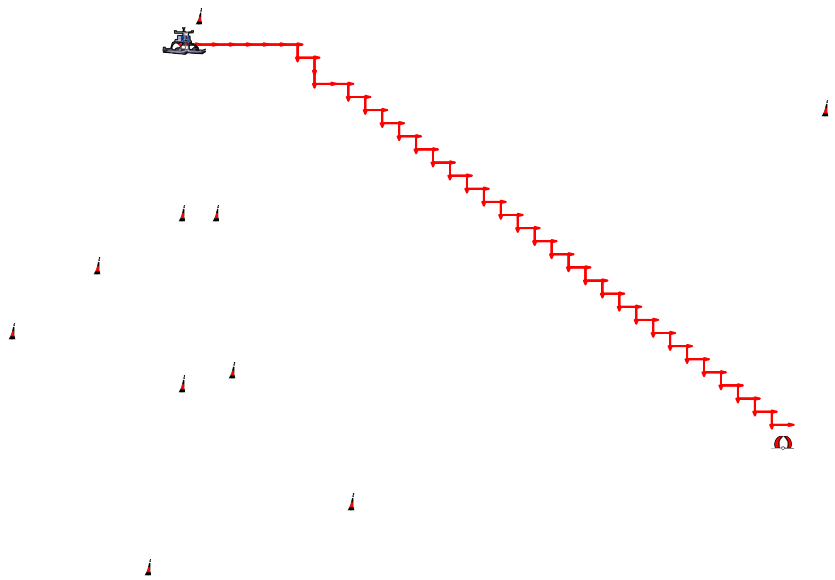


Figure D.1: Sample motion plan generated by Fast Downward with Lazy Greedy search, and the Context Enhanced-Additive (CEA) heuristic. This result is identical to the result of using the Fast Forward (FF) heuristic as seen in Figure 4.2.

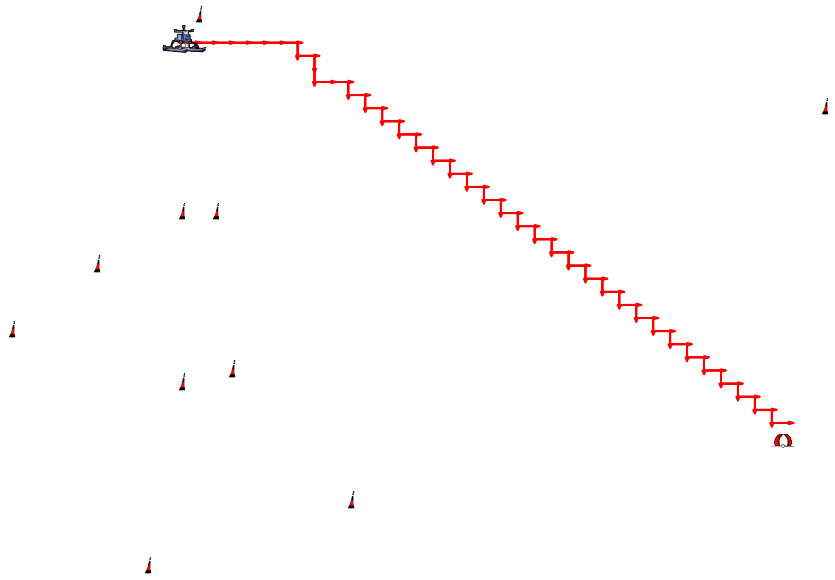


Figure D.2: Sample motion plan generated by Fast Downward with Lazy Greedy search, and the Dual heuristic. This result is identical to the result of using the Fast Forward (FF) heuristic as seen in Figure 4.2.

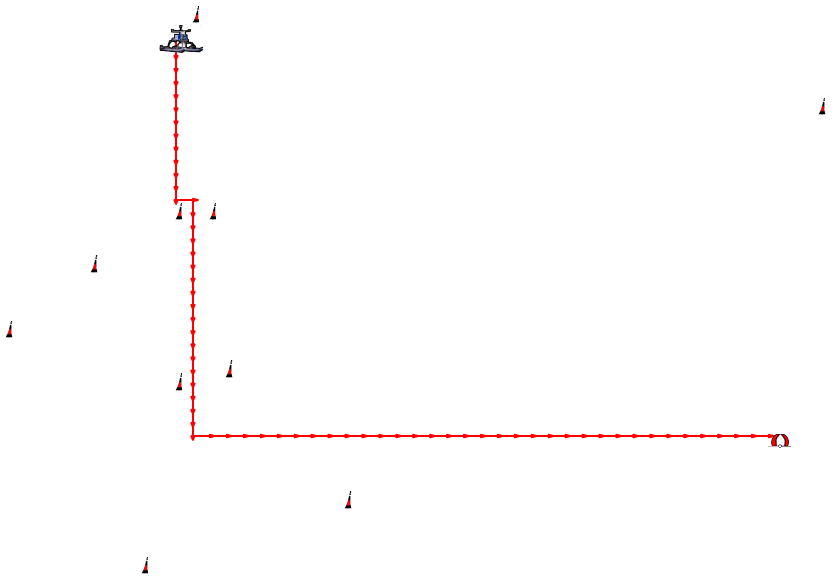


Figure D.3: Sample motion plan generated by Fast Downward with A* search, and the Blind heuristic. This result is identical to the result of using the Landmark-Cut (LM-cut) heuristic as seen in Figure 4.3.

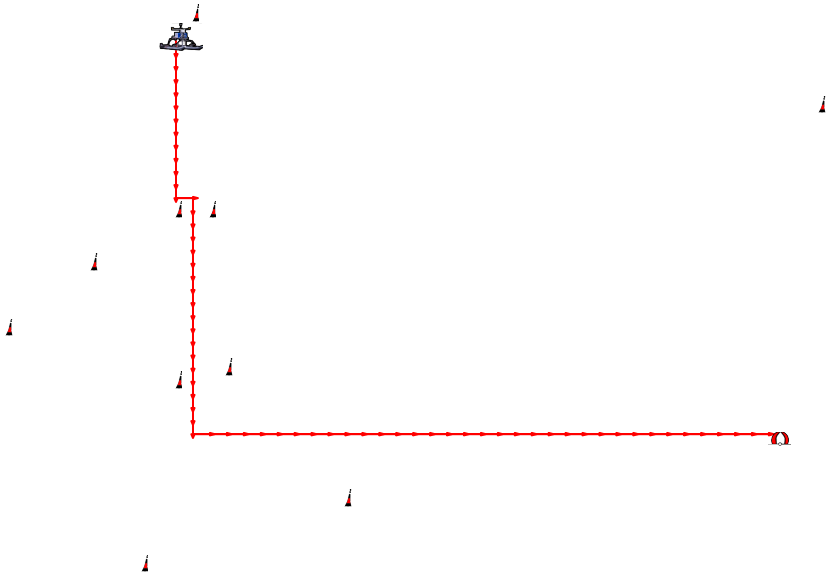


Figure D.4: Sample motion plan generated by Fast Downward with A* search, and the Pattern Database (iPDB) heuristic. This result is identical to the result of using the Landmark-Cut (LM-cut) heuristic as seen in Figure 4.3.

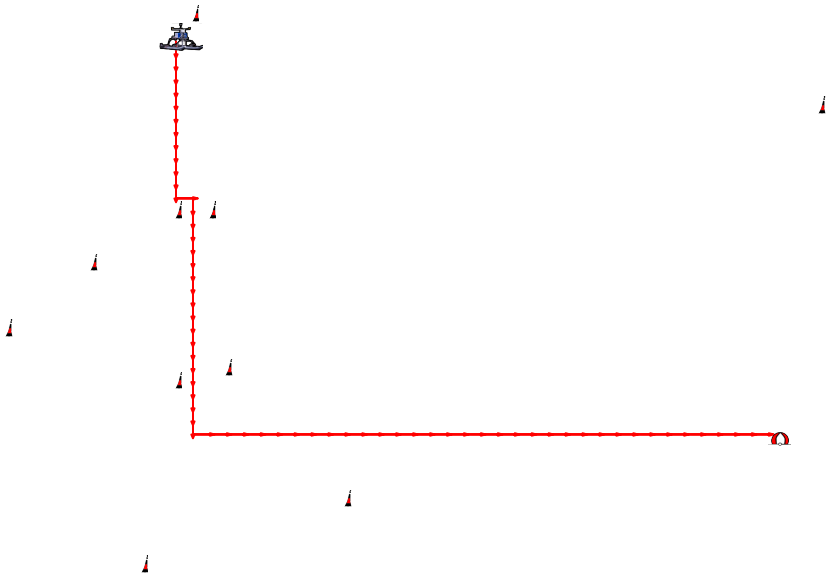


Figure D.5: Sample motion plan generated by the popf-2 planner. This result is identical to the result of using the Landmark-Cut (LM-cut) heuristic as seen in Figure 4.3.

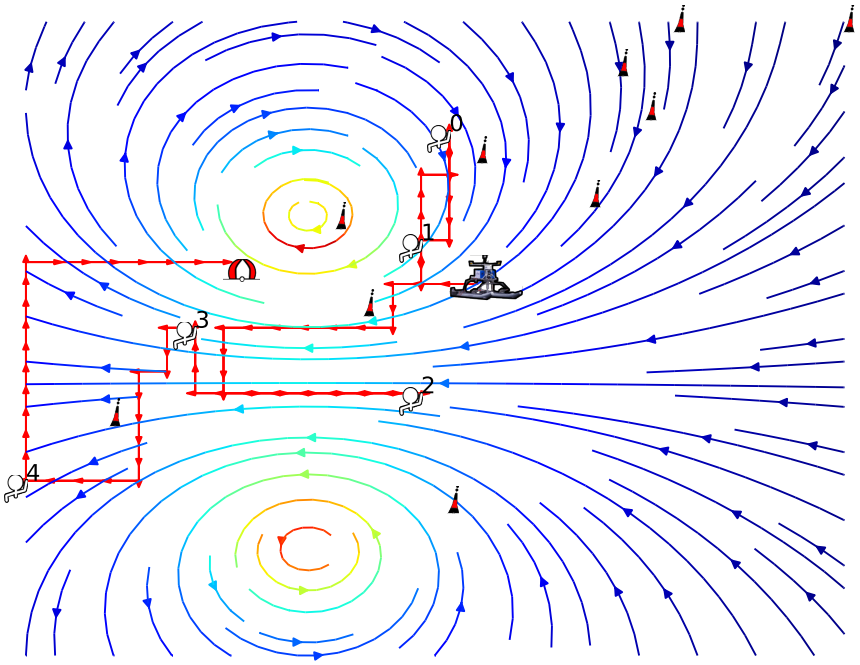
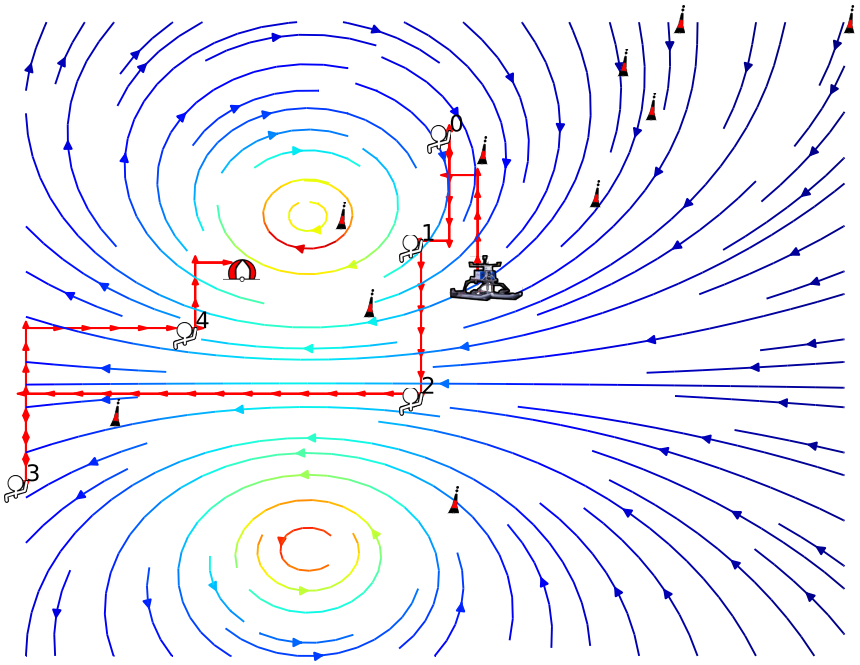
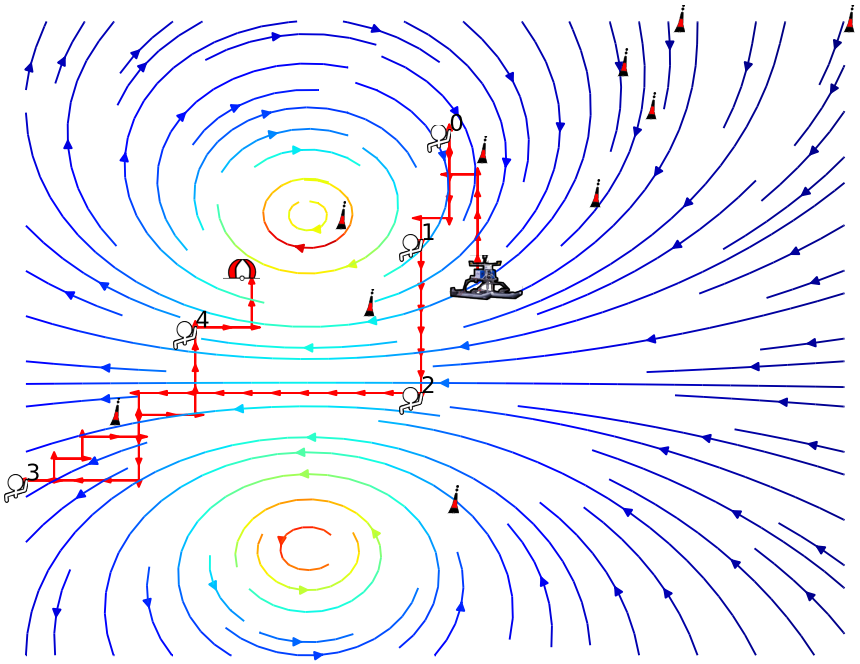


Figure D.6: Sample mission plan to rescue five subjects as generated by Fast Downward with Lazy Greedy search through an asymmetric cost environment and Dual Heuristic. Numbers indicate the order in which subjects were rescued. Result was identical to Fast Downward with the Fast Forward heuristic as seen in Figure 4.12a



(a)



(b)

Figure D.7: Sample mission plans to rescue five subjects as generated by Fast Downward with A* search through an asymmetric cost environment. Numbers indicate the order in which subjects were rescued (a) Fast Downward with Blind heuristic (b) Fast Downward with Pattern Database (iPDB) heuristic. Mission plans were similar to that generated by Fast Downward with the Landmark-Cut (LM-cut) heuristic as seen in Figure 4.13(a).

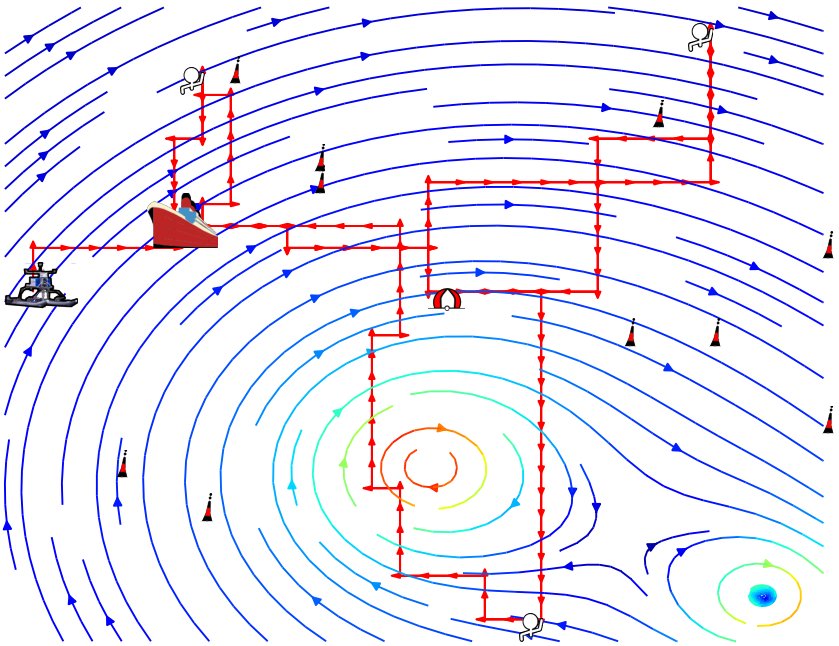
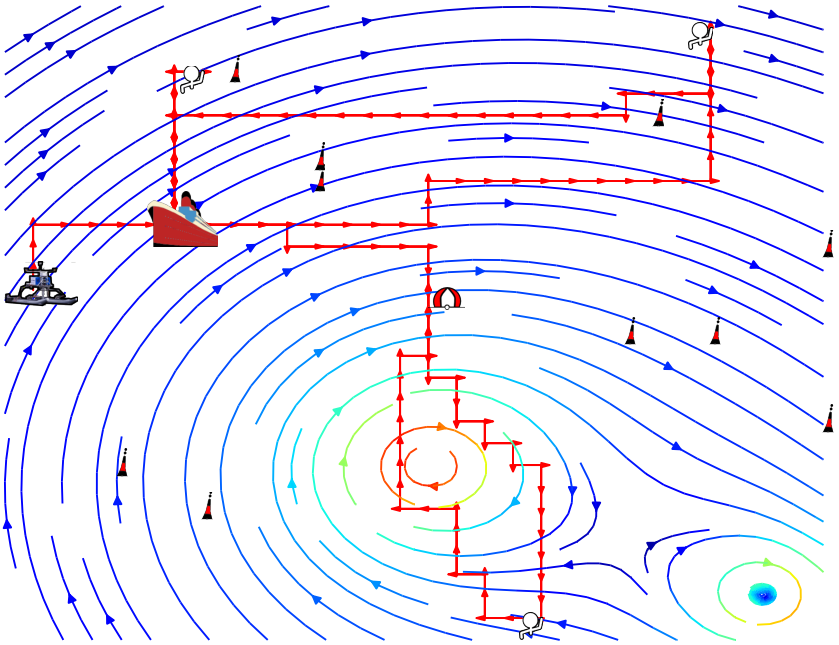
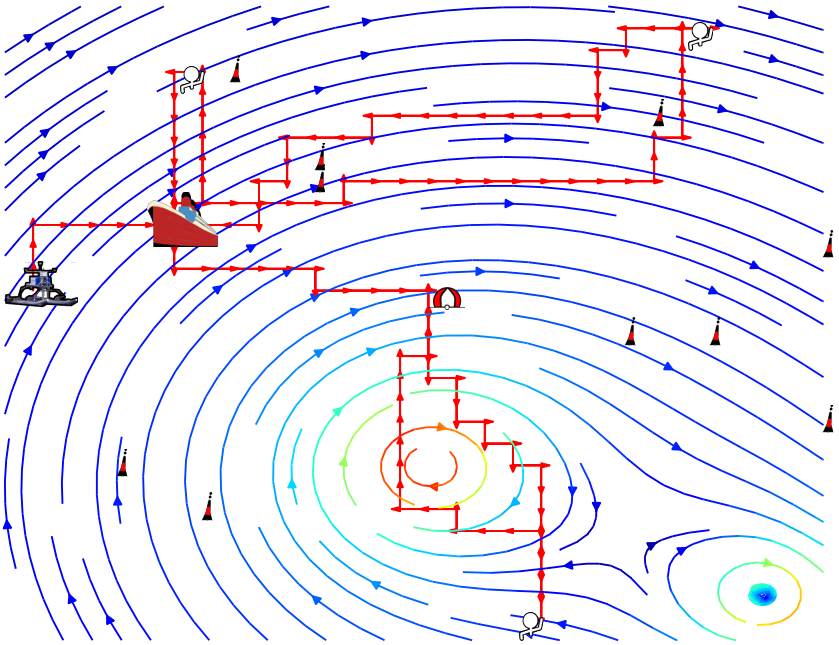


Figure D.8: Sample motion plan for domains including move, collect, and deploy actions using Fast Downward, Lazy Greedy Search, and the Dual Heuristic. This run included three subjects for rescue and one supply ship. Arrows on the red vehicle track indicate the direction of motion. Plots of Lazy Greedy Search with the Fast Forward and Context-Enhanced Additive heuristics can be found in Figure 4.17.

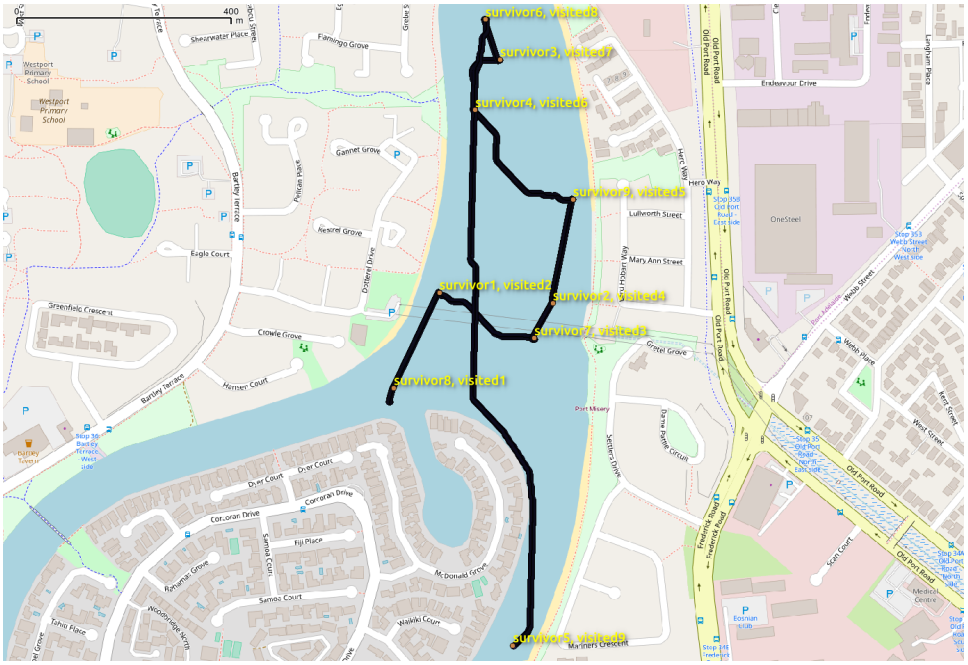


(a)

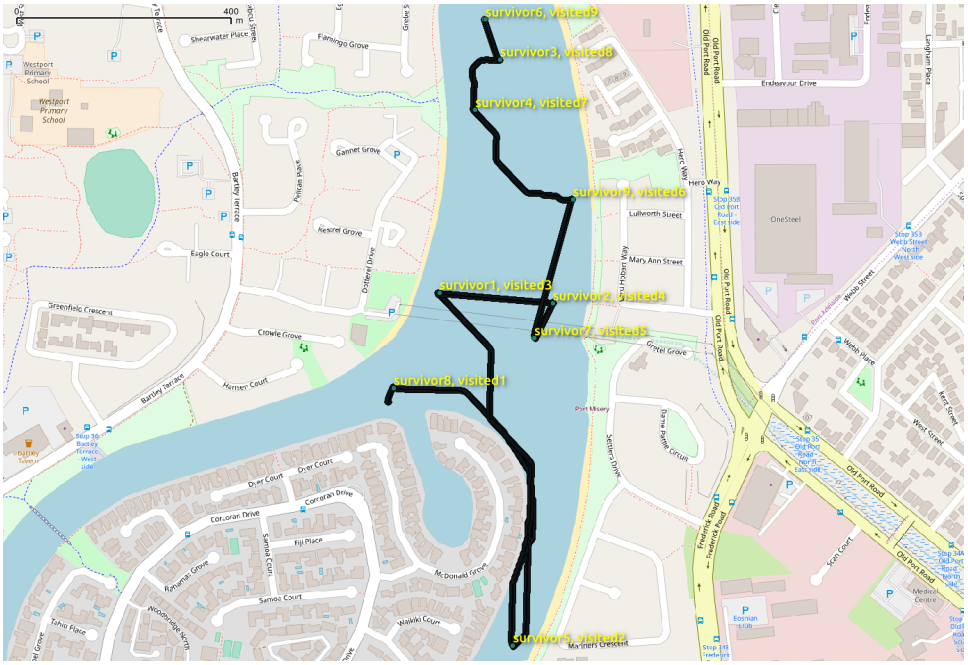


(b)

Figure D.9: Sample motion plans for domains including move, collect, and deploy actions using Fast Downward and A* search. These runs included three subjects for rescue and one supply ship. Arrows on the red vehicle track indicate the direction of motion. (a) Fast Downward with Blind heuristic (b) Fast Downward with Pattern Database (iPDB) heuristic. Fast Downward with the Landmark-Cut (LM-cut) heuristic can be found in Figure 4.18(a).

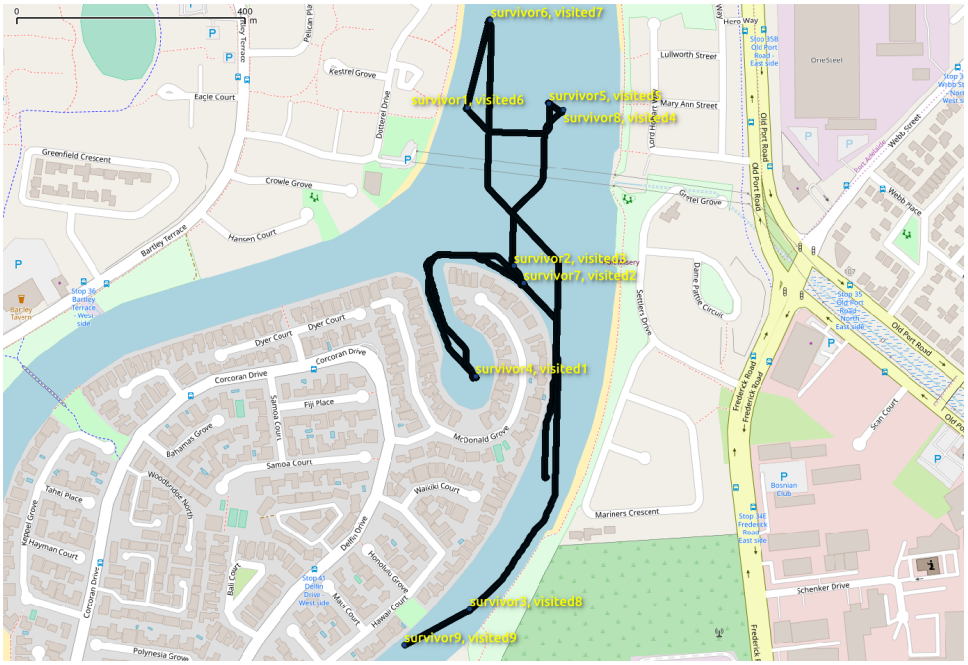


(a)

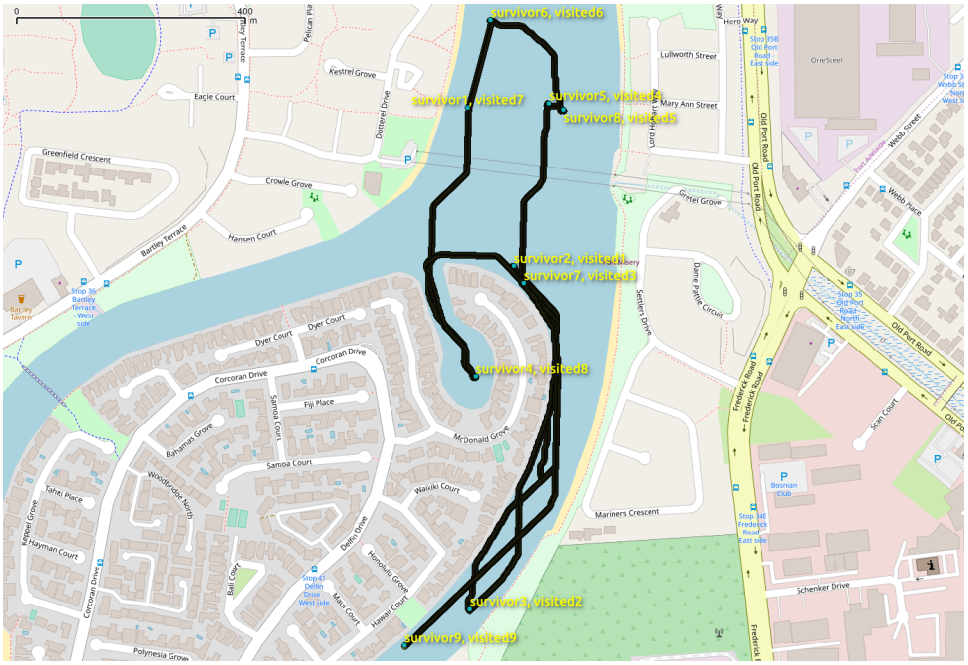


(b)

Figure D.10: Vehicle tracks for the first task configuration (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.12. Backing image is ©OpenStreetMap contributors.



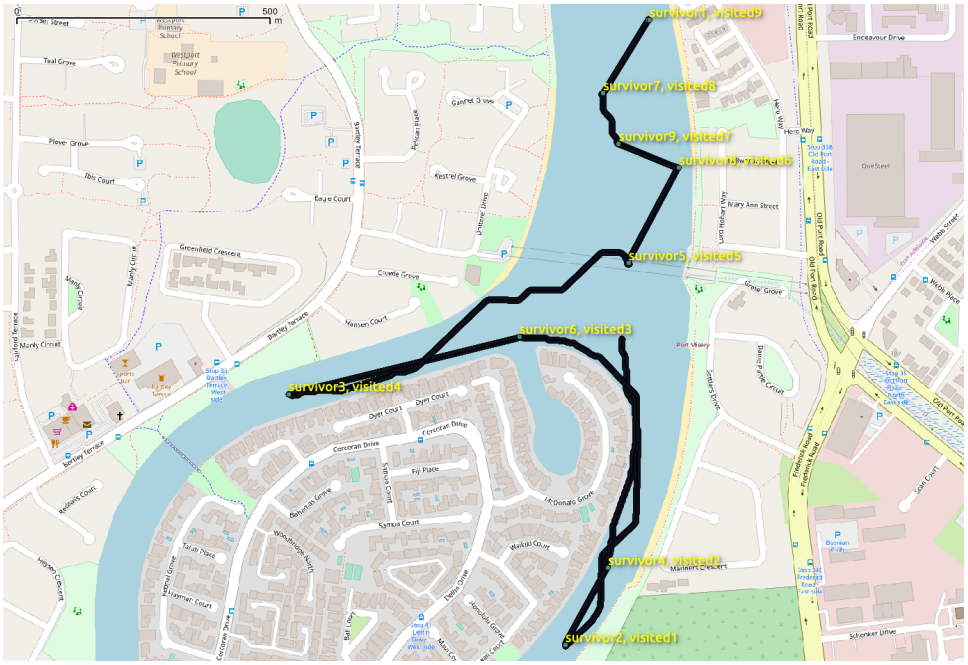
(a)



(b)

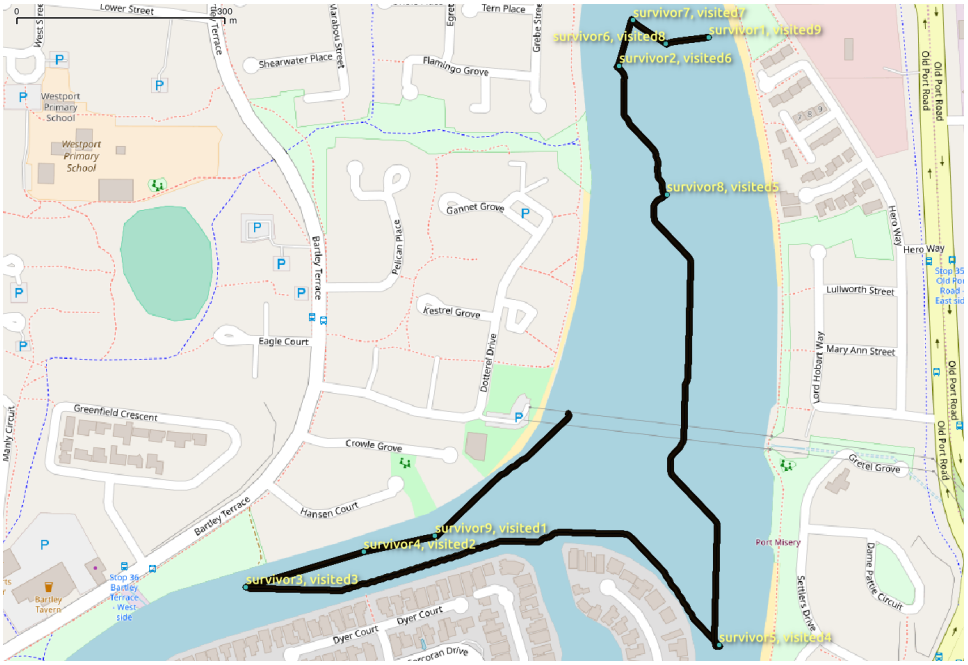
Figure D.11: Vehicle tracks for the second task configuration (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.12. Backing image is ©OpenStreetMap contributors.

(a)

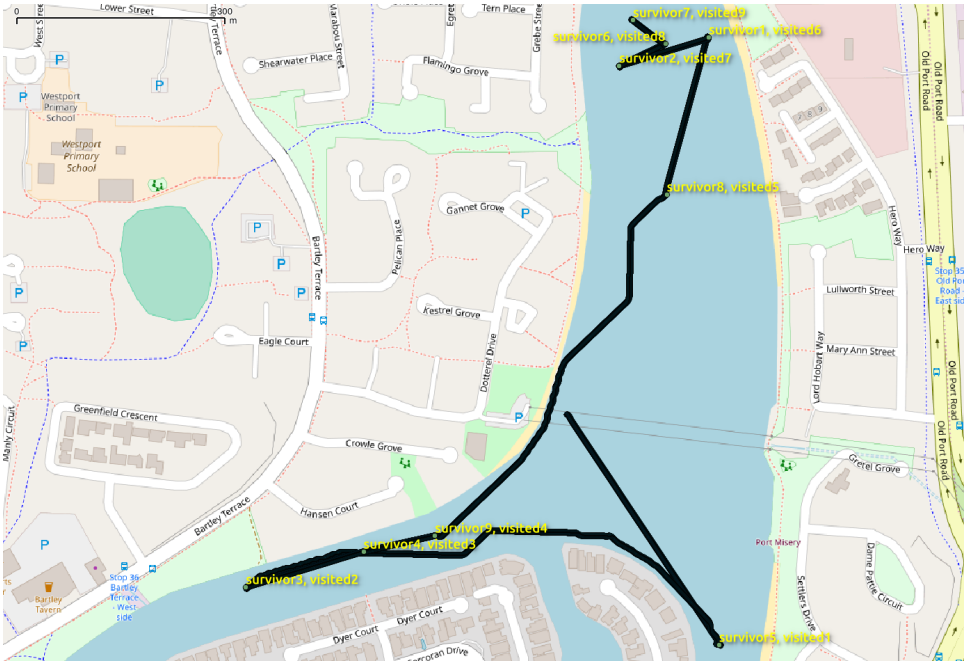


(b)

Figure D.12: Vehicle tracks for the third task configuration (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.12. Backing image is ©OpenStreetMap contributors.

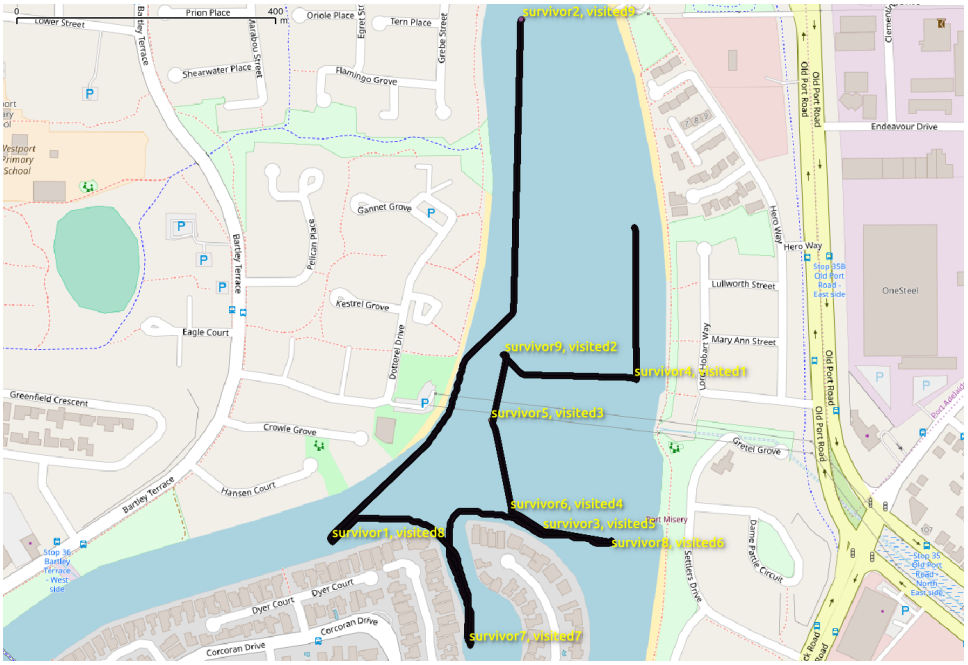


(a)

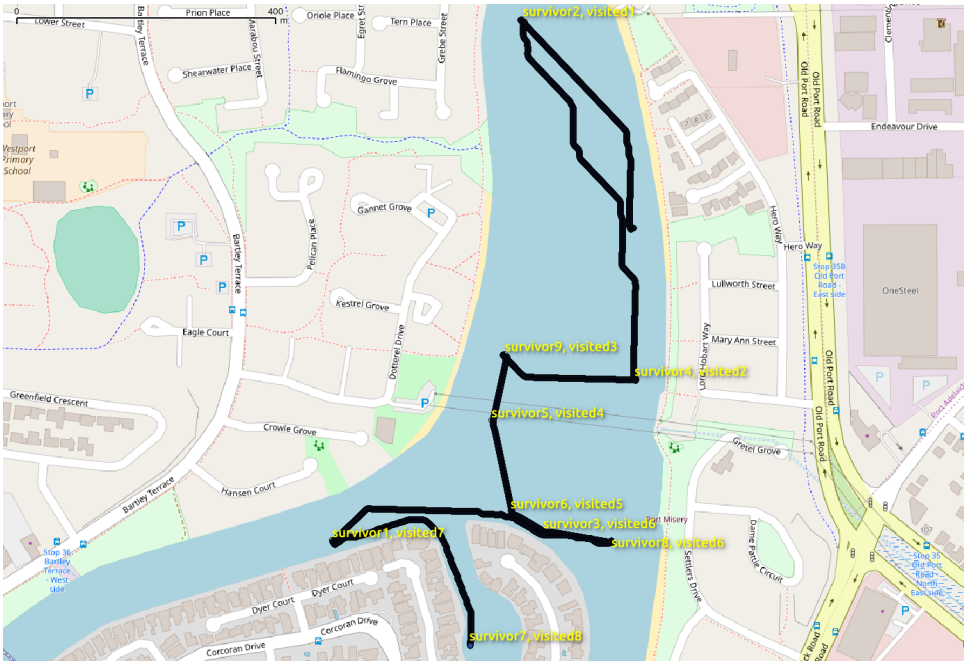


(b)

Figure D.13: Vehicle tracks for the fifth task configuration (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.12. Backing image is ©OpenStreetMap contributors.

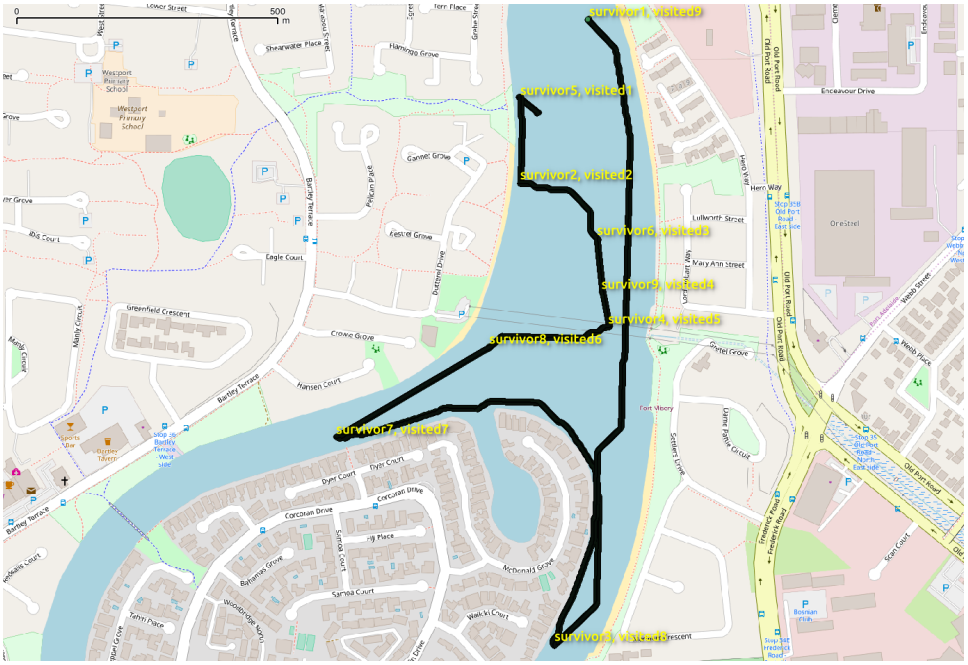


(a)

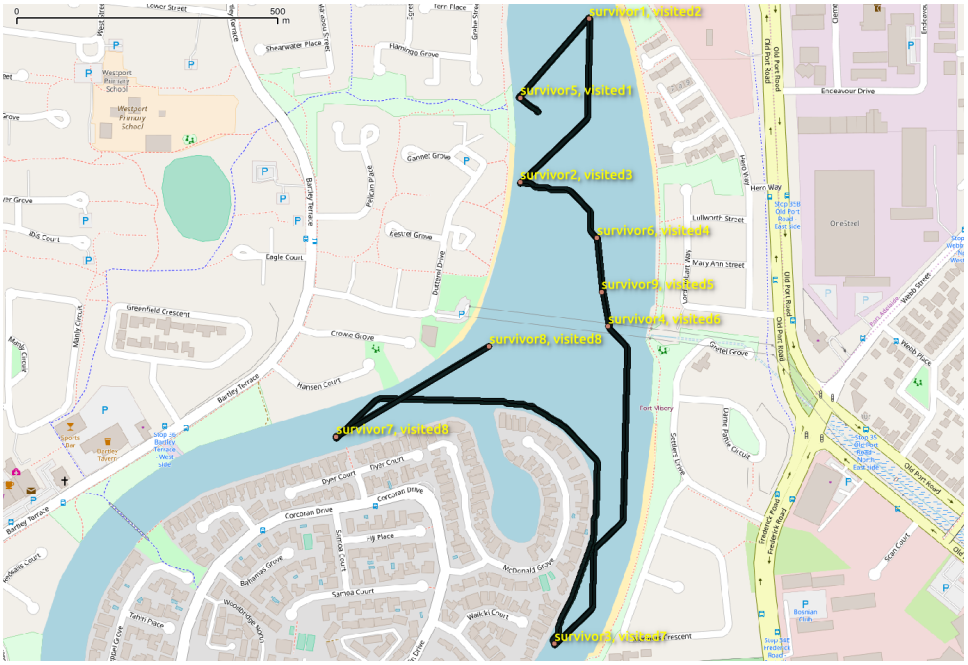


(b)

Figure D.14: Vehicle tracks for the first task configuration. (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic With Refinement planner. These tracks come from the same test runs as Figure 5.15. Backing image is ©OpenStreetMap contributors.



(a)



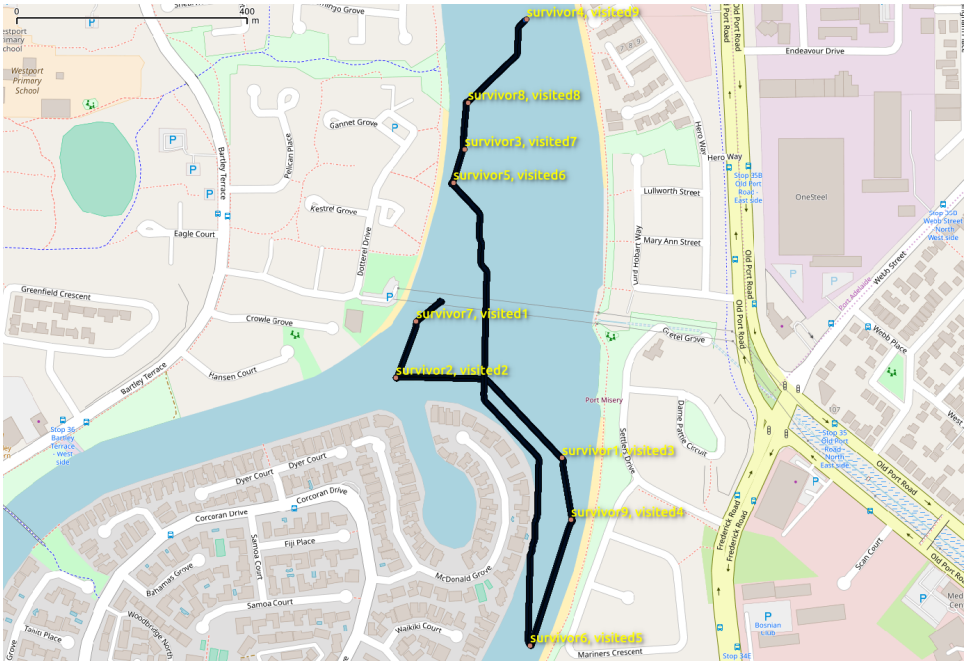
(b)

Figure D.15: Vehicle tracks for the second task configuration. (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic With Refinement planner. These tracks come from the same test runs as Figure 5.15. Backing image is ©OpenStreetMap contributors.

(a)

(b)

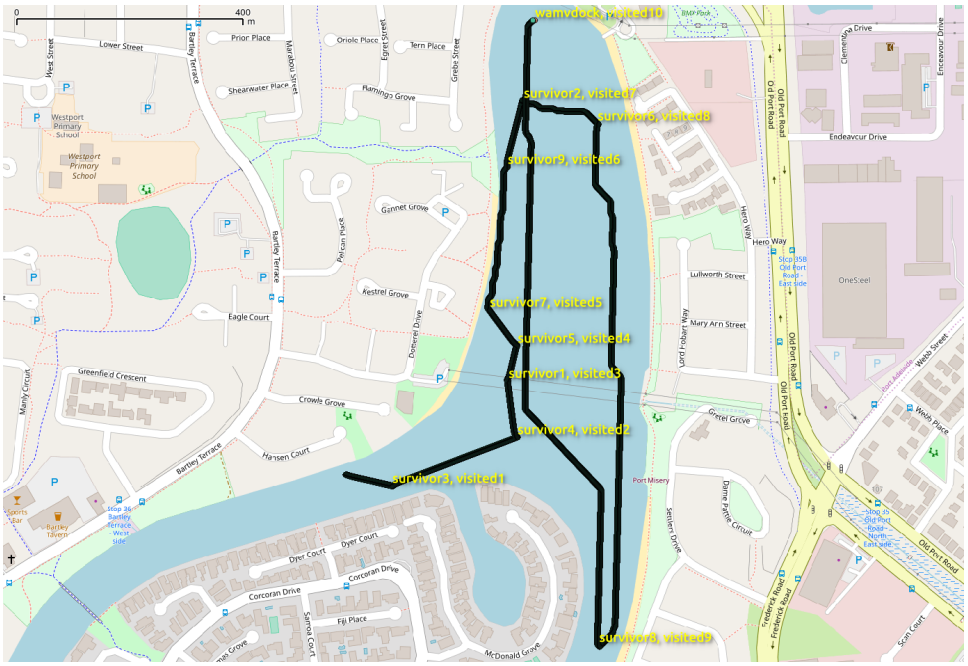
Figure D.16: Vehicle tracks for the fourth task configuration. (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic with Refinement planner. These tracks come from the same test runs as Figure 5.15. Backing image is ©OpenStreetMap contributors.



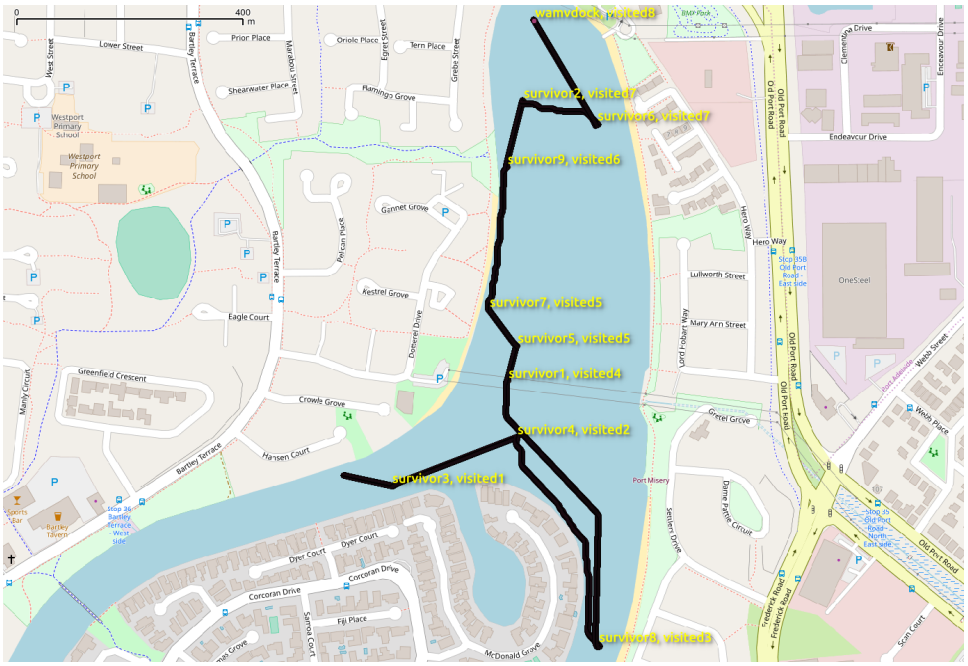
(a)

(b)

Figure D.17: Vehicle tracks for the fifth task configuration. (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic with Refinement planner. These tracks come from the same test runs as Figure 5.15. Backing image is ©OpenStreetMap contributors.

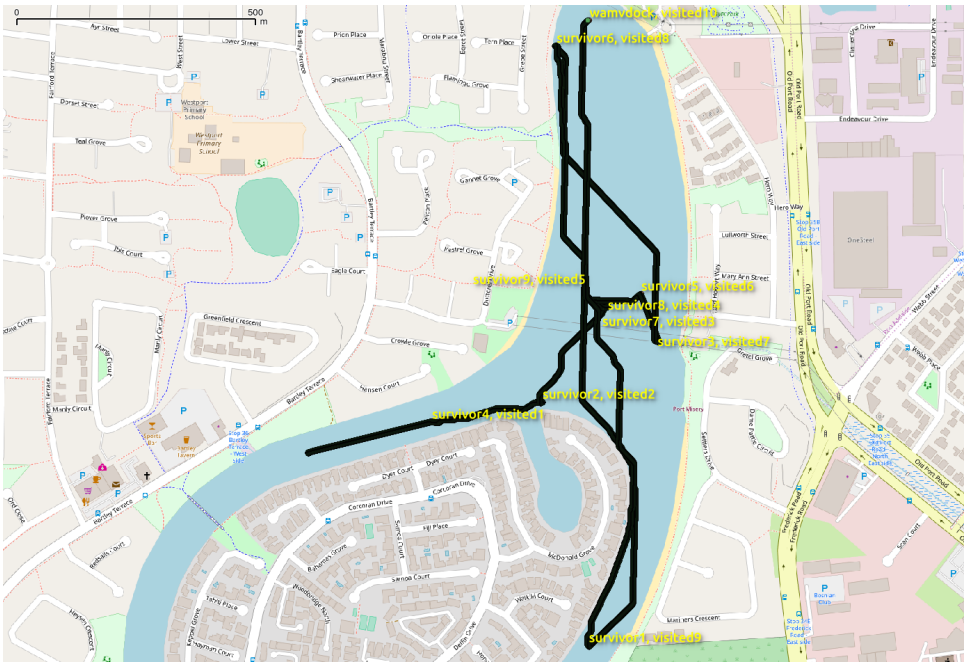


(a)

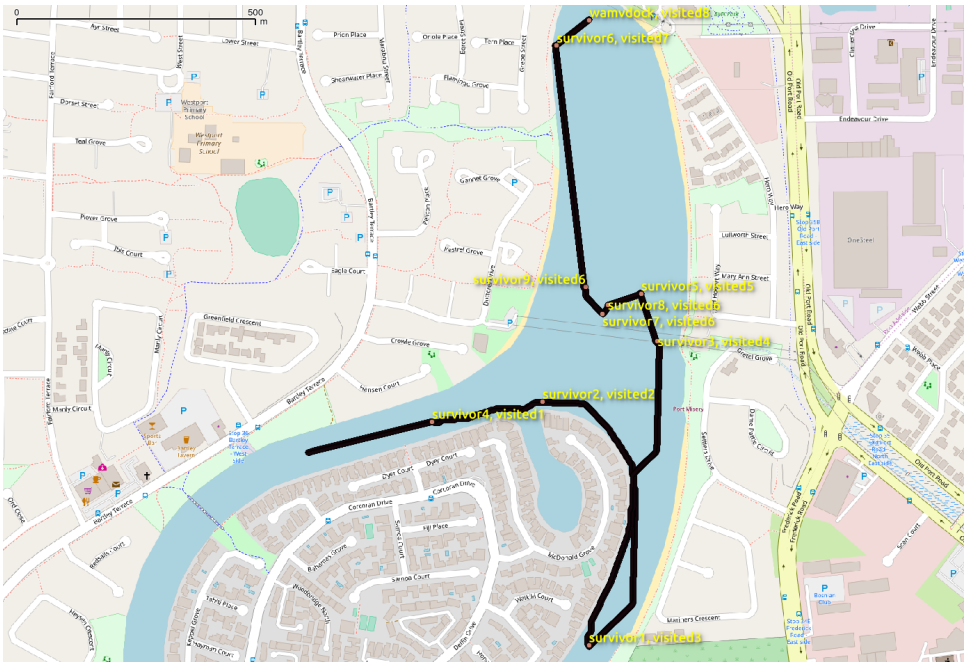


(b)

Figure D.18: Vehicle tracks for the second task configuration. Vehicle must dock at the end of mission (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.16. Backing image is ©OpenStreetMap contributors.

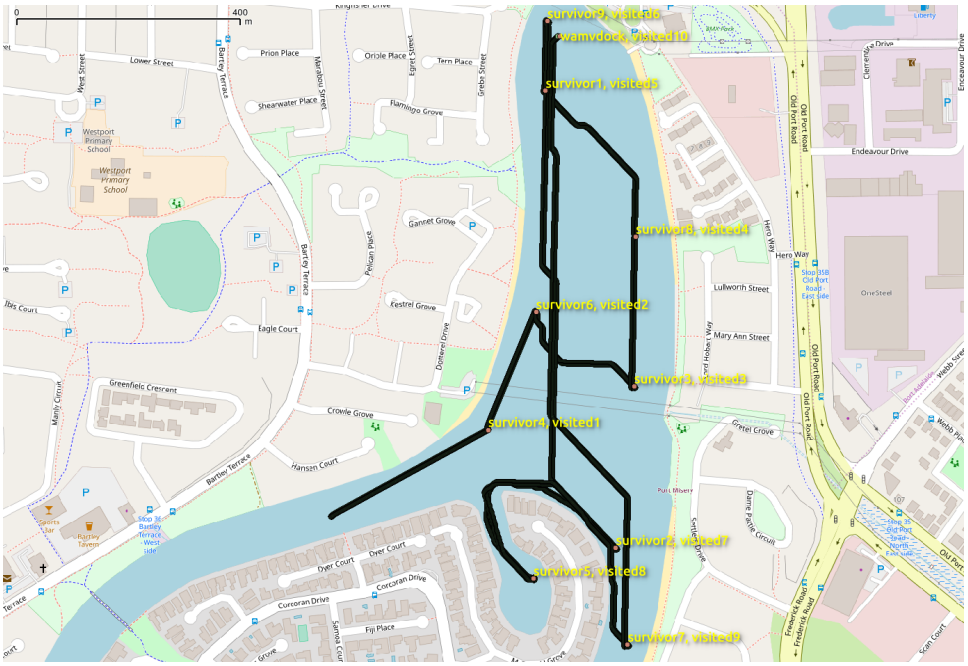


(a)

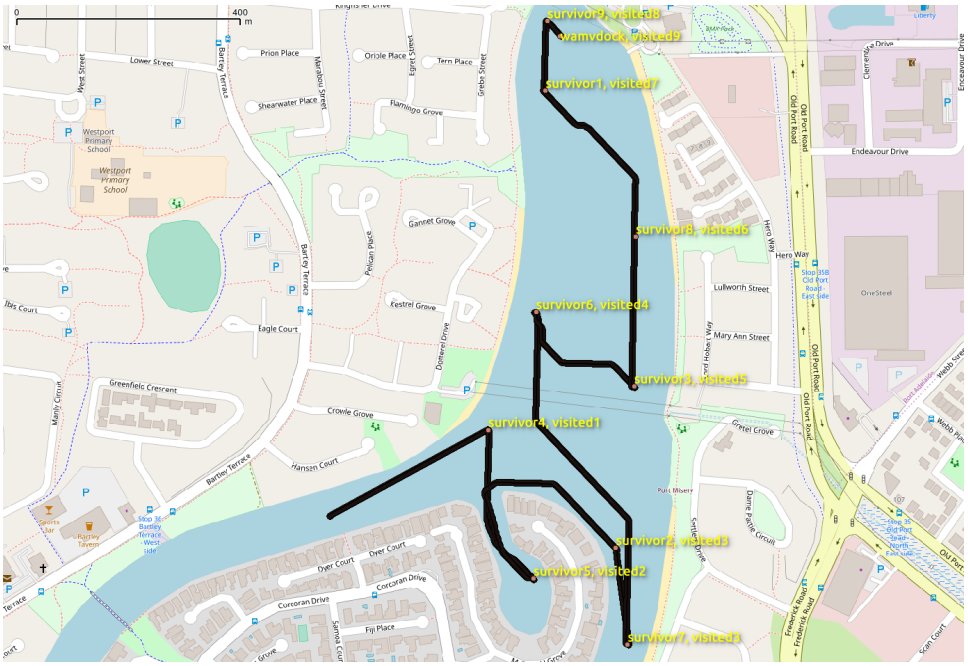


(b)

Figure D.19: Vehicle tracks for the fourth task configuration. Vehicle must dock at the end of mission (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.16. Backing image is ©OpenStreetMap contributors.



(a)



(b)

Figure D.20: Vehicle tracks for the fifth task configuration. Vehicle must dock at the end of mission (a) Vehicle track with Greedy planner (b) Vehicle track with Symbolic planner. These tracks come from the same test runs as Figure 5.16. Backing image is ©OpenStreetMap contributors.

Appendix E

Guidance for Autonomous Surface Vessels

E.1 Control of Maritime vehicles with the Robotics Operating System (ROS)

Through the use of kinematics, and when neglecting friction and wheel slip, an exact solution for the motion of wheeled ground vehicle can be derived. Since a maritime vehicle lacks a solid surface to act against, such a simple solution for motion cannot be derived for simulation or control. The control system must instead consider the effect of the hydrodynamic forces on a vehicle when deriving a motion model for the vehicle.

One solution to the problem of controlling a maritime vehicle is to develop a force-based model, such as the one implemented by Matthew Anderson for the TopCat Autonomous Surface Vessel (ASV) for the 2014 Maritime RobotX competition [Anderson, 2014]. The force-based model approach uses a model of the expected forces on the vehicle derived from performance data as measured by the RobotX team members at the Australian Maritime College. By using these forces as a base, the required difference in force to create the desired vehicle performance can be calculated. This solution was capable of controlling the vehicle, however it did not extend to modelling the control lag, and did not include support for a wind model.

Lacking a full model of the vehicle hydrodynamics, a simplified model of the vehicles motion can be developed empirically. The basic model of the vehicle was expressed as a set of discrete state equations with the state vector X defined as;

$$X = \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix} \tag{E.1}$$

where v_x and v_y are the velocity components in the vehicle frame, while ω is the angular velocity. Anderson characterised the non-linear dynamics of the vehicle to the equation:

$$M\dot{x} + C(x)x + D(x)x = \tau \tag{E.2}$$

where M is the mass and inertia matrix, C is the coriolis matrix, D is a damping matrix and τ is the external forces on the vehicle [Anderson, 2014].

Derivation of these terms is difficult, requiring extensive modelling of the vehicle and testing in controlled conditions to derive an effective solution. Given, however, that the vehicle normally manoeuvres at only a relatively small speed range, the assumption can be made that within this range the predicted future state can be linearly derived from the current state X_n and a control input u_n. Neglecting the effect of external forces, The vehicle's future state Y_{n+1} can be estimated using a state space model;

$$X_{n+1} = A * X_n + B * u_n \tag{E.3}$$

$$Y_{n+1} = C * X_{n+1} + D * u_n \tag{E.4}$$

The external observed state is given by Y, if this is the linear and angular velocities, then C will be the identity matrix. If the coefficients of D are zero, this can be written as;

$$Y_{n+1} = X_{n+1} \tag{E.5}$$

External forces may include water currents, but the vehicle is currently only able to perceive

wind and propulsion effects. Thus the vector u_n is expanded as;

$$u_n = \begin{bmatrix} \omega_{prop_{port}} \\ \omega_{prop_{starboard}} \\ v_{wind_x} \\ v_{wind_y} \end{bmatrix}_n \tag{E.6}$$

Derivation of the coefficients of such a model normally requires testing in a specialised facility. Lacking access to such facilities, these values were derived empirically from operation in the field. These data was processed by Dr. Andrew Lammas using MATLAB's system identification toolbox to produce the state space model.

E.1.1 State Selection

For the TopCat ASV, a lag of approximately 700ms between change in throttle settings and the motors beginning to respond has been observed. Combined with a 1000ms ramp time for the motors, any change made to the motor throttle setting would take almost two seconds to take effect. Modelling only a single step into the future will not consider these effects.

$$X_{n+m} = \Pi_{k=0}^{m}(A * X_{n+k} + B * u_{n+k}) \tag{E.7}$$

$$Y_{n+m} = I * X_{n+m} \tag{E.8}$$

Given a desired value of the vehicle state, $Y_{desired}$, and a set of possible throttle settings $U_{throttle}$ combined with the current wind data u_{wind}, a solution can then be sought for $u_{throttle}$ that minimises the absolute value of the weighted sum of the difference between the desired

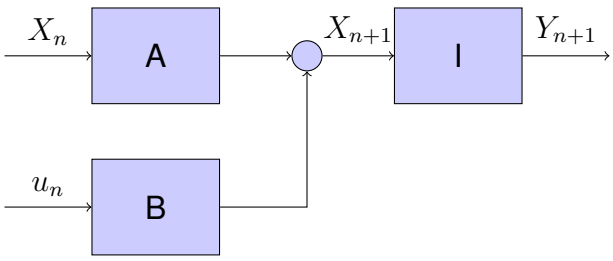


Figure E.1: Block diagram of state space model from Equations E.7 and E.8.

and predicted states.

$$\min_{u_n \in U_{throttle}, u_{wind}} |X_{weight} * (Y_{desired} - Y_{n+m,u_n})| \tag{E.9}$$

Variations in the weighting vector can allow vehicle behaviours to be prioritised, including the preference for minimising heading or velocity errors.

Over a period of several seconds, as the vehicle rotates in relation to the wind direction, the wind effect can also vary significant on the vehicle. Thus, the trajectory followed by the vehicle will not be a simple geometric shape. This lack of regularity makes a difficult to produce a closed form solution for the vehicle's trajectory. To enable the evaluation of the vehicle's future state, a multi-resolution grid search was used to generate potential vehicle trajectories. By projecting forward the potential trajectories of the vehicle, a combination of control inputs can be selected that will most closely match the intended vehicle behaviour.

E.1.2 Guidance

As stated in the introduction, the primary task of the vehicle is to observe the environment. This requires it to not only correctly drive to a goal pose, but to travel across a locus of observation points, typically expressed as a transect line. For a dataset that tracks the evolution of an environment over time, this transect must also be be repeatable. For this reason, the transect line, specified as a combination of a location given in geodetic coordinates, and a heading defining the line of approach, was chosen as the primary primitive of the motion planning system. The vehicle trajectory is generated using a line of sight algorithm that intersects a circle 5 m radius with the transect line, and the vehicle attempts to minimise the heading error to the solution that is closest to the goal. In the case of no solution, the point of closest approach on the line is used.

Additional primitives support operations such as driving to a goal position, turning to a heading, travelling at a fixed speed, and operating the motors at a fixed throttle setting. These additional primitives can be used for testing the vehicle's performance.

Initial testing of the primitive system was performed at West Lakes, and it was later used as

part of a demonstration on the River Torrens. This testing showed that the model worked well when the forward velocity was limited, however when travelling at cruising speed the vehicle consistently underestimated its turning rate. This underestimation resulted in the vehicle overshooting its transect when converging onto a new course. To limit this overshoot, an additional control term was added to the system. This used Equation E.10 to limit the velocity target of the vehicle when the heading differed from the transect. Use of this allowed the vehicle to more rapidly converge onto the goal transect.

$$v_x = v_x(max(\cos(\theta_r - \theta_h), 0))^k \tag{E.10}$$

E.2 Experimental Results

As covered in Chapter 5, the TopCat vehicle was used for a simulated Environmental Monitoring task in West Lakes. During this task, the vehicle was sent on a repeated transect pattern across the northern boating lake. The data from this event were recorded and analysed.

To better understand the vehicle's behaviour, the recorded data were segmented into sections by commanded transect, and the guidance error was plotted in the frame of reference of the goal. The resulting plots can be seen in Figures E.2 and E.3, which represent the error of the control system at tracking the line-of-sight goal, and the final transect goal. Since in most plots, the vehicle was given a new goal transect in a direction anti-parallel to its current course and separated by a lateral distance of 20m, these can be considered amongst the worst case of error that control system will need to recover from.

Examination of Figure E.2 shows that the vehicle does overshoot the target angle, with most of the trajectories settling after approximately 20 seconds. However the converged error is too small to infer the true accuracy of the system.

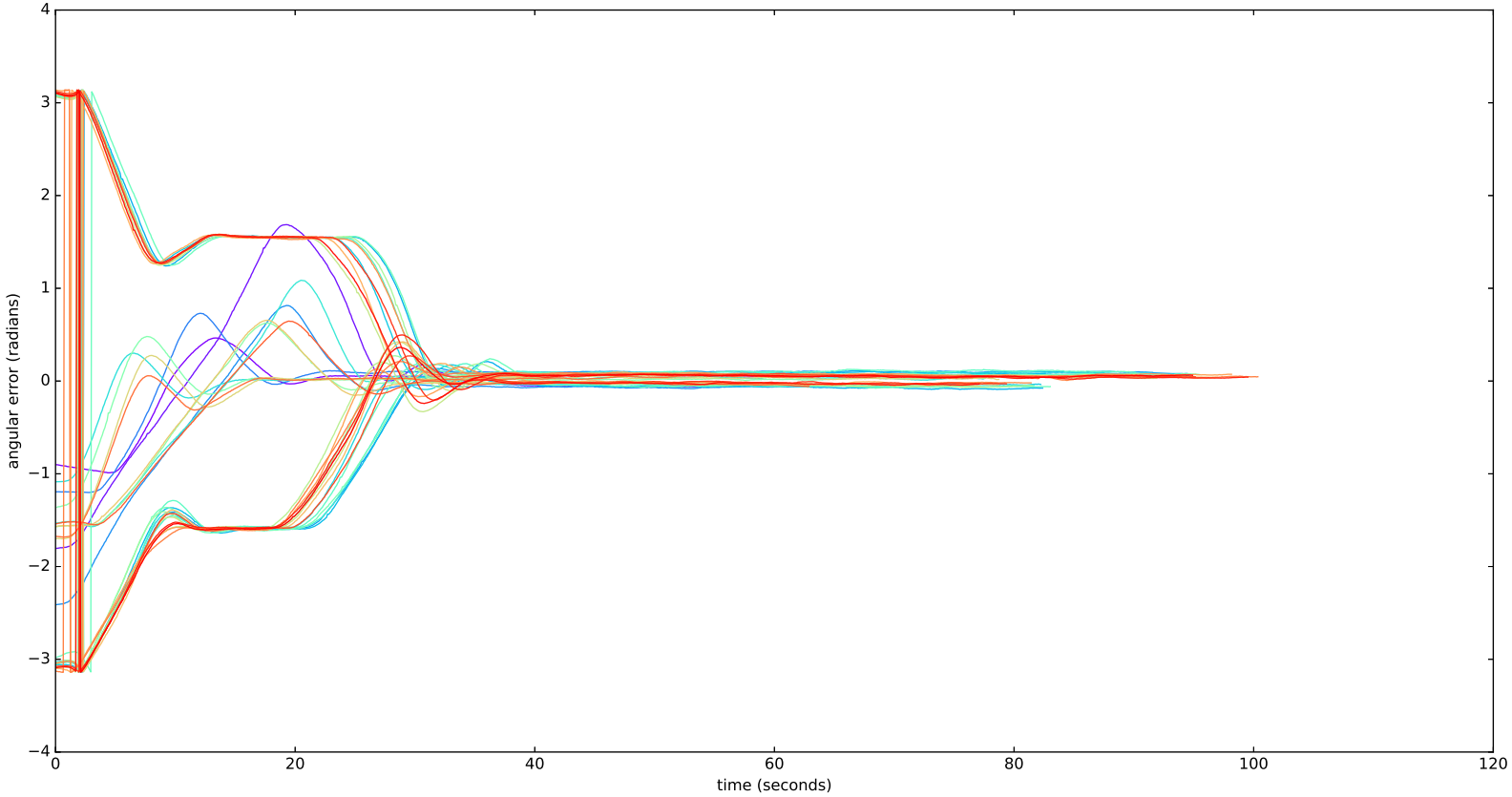


Figure E.2: Vehicle angular error to goal in goal frame generated during experimental runs in West Lakes. X axis shows time since start of goal. Total number of transects, $n = 37$.

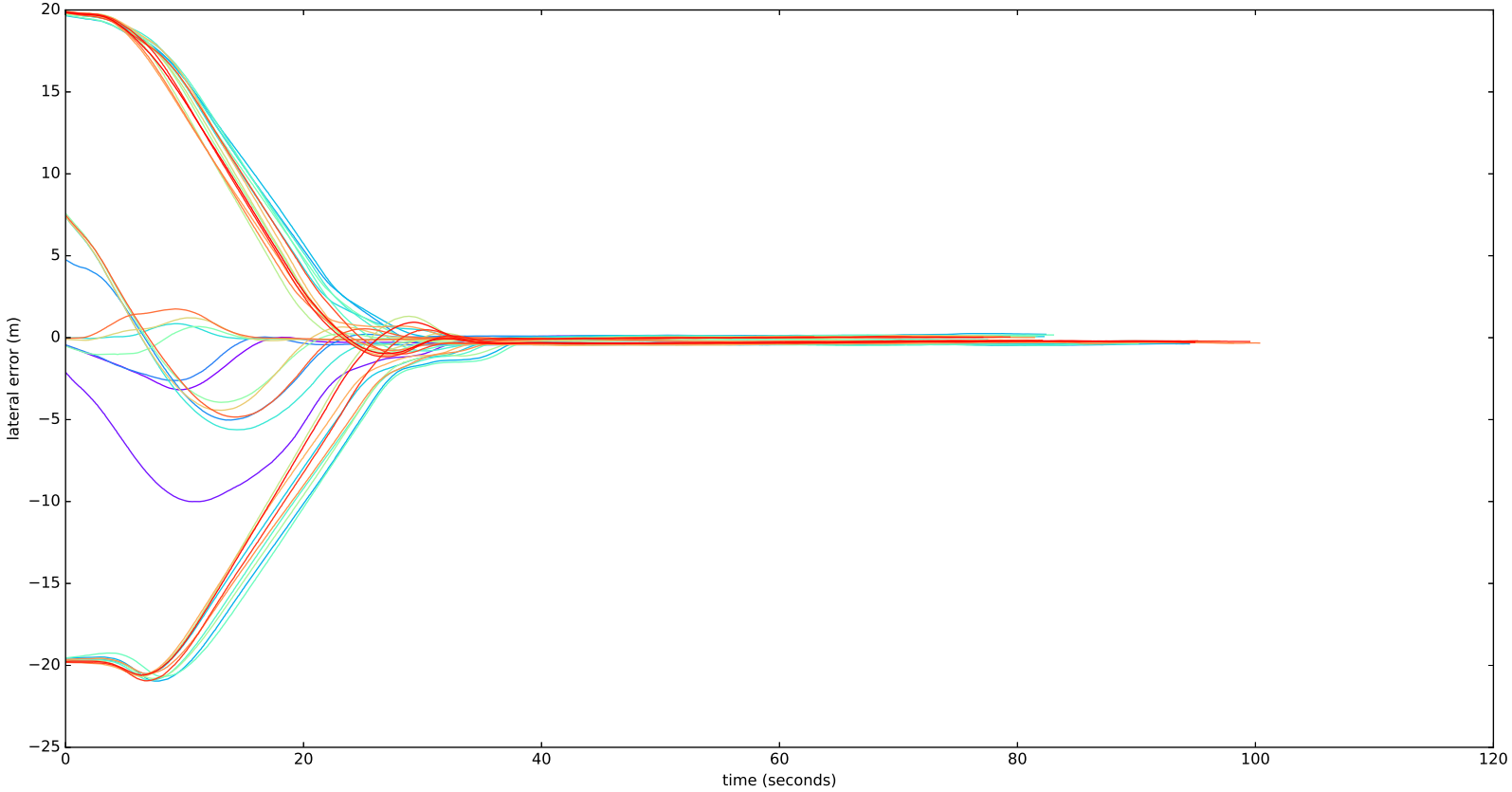


Figure E.3: Vehicle lateral error to goal in goal frame generated during experimental runs in West Lakes. X axis shows time since start of goal. $n = 37$. Total number of transects, $n = 37$.

E.3 Limitations of the Linear Model

The control system developed here has shown to be effective at consistently guiding the vehicle along transects, but the Equation E.10 was required due to its behaviour while off transect. To evaluate the effectiveness of the model at predicting the future state of the vehicle, a program was developed to compare the recorded state of the vehicle x_n with the value $\bar{x_n}$ predicted by combining its earlier state x_{n-10} with the linear model and the control vector $u_{n-10...n}$. If the linear model was correct, then it was expected that $\bar{x_n} = x_n$. The result was generated and converted to a scatterplot which can be seen in Figure E.4.

Notably, Figure E.4 shows an approximately linear relationship between estimated vs actual rate for low predicated rates, but predicted rates over $0.15 rad/s$ did not produce a reliable linear relationship. As such this model is likely only valid for a limited operating range. Use of the predictive controller outside of this range will not result in correct guidance of the vehicle.

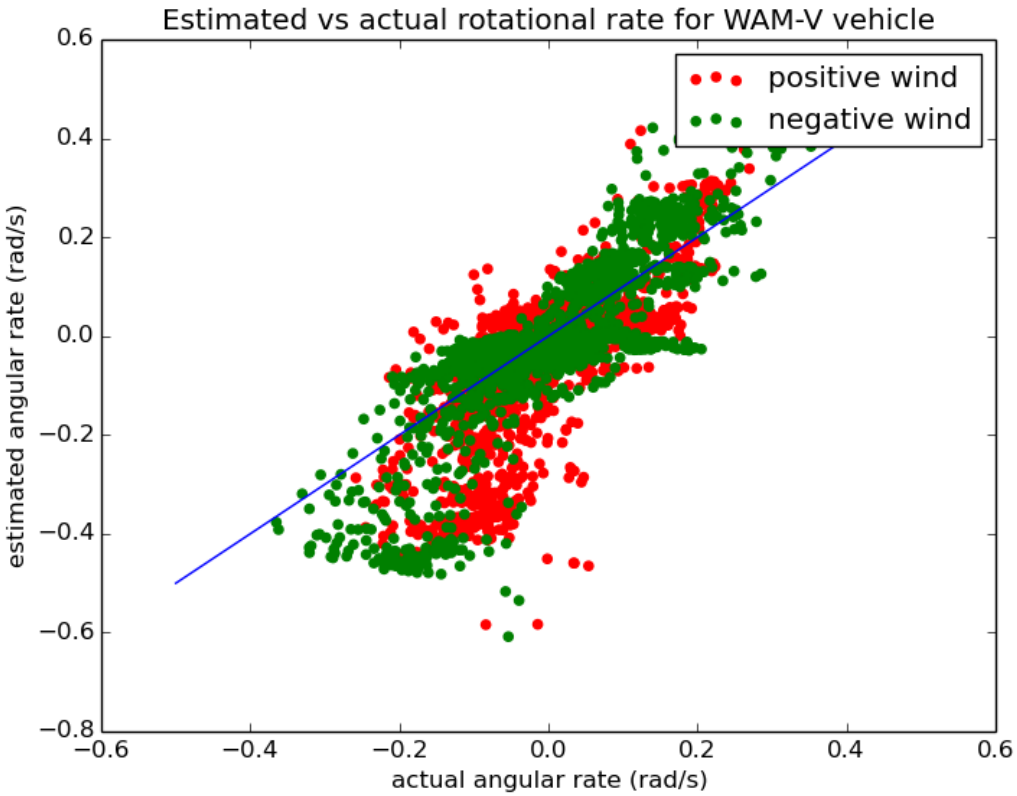


Figure E.4: Scatterplot of predicted vs measured rotational rate during testing. Non-linearity of the graph indicates the model inaccurately predicted the rotational rate.

E.4 Conclusion

The model predictive controller developed for TopCat has demonstrated an ability to control the vehicle on transects even given the $700ms$ latency of control. However, limitations in the linear model prevents the effectiveness of the solution when operating outside the range where the model is valid. Improvements in modelling should allow the system to more accurately control the vehicle in a variety of conditions.

Appendix F

Simulators

F.1 Introduction

Testing of a field robotic system can be long and expensive. To more rapidly develop a system, the use of simulation can reduce development time and reduce testing overhead. A suitable simulation system does not need to provide a fully authentic representation of all vehicle dynamics and systems, but increasing the authenticity and coverage of the system would allow more of the robots systems to be developed under the simulation environment.

The Robotics Operating System (ROS) provides interfaces to a number of simulation systems. These include Turtlesim which is used to demonstrate control of a very simple robotic system using ROS topics, and UWsim a specialised Autonomous Underwater Vehicle (AUV) simulator [IRSLab, 2012]. Neither of these simulators is suitable for the simulation of an Autonomous Surface Vehicle, so a combination of two simulators derived from the Player project were used.

F.2 Stage - a Ground Robot Simulator

The Stage robotic simulator originated as a system to simulate Multi-Robot Systems (MRS), allowing the development and testing of cooperative and swarming type systems [Gerkey et al., 2003]. Stage has been extended to support ROS, allowing integration with the ROS

ecosystem [Gerky, 2015]. The relative simplicity of stage allows it to be easily integrated in test harnesses, while the support for multiple agents in simulation potentially allows the testing of algorithms that can operate in parallel.

Stage represents its environment as a 2.5 dimensional world. Robots have position and heading, but are not capable of roll or pitch, while sensors are limited to a lidar and camera device. An example of a robot in such an environment can be seen in Figure F.1.

Terrain is created by the importation of images showing obstacles and traversable terrain. This allows the simple creation of complex environments.

F.3 Gazebo - a Field Robot Simulator

Stage allowed the development of ground robots that operated under control schemes such as differential drive, but this model cannot be extended to all simulation types. In particular, some types of simulation require the evaluation of the forces acting upon a robot, for example the effect of a suspension system, or the simulation of a multiple degree of freedom rigid body such as an aircraft. These types of simulation are supported by the Gazebo simulator

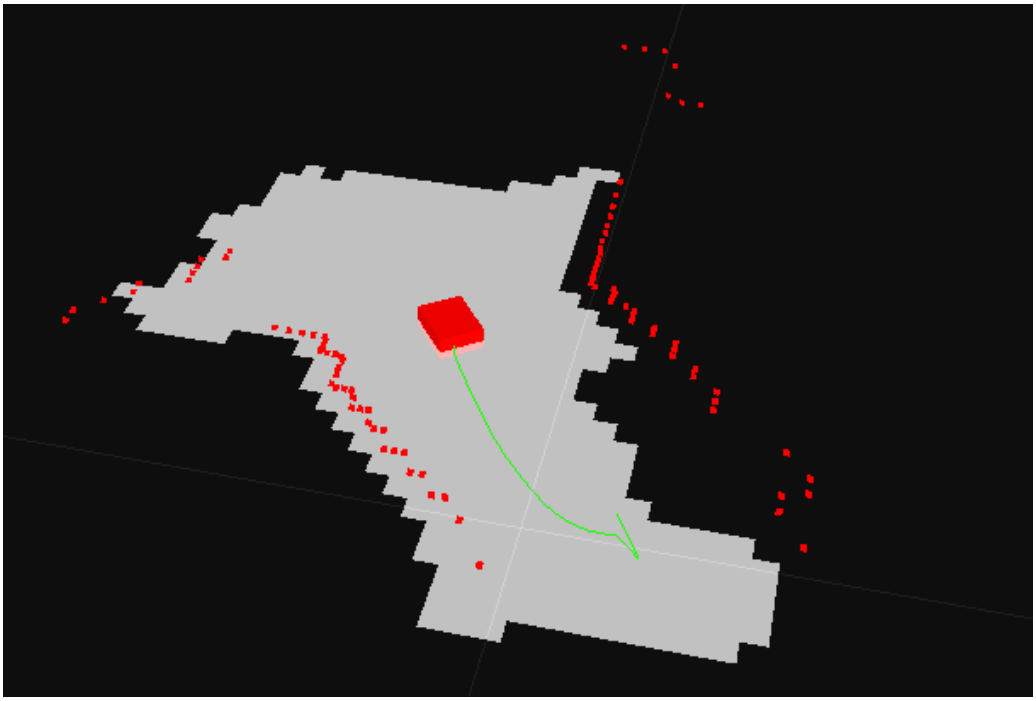


Figure F.1: Stage robot travelling on a small map segment. The red box represents the robot, while red dots are lidar returns. The vehicle's planned trajectory is shown by the green line.

[Koenig, 2004].

Gazebo is notable for the wide range of systems that it can simulate, ranging from the Hector quadrotor developed at Technische Universität Darmstadt [Meyer and Kohlbrecher, 2012], to the Atlas robots used by the Defence Advanced Research Projects Agency (DARPA) Humanoid Challenge [DARPA, 2012]. This flexibility is encouraged by the implementation of a plugin based system allowing user code to be loaded into the simulation. This architecture allows the addition of sensors and behaviours without the simulator itself requiring modification.

F.3.1 Vehicle Dynamics

Gazebo originally supported the Open Dynamics Engine (ODE), a physical simulation engine that supported both rigid body dynamics and collision detection [Smith, Russell, 2007]. This physics engine does not support hydrodynamic simulation, but an earlier project demonstrated that a plug-in could be developed that simulated an AUV using the equations of motion of a Remus 100 [Wheare, 2011]. This approach was extended to develop support for the Wave Adaptive Modular Vessel (WAM-V) based TopCat vehicle.

Vehicle performance data was developed from performance data gathered by students at the Australian Maritime College, including tow tank testing of the WAM-V Hull and the force generated by the Torqeedo motors [Keane et al., 2016].

Using this information, a fit was generated for the drag force on a single hull. This information was encoded into a Gazebo plug-in and a simulation was created. Additional plugins were created to simulate the force due to the motors, while existing plugins and sensors were used to simulate the lidar and cameras installed on the vehicle. This environment was used to both develop the higher level planning system used by TopCat and perform dry runs of the planned missions.

The visual simulation of TopCat is performed using a set of meshes derived from Computer Aided Design (CAD) models developed within the University. As shown in Figure F.2, simplified meshes are used for the representation of the vehicle in physical simulation.

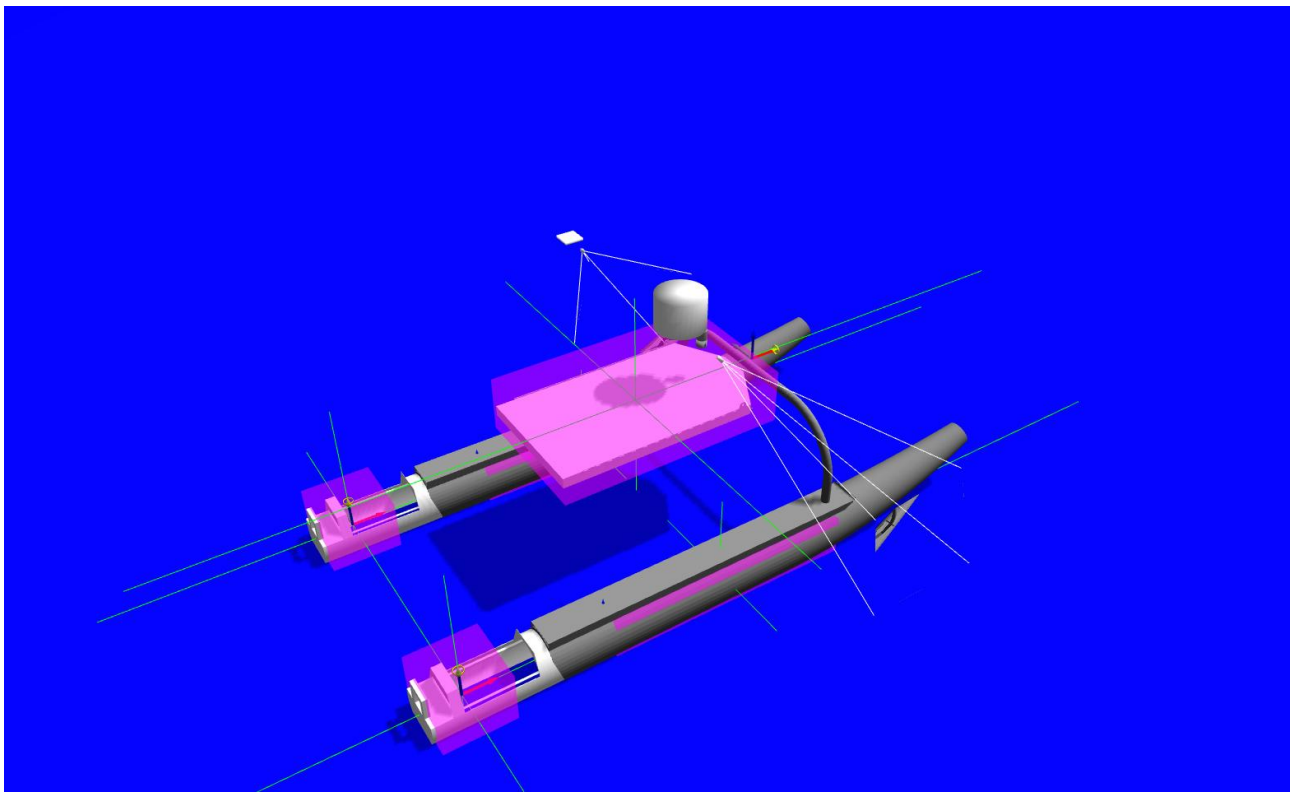


Figure F.2: Gazebo simulation of TopCat vehicle showing visual meshes (grey) and collision bodies (pink). Axes represent hinge points.

F.4 Experimental Validation of Vehicle Dynamics

Since the Gazebo based simulation system was developed to emulate the systems and dynamics of the TopCat vehicle, one method of demonstrating that the simulation resembles the real-world system is to repeat an experiment and analyse the resulting data with the same tools. A simulation of West Lakes was thus used with the task shown in Appendix A. A simulation was started, the run transect script was executed three times, and the data logged for later review.

The vehicle track was recorded and can be seen in Figure F.3. Using the same analysis script as Section E.2, graphs of the vehicle angular and lateral errors were created as shown in Figures F.4 and F.5. While differing in detail, these graphs showed broad similarity to the behaviour of the vehicle as shown in Figures E.2 and E.3.

Figure F.3: Visualisation of simulated vehicle track executed in Gazebo. Map data is ©OpenStreetMap contributors.

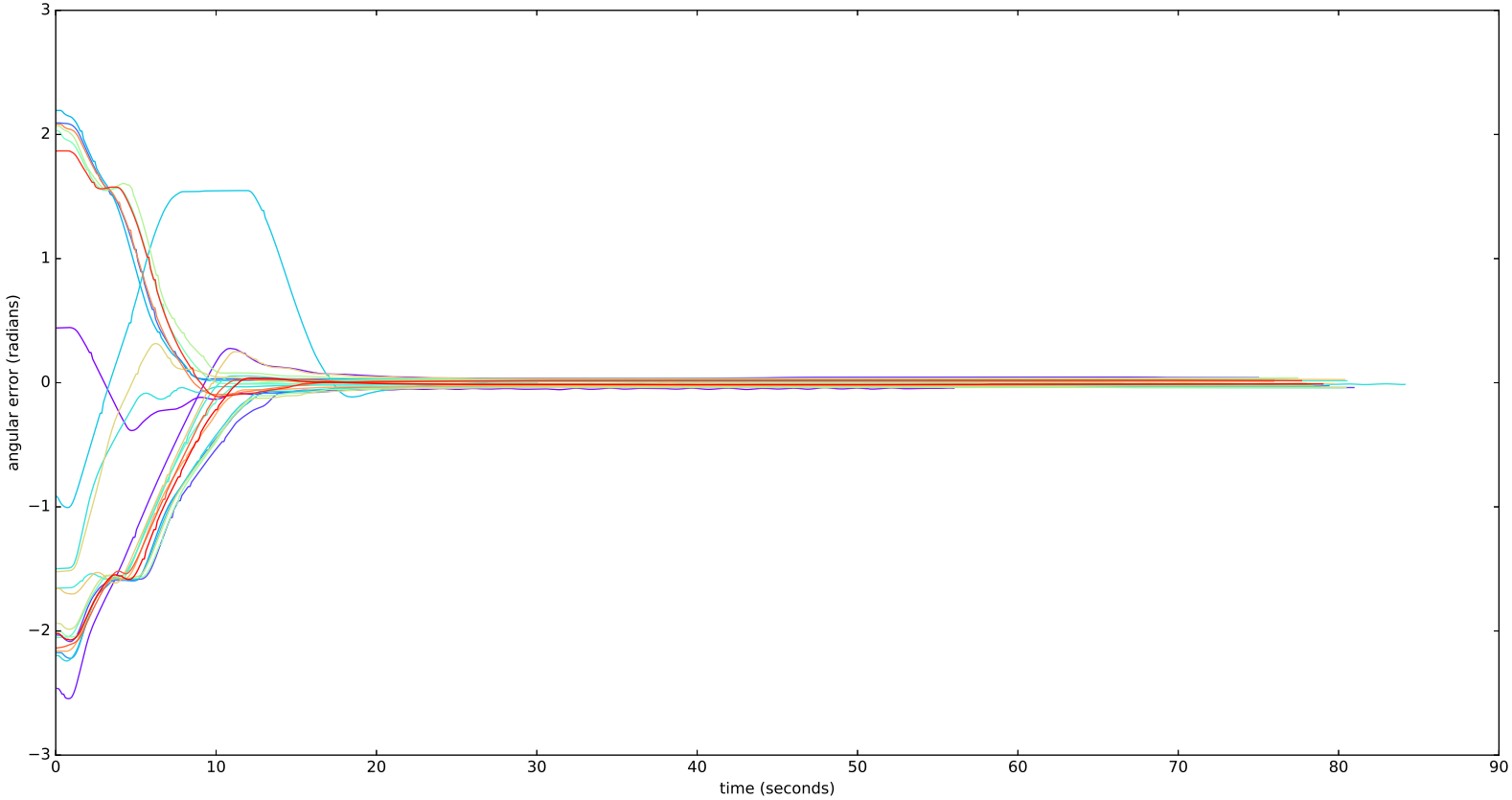


Figure F.4: Vehicle angular error to goal in goal frame. Total number of transects, $n = 22$.

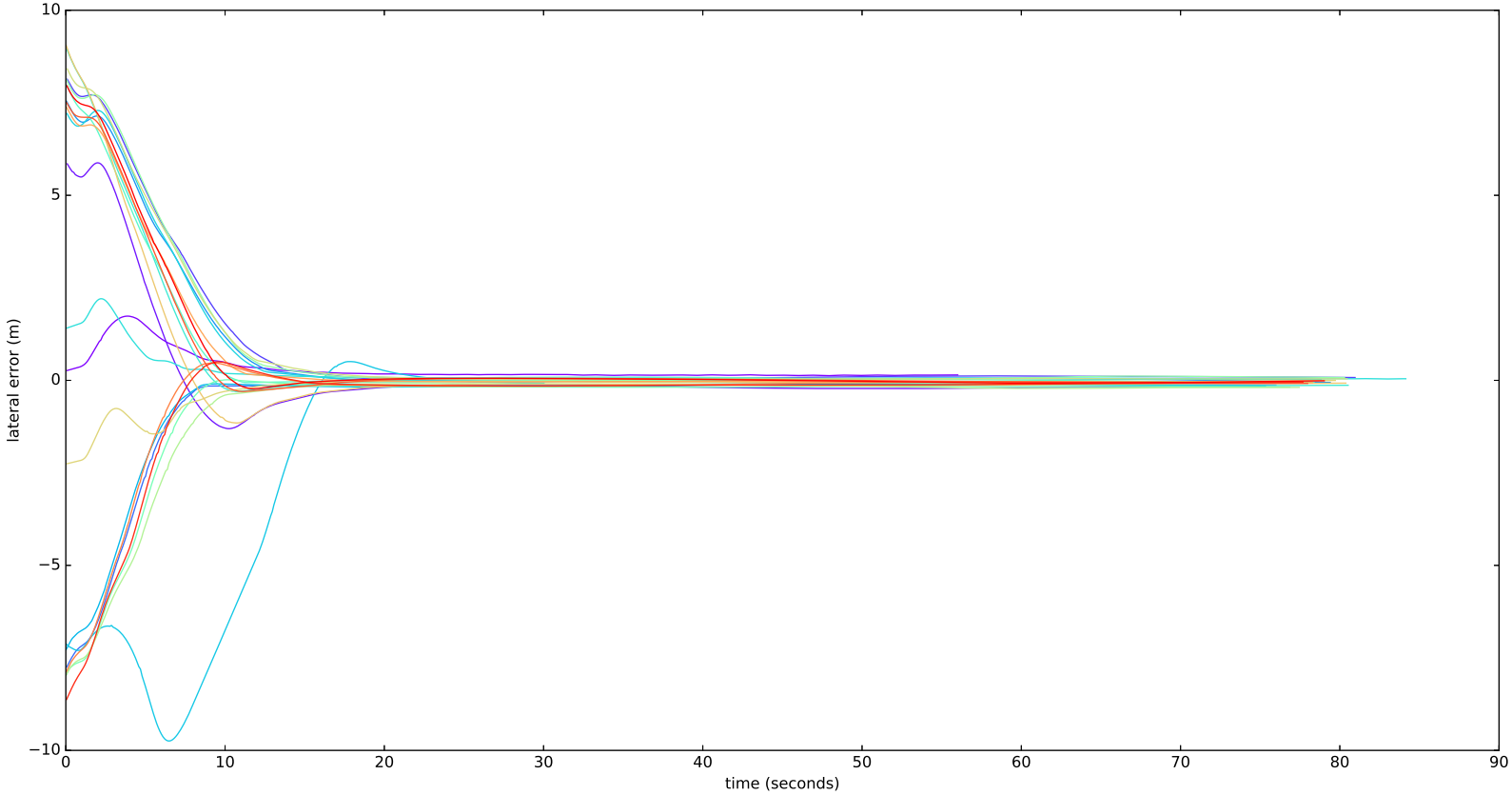


Figure F.5: Vehicle lateral error to goal in goal frame. Total number of transects, $n = 22$.

F.5 Conclusion

This chapter has covered systems used for the simulation of robotics in this thesis. While no simulation system can replace gathering data from field experiments, this appendix has shown that simulation can provide the ability to develop software and algorithms such that they are ready for field testing.

Appendix G

Spatial Data for Field Robotics

The modern world is increasingly networked, with the ubiquity of internet connections and mobile devices allowing the capture, storage and retrieval of data. Storage of geographic information adds further challenges, while the world can be treated as locally flat, on a sufficiently large scale its underlying geometry as an oblate spheriod is required. A solution to this is the use of *geodetic co-ordinate systems*, treating positions as sets of polar co-ordinates. While another method is to specify locally flat planes or *projections* that can be worked in.

This appendix will provide a brief overview of the GIS systems and data used in this thesis. Two external data sources were used, and additional spatial data was generated from the sensors on-board of the TopCat Autonomous Surface Vessel (ASV)

G.1 Manipulation of Spatial Data

A database that is designed to handle geodetic or projected co-ordinates is a Geographic Information System (GIS). GIS are becomming a rich source of data that could be used for planning of field robotic missions. Accessing these data is simplified by the existance of a number of open source projects such as Quantum GIS (QGIS) [QGIS Development Team., 2018], and the Geospatial Data Abstraction Library (GDAL) which allowed the viewing, creation and manipulation of spatial datasets [GDAL Development Team, 201x].

G.2 OpenStreetMap

OpenStreetMap began is a project that emphasises the creation of freely available spatial data made by individual contributors. As a collaborative system, the licensing is also more liberal, allowing usage of the data as long as the contributors copyrights are respected [Open Street Map, 2015c]. An offshoot of this project, the OpenSeaMap, is developing a similar database of maritime charts [Open Street Map, 2015a]. Both projects are in use for the provision of map data, with many applications developed for platforms including mobile phones.

An adapter to extract spatial data from OpenStreetMap has been developed for ROS to allow the use of spatial data[O'Quin, 2015]. An experimental system for route planning is under development, but does not yet appear to be ready for use [O'Quin, 2012].

Unlike other data providers, OpenStreetMap data is primarily crowd-sourced and feature based. Despite lacking overhead or satellite imagery, data from OpenStreetMap was used to provided spatial context and backing maps for both the development of mission plans and the analysis of post-run data.

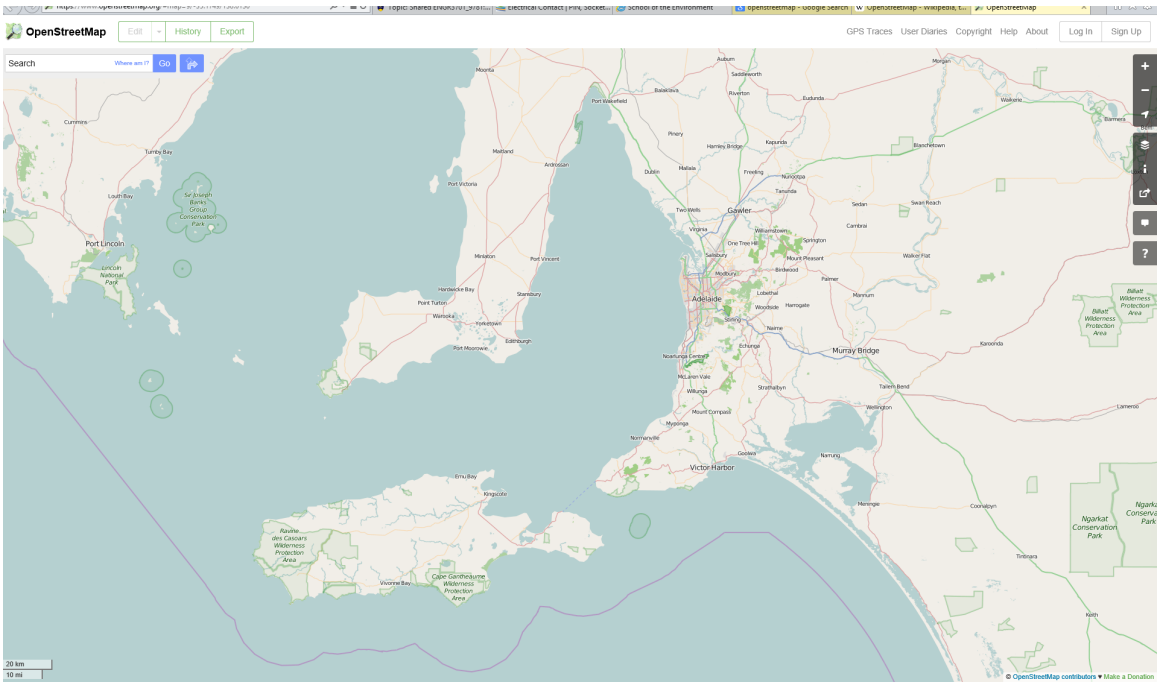


Figure G.1: An image of the OpenStreetMap webpage showing South Australia. Data is ©OpenStreetMap contributors.

G.3 National Oceanic and Atmospheric Administration

The National Oceanic and Atmospheric Administration publishes spatial information under liberal licenses including both projected data, such as the Apra harbour bathymetry used in Section 3.2, and geodetic data such as the Electronic Navigation Charts (ENCs) detailing the safe approach paths to American coasts and harbours [NOAA Office of Coast Survey, 2015]. This data has been useful in the exploration of spatial data available to maritime systems, and as a source of data for the testing of algorithms.

G.4 Map Generation for TopCat

In addition to external data sources, data captured during testing has been used to generate spatial products that were later used in testing and demonstration of the TopCat platform. TopCat is equipped with a Navico 4G maritime Radio Direction and Ranging (RADAR) system. This unit is normally used for maritime navigation, but with the use of TopCats navigation state estimate, the data can be used for mapping.

During an experiment on West Lakes, RADAR data was captured using the Robotic Operating Systems (ROS) bagfile format, and then later played back and converted to a projected map by averaging the intensity of all RADAR returns for each grid cell. This map data was then converted into a spatial file using the Geospatial Data Abstraction Layer (GDAL) system. The resultant data could then be used with a GIS program for display with to other georeferenced data.

As seen in Figure 5.8, this method has allowed mapping of surface and waterborne features of the robots environment. Examination of the detail of this image in Figure G.2 shows that the navigation RADAR is capable of resolving features such as jetties and trees.

This image was produced by combining information transformed into a geo-referenced image using GDAL and then combining this with a backing map using the Quatum GIS package (QGIS) [QGIS Development Team., 2018].

Figure G.2: RAdio Direction And Ranging (RADAR) map of the east branch of West Lakes rendered using the QGIS spatial package. White areas are accumulated RADAR returns. Backing map imagery ©2007 Aerometrex.

G.5 Conclusion

This chapter has covered the spatial data handling that is has been used in this thesis. These packages and data sources have been useful both for generating data for the planning systems as well as visualising the resulting plans and vehicle tracks.

Bibliography

[SRI, 2015] (2015). Shakey the robot. http://www.sri.com/work/timeline-innovation/timeline.php?timeline=computing-digital#!&innovation=shakey-the-robot. Accessed: 2015-06-26.

[Blu, 2017] (2017). Bluefin sandshark autonomous underwater vehicle (AUV). https://gdmissionsystems.com/bluefinrobotics/vehicles-batteries-and-services/bluefin-sandshark.

[Acar and Choset, 2002] Acar, E. U. and Choset, H. (2002). Sensor-based coverage of unknown environments: Incremental construction of morse decompositions. *The International Journal of Robotics Research*, 21(4):345–366.

[Agrawal and Dolan, 2015] Agrawal, P. and Dolan, J. M. (2015). COLREGS-compliant target following for an unmanned surface vehicle in dynamic environments. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1065–1070. IEEE.

[Aleotti and Caselli, 2011] Aleotti, J. and Caselli, S. (2011). Part-based robot grasp planning from human demonstration. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4554–4560. IEEE.

[Allen et al., 1997] Allen, B., Stokey, R., Austin, T., Forrester, N., Goldsborough, R., Purcell, M., and von Alt, C. (1997). REMUS: a small, low cost AUV; system description, field trials and performance results. In *OCEANS '97. MTS/IEEE Conference Proceedings*, volume 2, pages 994–1000 vol.2.

[Amenta et al., 2001] Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust,

unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19((2-3)):127–153.

[Anderson and Crowell, 2005] Anderson, B. and Crowell, J. (2005). Workhorse AUV–a cost-sensible new Autonomous Underwater Vehicle for Surveys/Soundings, Search & Rescue, and Research. In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 1–6. IEEE.

[Anderson, 2014] Anderson, M. (2014). Model-based control and control allocation system for a wave adaptive modular vessel.

[Antiga et al., 2008] Antiga, L., Piccinelli, M., Botti, L., Ene-Iordache, B., Remuzzi, A., and Steinman, D. A. (2008). An image-based modeling framework for patient-specific computational hemodynamics. *Medical & biological engineering & computing*, 46(11):1097–1112.

[Antiga and Steinman, 2004] Antiga, L. and Steinman, D. A. (2004). Robust and objective decomposition and mapping of bifurcating vessels. *Medical Imaging, IEEE Transactions on*, 23(6):704–713.

[Atkeson et al., 2015] Atkeson, C., Babu, B., Banerjee, N., Berenson, D., Bove, C., Cui, X., DeDonato, M., Du, R., Feng, S., Franklin, P., et al. (2015). No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 623–630. IEEE.

[Barreiro et al., 2012] Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., et al. (2012). Europa: A platform for ai planning, scheduling, constraint programming, and optimization. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)–The 4th International Competition on Knowledge Engineering for Planning and Scheduling*.

[Barrett and Weld, 1994] Barrett, A. and Weld, D. S. (1994). Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112.

[Beazley, 2016] Beazley, D. (2016). Ply.

[Belouaer et al., 2010] Belouaer, L., Bouzid, M., and Mouaddib, A. (2010). Ontology based spatial planning for human-robot interaction. *17th International Symposium on Temporal*

Representation and Reasoning.

[Belouaer et al., 2012] Belouaer, L., Bouzid, M., and Mouaddib, A.-I. (2012). Spatial planning. In *Journées Francophones sur la planification, la décision et l'apprentissage pour le contrôle des systémes-JFPDA 2012*.

[Belta et al., 2007] Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. J. (2007). Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 14(1):61–70.

[Benjamin, 2004] Benjamin, M. R. (2004). The interval programming model for multi-objective decision making.

[Benjamin et al., 2004] Benjamin, M. R., Curcio, J. A., et al. (2004). Colregs-based navigation of autonomous marine vehicles. *Proceedings of Autonomous Underwater Vehicles*, pages 32–39.

[Benjamin et al., 2006] Benjamin, M. R., Leonard, J. J., Curcio, J. A., and Newman, P. M. (2006). A method for protocol-based collision avoidance between autonomous marine surface craft. *Journal of Field Robotics*, 23(5):333–346.

[Bernardini et al., 2013] Bernardini, S., Fox, M., Long, D., and Bookless, J. (2013). Autonomous search and tracking via temporal planning.

[Bhattacharya et al., 2012] Bhattacharya, S., Likhachev, M., and Kumar, V. (2012). Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290.

[Blue Robotics Inc, 2017] Blue Robotics Inc (2017). BlueROV2. `http://docs.bluerobotics.com/brov2/`.

[Bohren and Cousins, 2010] Bohren, J. and Cousins, S. (2010). The smach high-level executive.

[Boissonnat et al., 2000] Boissonnat, J.-D., Devillers, O., Teillaud, M., and Yvinec, M. (2000). Triangulations in cgal. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 11–18. ACM.

[Brassington et al., 2007] Brassington, G. G., Pugh, T., Spillman, C., Schulz, E., Beggs, H., Schiller, A., and Oke, P. R. (2007). Bluelink development of operational oceanography and servicing in australia. *Journal of Research and Practice in Information Technology*, 39(2):151–164.

[Brooks, 1990] Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, (6):3–15.

[Brooks, 1991a] Brooks, R. A. (1991a). Intelligence without reason. *Proceedings of 12th Int. Joint Conf. on Artificial Intelligence Sydney, Australia*, pages 569–595.

[Brooks, 1991b] Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159.

[Bruck et al., 2007] Bruck, J., Gao, J., and Jiang, A. (2007). Map: medial axis based geometric routing in sensor networks. *Wirel. Netw.*, 13(6):835–853.

[Brunskill et al., 2007] Brunskill, E., Kollar, T., and Roy, N. (2007). Topological mapping using spectral clustering and classification. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3491–3496. IEEE.

[Bylander, 1994] Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1):165–204.

[Camilli et al., 2004] Camilli, R., Bingham, B., Jakuba, M., Singh, H., and Whelan, J. (2004). Integrating in-situ chemical sampling with AUV control systems. In *OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04*, volume 1, pages 101–109. IEEE.

[Camilli et al., 2010] Camilli, R., Reddy, C. M., Yoerger, D. R., Van Mooy, B. A. S., Jakuba, M. V., Kinsey, J. C., McIntyre, C. P., Sylva, S. P., and Maloney, J. V. (2010). Tracking hydrocarbon plume transport and biodegradation at deepwater horizon. *Science*, 330(6001):201–204.

[Campbell, 2004] Campbell, D. (2004). Events and sightings. *Annals of the History of Computing, IEEE*, 26(2):84–85.

[Caress et al., 2008] Caress, D. W., Thomas, H., Kirkwood, W. J., McEwen, R., Henthorn, R., Clague, D. A., Paull, C. K., Paduan, J., and Maier, K. L. (2008). High-resolution multibeam, sidescan, and subbottom surveys using the MBARI AUV D. Allan B. *Marine habitat mapping technology for Alaska*, pages 47–69.

[Carton, 2001] Carton, X. (2001). Hydrodynamical modeling of oceanic vortices. *Surveys in Geophysics*, 22(3):179–263.

[Cashmore et al., 2015] Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., and Carreras, M. (2015). ROSPlan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, Jerusalem, Israel.

[Choset and Burdick, 1995] Choset, H. and Burdick, J. (1995). Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In *IEEE International Conference on Robotics and Automation*, pages 1643–1643. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE).

[Choset and Nagatani, 2001] Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization. *Robotics and Automation, IEEE Transactions on*, 17(2):125–137.

[Christofides, 1976] Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

[Cline et al., 2009] Cline, D. E., Edgington, D. R., Smith, K. L., Vardaro, M. F., Kuhnz, L., and Ellena, J. A. (2009). An automated event detection and classification system for abyssal time-series images of station m, ne pacific. *Monterey Bay Aquarium Research Institute*.

[Coles et al., 2011] Coles, A., Coles, A., Fox, M., and Long, D. (2011). Popf2: a forward-chaining partial order planner. *The 2011 International Planning Competition*, page 65.

[Collar and McPhail, 1995] Collar, P. and McPhail, S. (1995). Autosub: an autonomous unmanned submersible for ocean data collection. *Electronics & communication engineering journal*, 7(3):105–114.

[Commonwealth of Australia, 2018] Commonwealth of Australia, B. (2018). Sea temperatures and currents. http://www.bom.gov.au/oceanography/forecasts/.

[Cornea, 2005] Cornea, N. D. (2005). Computing hierarchical curve-skeletons of 3d objects.

[Cornea et al., 2006] Cornea, N. D., Silver, D., and Min, P. (2006). Curve-skeleton properties, applications and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, Jun 2006.

[Couprie and Bertrand, 2009] Couprie, M. and Bertrand, G. (2009). New characterizations of simple points in 2d, 3d, and 4d discrete spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):637–648.

[Cousins, 2010] Cousins, S. (2010). ROS on the pr2 [ros topics]. *Robotics & Automation Magazine, IEEE*, 17(3):23–25.

[Curcio et al., 2005] Curcio, J., Leonard, J., and Patrikalakis, A. (2005). Scout-a low cost autonomous surface platform for research in cooperative autonomy. In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 725–729. IEEE.

[da Silva et al., 2012] da Silva, G. M., Mousavi, S. M. S., and Jose, F. (2012). Wave-driven sediment transport and beach-dune dynamics in a headland bay beach. *Marine Geology*, 323-325:29 – 46.

[DARPA, 2012] DARPA (2012). Darpa robotics challenge (drc).

[Das et al., 2011] Das, J., Maughan, T., McCann, M., Godin, M., OReilly, T., Messié, M., F, F. B., Gomes, K., F, F. P., Bellingham, J., Sukhatme, G., and Rajan, K. (2011). Towards mixed-initiative, multi-robot field experiments: Design, deployment, and lessons learned. *IROS 2011*.

[De Cubber et al., 2013] De Cubber, G., Doroftei, D., Serrano, D., Chintamani, K., Sabino, R., and Ourevitch, S. (2013). The eu-icarus project: developing assistive robotic tools for search and rescue operations. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE international symposium on*, pages 1–4. IEEE.

[Dekker et al., 2005] Dekker, A. G., Brando, V. E., and Anstee, J. M. (2005). Retrospective seagrass change detection in a shallow coastal tidal australian lake. *Remote Sensing of Environment*, 97(4):415–433.

[DeMarco et al., 2011] DeMarco, K., West, M. E., and Collins, T. R. (2011). An implementation of ROS on the Yellowfin autonomous underwater vehicle (AUV). In *OCEANS 2011*, pages 1–7. IEEE.

[Do et al., 2009] Do, M., Helmert, M., and Refanidis, I. (2009). Modeling action costs in ipc-2008.

[Doraiswamy and Natarajan, 2009] Doraiswamy, H. and Natarajan, V. (2009). Efficient algorithms for computing reeb graphs. *Computational Geometry*, 42(6-7):606–616.

[Dornhege et al., 2012] Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2012). Semantic attachments for domain-independent planning systems. In *Towards Service Robots for Everyday Environments*, pages 99–115. Springer.

[Dornhege et al., 2009] Dornhege, C., Gissler, M., Teschner, M., and Nebel, B. (2009). Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1–6. IEEE.

[Duarte, 1991] Duarte, C. M. (1991). Seagrass depth limits. *Aquatic Botany*, 40(4):363–377.

[Englot and Hover, 2013] Englot, B. and Hover, F. S. (2013). Three-dimensional coverage planning for an underwater inspection robot. *The International Journal of Robotics Research*, 32(9-10):1048–1073.

[Ennex Corporation, 2015] Ennex Corporation (2015). The stl format, standard data format for fabbers. `http://www.ennex.com/~fabbers/StL.asp`. Accessed: 2015-03-12.

[Environmental Systems Research Institute, 2015] Environmental Systems Research Institute (2015). Arcgis. `http://www.esri.com/software/arcgis`.

[Evans et al., 2003] Evans, J., Petillot, Y., Redmond, P., Wilson, M., and Lane, D. (2003). Autotracker: Real-time architecture for pipeline and cable tracking on AUVs. *Ocean Systems Laboratory, Heriot-Watt University, Edinburgh, EH14 4AS, SCOTLAND, UK*.

[Eyerich et al., 2012] Eyerich, P., Mattmüller, R., and Röger, G. (2012). Using the context-enhanced additive heuristic for temporal and numeric planning. In *Towards Service Robots for Everyday Environments*, pages 49–64. Springer.

[Fawcett et al., 2011] Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., and Seipp, J. (2011). Fd-autotune: Domain-specific configuration using fast downward. *Proc. of ICAPS-PAL*, 2011(8).

[Field et al., 2014] Field, T., Leibs, J., and Bowman, J. (2014). Rosbag.

[Fikes and Nilsson, 1972] Fikes, R. E. and Nilsson, N. J. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). PDDL2.1 : An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124.

[Fox, 1994] Fox, M. S. (1994). Isis: a retrospective. *Intelligent scheduling*, 1:3–28.

[Frank et al., 2016] Frank, D., Gray, A., Allen, K., Bianchi, T., Cohen, K., Dugger, D., Easterling, J., Griessler, M., Hyman, S., Langford, M., Leyva, R., Murphy, L., Nezvadovitz, J., Olive, A., Peterson, B., Soto, D., Voight, F., Volya, D., Williams, T., Schwartz, E., Crane, C., Hill, I., and Ridgeway, S. (2016). University of florida: Team navigator ams.

[Free Software Foundation, Inc, 2018] Free Software Foundation, Inc (2018). Gnu bash. `https://www.gnu.org/software/bash/`.

[Frey and George, 2000] Frey, P. J. and George, P. L. (2000). *Mesh generation: application to finite elements*. Wiley Online Library. `https://onlinelibrary.wiley.com/doi/book/10.1002/9780470611166`.

[Garcia and Gonzalez de Santos, 2004] Garcia, E. and Gonzalez de Santos, P. (2004). Mobile-robot navigation with complete coverage of unstructured environments. *Robotics and Autonomous Systems*, 46(4):195–204.

[GDAL Development Team, 201x] GDAL Development Team (201x). *GDAL - Geospatial Data Abstraction Library, Version 1.11.2*. Open Source Geospatial Foundation.

[Geoscience Australia, 2017] Geoscience Australia (2017). Geosciences Australia. http://www.ga.gov.au/.

[Gerkey et al., 2003] Gerkey, B., Vaughan, R. T., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323.

[Gerky, 2015] Gerky, B. (2015). stage_ros. http://wiki.ros.org/stage_ros.

[GRASS Development Team, 2015] GRASS Development Team (2015). Grass gis. http://grass.osgeo.org/.

[Gray and Schwartz, 2016] Gray, A. and Schwartz, E. (2016). Anglerfish: An ASV controlled ROV. *Proceedings of the 29th Florida Conference on Recent Advances in Robotics*.

[Hart et al., 1972] Hart, P. E., Fikes, R. E., Garvey, T. D., Nilsson, N. J., Nitzan, D., Tenenbaum, J. M., and Wilber, B. M. (1972). Artificial intelligence - research and applications, technical report, project 1530 annual technical report.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

[Haslum et al., 2007] Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012.

[Hatcher, 2002] Hatcher, A. (2002). *Algebraic Topology*. Cambridge University Press.

[Helicon Publishing, 2005] Helicon Publishing (2005). *Hutchinson Pocket Dictionary of Maths*. Helicon Publishing, Abingdon.

[Helmert, 2006] Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246.

[Helmert and Domshlak, 2009] Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?

[Helmert and Domshlak, 2011] Helmert, M. and Domshlak, C. (2011). Lm-cut: Optimal planning with the landmark-cut heuristic. *Seventh International Planning Competition (IPC 2011)*, pages 103–105.

[Helmert et al., 2014] Helmert, M., Röger, G., Karpas, E., Seipp, J., Sievers, S., and Pommerening, F. (2014). Fast downward.

[Hitz et al., 2014] Hitz, G., Gotovos, A., Garneau, M.-É., Pradalier, C., Krause, A., Siegwart, R. Y., et al. (2014). Fully autonomous focused exploration for robotic environmental monitoring. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2658–2664. IEEE.

[Hitz et al., 2012] Hitz, G., Pomerleau, F., Garneau, M. E., Pradalier, C., Posch, T., Pernthaler, J., and Siegwart, R. Y. (2012). Autonomous inland water monitoring: Design and application of a surface vessel. *IEEE Robotics Automation Magazine*, 19(1):62–72.

[Hoffmann, 2000] Hoffmann, J. (2000). Ff-x.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, pages 253–302.

[Homann, 2007] Homann, H. (2007). Implementation of a 3d thinning algorithm. *The Insight Journal*, (July - December).

[Hover et al., 2012] Hover, F. S., Eustice, R. M., Kim, A., Englot, B., Johannsson, H., Kaess, M., and Leonard, J. J. (2012). Advanced perception, navigation and planning for autonomous in-water ship hull inspection. *The International Journal of Robotics Research*, 31(12):1445–1464.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

[HYDROID, 2016] HYDROID (2016). Remus 100 brochure. `https://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/09E7AAF768254B62C1257EC20025BC36/$file/REMUS_100_Brochure.pdf?OpenElement`.

[IPC benchmark authors, 2010] IPC benchmark authors (2010). Fast downward benchmarks. Accessed: 2016-6-6.

[IRSLab, 2012] IRSLab, Jaumel University, C. (2012). Uwsim the underwater simulator.

[Johnson and McGeoch, 1997] Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310.

[Jones et al., 2001] Jones, E., Oliphant, T., Peterson, P., et al. (2001). Scipy: Open source scientific tools for python.

[Karp, 1977] Karp, R. M. (1977). Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. *Mathematics of operations research*, 2(3):209–224.

[Kawatsuma et al., 2012] Kawatsuma, S., Fukushima, M., and Okada, T. (2012). Emergency response by robots to fukushima-daiichi accident: summary and lessons learned. *Industrial Robot: An International Journal*, 39(5):428–435.

[KCL-Planning, 2016] KCL-Planning (2016). ROSPLAN github repository. `https://github.com/KCL-Planning/ROSPlan`. Accessed: 2016-7-15.

[Keane et al., 2016] Keane, J., Duffy, J., Haase, M., Randeni, S., Hubbert, H., Kent, R., Forrest, A., Sammut, K., Lammas, A., Wheare, J., and Battle, D. (2016). Preparing a wave adaptive modular vessel for automation. *Royal Institution of Naval Architects Transactions 2016 Part B2 - International Journal of Small Craft Technology*, 158:113–121.

[Keane, 2016] Keane, R. (2016). 10m Digital Elevation Model. From Flinders University Geographical Information System server.

[Kendall and Stuart, 1967] Kendall, M. G. and Stuart, A. (1967). The advanced theory of statistics, vol. 2, hafner. *New York*.

[Kiraly et al., 2004] Kiraly, A. P., Helferty, J. P., Hoffman, E. A., McLennan, G., and Higgins, W. E. (2004). Three-dimensional path planning for virtual bronchoscopy. *IEEE TRANSACTIONS ON MEDICAL IMAGING*, 23(9):1365–1379.

[Kleiner et al., 2017] Kleiner, A., Baravalle, R., Kolling, A., Pilotti, P., and Munich, M. (2017). A solution to room-by-room coverage for autonomous cleaning robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5346–5352.

[Knoblock et al., 1998] Knoblock, C., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D. E., Sun, Y., and Weld, D. (1998). PDDL - the planning domain definition language version 1.2. *AIPS98 planning committee*, 78(4):1–27.

[Koenig, 2004] Koenig, N. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator 2004 ieee/rsj international conference on intelligent robots and systems (iros) (ieee cat. no.04ch37566). 3:2149.

[Koenig, 2018] Koenig, N. (2018). Gazebo model database. `https://bitbucket.org/osrf/gazebo_models`.

[Kollar et al., 2010] Kollar, T., Tellex, S., Roy, D., and Roy, N. (2010). Toward understanding natural language directions. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 259–266. IEEE.

[Kruger et al., 2006] Kruger, D., Stolkin, R., Blum, A., and Briganti, J. (2006). Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments.

[Kümmerle et al., 2011] Kümmerle, R., Steder, B., Dornhege, C., Kleiner, A., Grisetti, G., and Burgard, W. (2011). Large scale graph-based slam using aerial images as prior information. *Autonomous Robots*, 30(1):25–39.

[LaValle, 1998] LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University. TR 98-11.

[Lee et al., 1994] Lee, T., Kashyap, R. L., and Chu, C. (1994). Building skeleton models via 3-d medial surface/axis thinning algorithms. *CVGIP GRAPHICAL MODELS AND IMAGE PROCESSING*, 56(6):462.

[Lin and Kernighan, 1973] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[Lipovetzky, 2010] Lipovetzky, N. (2010). Visit-all domain. Accessed: 2016-6-6.

[Lipovetzky, 2011] Lipovetzky, N. (2011). Visit-all file repository. Accessed: 2018-4-7.

[Machado et al., 2014] Machado, D., Martins, A., Almeida, J. M., Ferreira, H., Amaral, G., Ferreira, B., Matos, A., and Silva, E. (2014). Water jet based autonomous surface vehicle for coastal waters operations. In *Oceans-St. John's, 2014*, pages 1–8. IEEE.

[Macmillan Publishers Australia, 2018] Macmillan Publishers Australia (2018). Macquarie dictionary - lidar. `https://www.macquariedictionary.com.au/features/word/search/?word=lidar&search_word_type=Dictionary`.

[Marder-Eppstein et al., 2018] Marder-Eppstein, E., Lu, D. V., Hershberger, D., and contradict@gmail.com (2018). *costmap_2d*. ROS.

[Marine Advanced Research Inc., 2015] Marine Advanced Research Inc. (2015). Wave adaptive multipurpose vessel. Accessed: 2015-12-18.

[Maritime RobotX Challenge, 2014] Maritime RobotX Challenge (2014). Maritime RobotX Challenge.

[Maritime RobotX Challenge, 2018] Maritime RobotX Challenge (2018). 2018 maritime robotx challenge task summary.

[Martins et al., 2007] Martins, A., Ferreira, H., Almeida, C., Silva, H., Almeida, J. M., and Silva, E. (2007). Roaz and roaz ii autonomous surface vehicle design and implementation. In *International Lifesaving Congress 2007*.

[Matos et al., 2013] Matos, A., Silva, E., Cruz, N., Alves, J. C., Almeida, D., Pinto, M., Martins, A., Almeida, J., and Machado, D. (2013). Development of an unmanned capsule for large-scale maritime search and rescue. In *2013 OCEANS-San Diego*, pages 1–8. IEEE.

[McCarthy, 1963] McCarthy, J. (1963). Situations, actions, and causal laws. Report, DTIC Document.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - the planning domain definition language.

[McGann et al., 2007] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2007). T-REX: A model-based architecture for AUV control.

[McGill et al., 1978] McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of box plots. *The American Statistician*, 32(1):12–16.

[Meunier and Villermaux, 2003] Meunier, P. and Villermaux, E. (2003). How vortices mix. *Journal of Fluid Mechanics*, 476:213–222.

[Meyer and Kohlbrecher, 2012] Meyer, J. and Kohlbrecher, S. (2012). hector_quadrotor. *http://www.ros.org/wiki/hector_quadrotor*.

[Microsoft, 2014] Microsoft (2014). Depthimageformat enumeration.

[MKFI, 2014] MKFI (2014). Remus 100 merivoimien vuosipäivä 2014 01.jpg. `https://commons.wikimedia.org/wiki/File:REMUS_100_Merivoimien_vuosip%C3%A4iv%C3%A4_2014_01.JPG`.

[Molnar et al., 2010] Molnar, L., Ezekiel, J., Veres, S. M., Lomuscio, A. R., and Pebody, M. (2010). Formal verification of the autosub autonomous underwater vehicle: A case study.

[Muñoz et al., 2016] Muñoz, P., R-Moreno, M. D., and Barrero, D. F. (2016). Unified framework for path-planning and task-planning for autonomous robots. *Robotics and Autonomous Systems*, 82:1–14.

[Murphy, 2014] Murphy, R. R. (2014). *Disaster robotics*. MIT press.

[Newman, 2008] Newman, P. M. (2008). MOOS-mission orientated operating suite. *Massachusetts Institute of Technology, Tech. Rep*, 2299(08).

[Nilsson, 1984] Nilsson, N. J. (1984). Shakey the robot.

[Nilsson et al., 1968] Nilsson, N. J., Rosen, C. A., Raphael, B., Forsen, G., Chaitin, L., and Wahlstrom, S. (1968). Application of intelligent automata to reconnaissance, technical report.

[NOAA Office of Coast Survey, 2015] NOAA Office of Coast Survey (2015). Electronic navigational charts: Noaa enc.

[Office of Naval Research, 2016] Office of Naval Research (2016). 160515-n-po203-194. https://www.flickr.com/photos/usnavyresearch/26996025532.

[Okada and Ueda, 2016] Okada, K. and Ueda, R. (2016). Ff-x.

[Open Source Robotics Foundation, 2014a] Open Source Robotics Foundation (2014a). actionlib. http://wiki.ros.org/actionlib.

[Open Source Robotics Foundation, 2014b] Open Source Robotics Foundation (2014b). Navigation.

[Open Source Robotics Foundation, 2014c] Open Source Robotics Foundation (2014c). Turtlebot. http://www.turtlebot.com/.

[Open Source Robotics Foundation, 2014d] Open Source Robotics Foundation (2014d). Turtlebot_navigation. http://wiki.ros.org/turtlebot_navigation.

[Open Source Robotics Foundation, 2018] Open Source Robotics Foundation (2018). ROS robots. http://wiki.ros.org/robots.

[Open Street Map, 2015a] Open Street Map (2015a). Open sea map. http://www.openseamap.org. Accessed: 2015-03-23.

[Open Street Map, 2015b] Open Street Map (2015b). Open street map. http://wiki.openstreetmap.org/. Accessed: 2015-03-23.

[Open Street Map, 2015c] Open Street Map (2015c). Open street map legal faq. http://wiki.openstreetmap.org/wiki/Legal_FAQ. Accessed: 2015-03-23.

[O'Quin, 2012] O'Quin, J. (2012). route_network. http://wiki.ros.org/route_network?distro=indigo. Accessed: 2015-03-23.

[O'Quin, 2015] O'Quin, J. (2015). osm_cartography. http://wiki.ros.org/osm_cartography. Accessed: 2015-03-23.

[O'Quinn, 2016] O'Quinn, J. (2016). velodyne.

[Oyama et al., 2009] Oyama, A., Konolige, K., Cousins, S., Chitta, S., Conley, K., and Bradski, G. (2009). Come on in, our community is wide open for robotics research. *RSJ*.

[Pacific Islands Benthic Habitat Mapping Center (PIBHMC) et al., 2010] Pacific Islands Benthic Habitat Mapping Center (PIBHMC), Coral Reef Ecosystem Division (CRED), Pacific Islands Fisheries Science Center (PIFSC), National Marine Fisheries Service (NMFS), and National Oceanic and Atmospheric Administration (NOAA) (2010). Gridded multibeam bathymetry of apra harbor, guam u.s. territory. Technical report.

[Palágyi and Kuba, 1999] Palágyi, K. and Kuba, A. (1999). A parallel 3d 12-subiteration thinning algorithm. *Graphical Models and Image Processing*, 61(4):199–221.

[Palomeras et al., 2012] Palomeras, N., El-Fakdi, A., Carreras, M., and Ridao, P. (2012). COLA2: A control architecture for AUVs. *IEEE Journal of Oceanic Engineering*, 37(4):695–716.

[Papadopoulos et al., 2011] Papadopoulos, G., Kurniawati, H., Bin Mohd Shariff, A. S., Wong, L. J., and Patrikalakis, N. M. (2011). 3d-surface reconstruction for partially submerged marine structures using an autonomous surface vehicle. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3551–3557.

[Patterson et al., 2013] Patterson, M. C., Mulligan, A., and Boiteux, F. (2013). Safety and security applications for micro-unmanned surface vessels. In *Oceans-San Diego, 2013*, pages 1–6. IEEE.

[Perko et al., 2016] Perko, E., Rockey, C., Agrawal, R., and Okay, S. d. (2016). xv_11_laser_driver.

[Pêtrès et al., 2007] Pêtrès, C., Pailhas, Y., Patrón, P., Petillot, Y., Evans, J., and Lane, D. (2007). Path planning for autonomous underwater vehicles.

[Phinn et al., 2008] Phinn, S., Roelfsema, C., Dekker, A., Brando, V., and Anstee, J. (2008). Mapping seagrass species, cover and biomass in shallow waters: An assessment of satellite multi-spectral and airborne hyper-spectral imaging systems in moreton bay (australia). *Remote Sensing of Environment*, 112(8):3413–3425.

[Pinto et al., 2014a] Pinto, E., Marques, F., Mendonça, R., Lourenço, A., Santana, P., and Barata, J. (2014a). An autonomous surface-aerial marsupial robotic team for riverine environmental monitoring: Benefiting from coordinated aerial, underwater, and surface level

perception. In *Proc. of the IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE*.

[Pinto et al., 2014b] Pinto, E., Santana, P., Marques, F., Mendonça, R., Lourenço, A., and Barata, J. (2014b). On the design of a robotic system composed of an unmanned surface vehicle and a piggybacked vtol. In *Technological Innovation for Collective Awareness Systems*, pages 193–200. Springer.

[Pivtoraiko and Kelly, 2005] Pivtoraiko, M. and Kelly, A. (2005). Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7.

[Pommerening et al., 2014] Pommerening, F., Röger, G., Helmert, M., and Bonet, B. (2014). Lp-based heuristics for cost-optimal planning. In *ICAPS*.

[Python Software Foundation, 2018] Python Software Foundation (2018). Python programming language.

[QGIS Development Team., 2018] QGIS Development Team. (2018). QGIS geographic information system. `http://qgis.osgeo.org`.

[Quigley et al., 2007] Quigley, M., Berger, E., and Ng, A. Y. (2007). Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37.

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, number 3.2, page 5.

[Rajan et al., 2009] Rajan, K., Py, F., McGann, C., Ryan, J., O'Reilly, T., Maughan, T., and Roman, B. (2009). Onboard adaptive control of AUVs using automated planning and execution.

[Rao, 1993] Rao, N. S. V. (1993). *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*.

[Ren et al., 2008] Ren, T. R., Kwok, N. M., Liu, D. K., and Huang, S. D. (2008). Path planning for a robotic arm sand-blasting system. In *2008 International Conference on Information*

and Automation, pages 1067–1072.

[Requicha and Rossignac, 1992] Requicha, A. A. and Rossignac, J. R. (1992). Solid modeling and beyond. *IEEE Computer Graphics and Applications*, 12(5):31–44.

[Richter and Westphal, 2010] Richter, S. and Westphal, M. (2010). The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177.

[Roelfsema et al., 2015] Roelfsema, C., Lyons, M., Dunbabin, M., Kovacs, E. M., and Phinn, S. (2015). Integrating field survey data with satellite image data to improve shallow water seagrass maps: the role of AUV and snorkeller surveys? *Remote Sensing Letters*, 6(2):135–144.

[Roubeyrie and Crook, 2018] Roubeyrie, L. and Crook, C. (2018). Qgis-contour-plugin.

[Rudnick et al., 2004] Rudnick, D. L., Davis, R. E., Eriksen, C. C., Fratantoni, D. M., and Perry, M. J. (2004). Underwater gliders for ocean research. *Marine Technology Society Journal*, 38(2):73–84.

[Rufat, 2017] Rufat, D. (2017). Python binding to the triangle library.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial intelligence: a modern approach third edition*. Prentice Hall. ISBN: 978-0-13-207148-2.

[Rusu et al., 2010] Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. (2010). Fast 3d recognition and pose using the viewpoint feature histogram.

[Rusu and Cousins, 2011] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

[Ryan et al., 2010] Ryan, J., Johnson, S., Sherman, A., Rajan, K., Py, F., Thomas, H., Harvey, J., Bird, L., Paduan, J., and Vrijenhoek, R. (2010). Mobile autonomous process sampling within coastal ocean observing systems. *Limnology and Oceanography: Methods*, 8(8):394–402.

[Saha et al., 2006] Saha, M., Roughgarden, T., Latombe, J.-C., and Sánchez-Ante, G. (2006). Planning tours of robotic arms among partitioned goals. *The International Journal of Robotics Research*, 25(3):207–223.

[Sammut et al., 2014] Sammut, K., Wheare, J., Lammas, A., Bowyer, R., Anderson, M., Arbon, T., Donnelly, B., Peake, R., Crouch, T., Pivetta, R., Renfrey, J., Wooldridge, T., Stevens, S., Webb, A., Forrest, A., Keane, J., Hubbert, H., Kent, R., and Pathiranachchilage, S. R. (2014). Maritime RobotX journal paper - Flinders University / Australian Maritime College - Team Topcat.

[Sanz et al., 2013] Sanz, P. J., Ridao, P., Oliver, G., Casalino, G., Petillot, Y., Silvestre, C., Melchiorri, C., and Turetta, A. (2013). Trident an european project targeted to increase the autonomy levels for underwater intervention missions. In *Oceans-San Diego, 2013*, pages 1–10. IEEE.

[Schillinger, 2015] Schillinger, P. (2015). An approach for runtime-modifiable behavior control of humanoid rescue robots. Master's thesis, Technische Universität Darmstad.

[scikit-image development team, 2015a] scikit-image development team (2015a). Scikit.image. `http://scikit-image.org/`. Accessed: 2016-8-27.

[scikit-image development team, 2015b] scikit-image development team (2015b). Scikit.morphology.watershed. `http://scikit-image.org/docs/dev/api/skimage.morphology.html#watershed`. Accessed: 2015-10-19.

[Shade and Newman, 2011] Shade, R. and Newman, P. (2011). Choosing where to go: Complete 3d exploration with stereo.

[Shah et al., 2016] Shah, B. C., Švec, P., Bertaska, I. R., Sinisterra, A. J., Klinger, W., von Ellenrieder, K., Dhanak, M., and Gupta, S. K. (2016). Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots*, 40(7):1139–1163.

[Si, 2015] Si, H. (2015). Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)*, 41(2):11.

[Sjöö et al., 2010] Sjöö, K., Aydemir, A., Schlyter, D., and Jensfelt, P. (2010). Topological spatial relations for active visual search.

[Smith et al., 2008] Smith, D. E., Frank, J., and Cushing, W. (2008). The anml language. *Proceedings of ICAPS-08*.

[Smith, Russell, 2007] Smith, Russell (2007). The open dynamics engine. Accessed: 2016-9-14.

[Sotzing et al., 2008] Sotzing, C. C., Johnson, N., and Lane, D. M. (2008). Improving multi-AUV coordination with hierarchical blackboard-based plan representation. In *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, pages 110–117.

[SRI International, 1972] SRI International (1972). Sri shakey with callouts.jpg. `https://commons.wikimedia.org/wiki/File:SRI_Shakey_with_callouts.jpg`. Accessed: 2015-06-26.

[Stentz, 1994] Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. *In Proceedings IEEE International Conference on Robotics and Automation*, May 1994.

[Sud et al., 2007] Sud, A., Andersen, E., Curtis, S., Ming, L., and Manocha, D. (2007). Real-time path planning for virtual agents in dynamic environments. In *Virtual Reality Conference, 2007. VR '07. IEEE*, pages 91–98.

[Sun and Tsung-Ying, 2008] Sun, T. Y. and Tsung-Ying, S. (2008). Optimal UAV flight path planning using skeletonization and particle swarm optimizer. *2008 IEEE Congress on Evolutionary Computation*, page 1183.

[Teledyne Oceanscience, 2015a] Teledyne Oceanscience (2015a). Featured survey: Autonomous hydrographic survey of lake hodges, california usa. `http://www.oceanscience.com/news/detail.aspx?nid=107`. Accessed: 2015-04-28.

[Teledyne Oceanscience, 2015b] Teledyne Oceanscience (2015b). Z-boat 1800 remote hydrographic survey boat. `http://www.oceanscience.com/pdf/Z-Boat%20Spec%20Sheet%202015%20SM.pdf`. Accessed: 2015-04-28.

[Thrun, 1998] Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71.

[Trimble, 2014] Trimble (2014). trimble R10 GNSS system.

[Trimble, 2016] Trimble (2016). Trimble BX982 datasheet.

[Universidad Politechnica de Cartagena, 2014] Universidad Politechnica de Cartagena, University of Porto, U. (2014). Uready4os. `https://gdmissionsystems.com/bluefinrobotics/vehicles-batteries-and-services/bluefin-sandshark`.

[Vaughan, 2008] Vaughan, R. (2008). Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208.

[Von Alt et al., 1994] Von Alt, C., Allen, B., Austin, T., and Stokey, R. (1994). Remote environmental measuring units. In *Autonomous Underwater Vehicle Technology, 1994. AUV'94., Proceedings of the 1994 Symposium on*, pages 13–19. IEEE.

[Waibel et al., 2011] Waibel, M., Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Galvez-Lopez, D., Haussermann, K., Janssen, R., Montiel, J. M. M., Perzylo, A., Schiessle, B., Tenorth, M., Zweigle, O., and van de Molengraft, R. (2011). Roboearth. *Robotics & Automation Magazine, IEEE*, 18(2):69–82.

[Webb et al., 2016] Webb, A., Donnelly, B., Stewart, J., Wheare, J., Kossatz, M., Hutchinson, S., Geyer, S., Crouch, T., Lammas, A., and Sammut, K. (2016). Development and testing of the TopCat autonomous surface vessel for the Maritime RobotX Challenge 2016.

[West et al., 2010] West, M. E., Novitzky, M., Varnell, J. P., Melim, A., Sequin, E., Toler, T. C., Collins, T. R., and Bogle, J. R. (2010). Design and development of the yellowfin uuv for homogenous collaborative missions.

[Wheare, 2011] Wheare, J. (2011). A reactive scheduler for the flinders autonomous underwater vehicle. *Flinders University*.

[Wheare et al., 2018] Wheare, J., Lammas, A., and Sammut, K. (2018). Towards the generation of autonomous vehicle plans for operation in confined areas.

[Widditsch, 1973] Widditsch, H. (1973). Spurv-the first decade. Technical report, DTIC Document.

[Wingbermuehle, 2010] Wingbermuehle, J. (2010). Maze generator in python.

[Wong and MacDonald, 2003] Wong, S. C. and MacDonald, B. A. (2003). A topological coverage algorithm for mobile robots. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1685–1690. IEEE.

[Wurm et al., 2010] Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems.

[Wyrobek et al., 2008] Wyrobek, K. A., Berger, E. H., Van der Loos, H., and Salisbury, J. K. (2008). Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2165–2170. IEEE.

[Yahja et al., 1998] Yahja, A., Stentz, A., Singh, S., and Brumitt, B. L. (1998). Framed-quadtree path planning for mobile robots operating in sparse environments. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 650–655. IEEE.

[Yamauchi, 2004] Yamauchi, B. M. (2004). Packbot: A versatile platform for military robotics. In *Unmanned Ground Vehicle Technology VI*, volume 5422, pages 228–238. International Society for Optics and Photonics.

[Younes and Littman, 2004] Younes, H. L. and Littman, M. L. (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-167*.

[Zeng et al., 2012] Zeng, Z., Lammas, A., Sammut, K., and He, F. (2012). Optimal path planning based on annular space decomposition for auvs operating in a variable environment. In *Autonomous Underwater Vehicles (AUV), 2012 IEEE/OES*, pages 1–9. IEEE.